Natively-Heterogeneous Design of Distributed Computing Systems

Ali Mokhtari

A Dissertation presented to the Graduate Faculty
in Partial Fulfillment of the Requirements for the Degree
Doctor of Philosophy

University of Louisiana at Lafayette
Fall 2023

APPROVED:

Martin Margala, Chair
School of Computing and Informatics

Mohsen Amini Salehi, Co-chair
Associate Professor
University of North Texas

Sheng Chen
School of Computing and Informatics

Li Chen
School of Computing and Informatics

Mary Farmer-Kaiser
Dean of the Graduate School

Abstract

Heterogeneity is an indispensable part of modern distributed computing systems. With the ubiquity of accelerators (*e.g.,* GPUs and TPUs) and domain-specific computing (through ASICs and FPGA), the matter of heterogeneity and harnessing it has become a more critical challenge than ever to deal with. In fact, considering heterogeneity in devising middleware-level solutions (*e.g.,* resource allocation) is decisive in optimizing the system performance objective (*e.g.,* QoS, incurred cost, and energy consumption). Accordingly, our goal in this dissertation is to take a holistic approach to heterogeneity and harness it as a result of multiple cooperation between middleware solutions. The core contribution of this dissertation is to propose a "performance-driven heterogeneity measure" that can characterize the impact of the heterogeneity level of a system on its performance objective (QoS) and make the system comparable with its counterparts. In particular, we develop a mathematical model based on statistical measures to characterize a heterogeneous system in terms of its task and machine heterogeneity dimensions, and then reduce it to a single value that represents the execution time behavior of the entire system. Performance evaluations across various simulated and real-world heterogeneous systems demonstrate that the mathematical model can accurately characterize the performance behavior of these systems. Such a heterogeneity measure is instrumental for solution architects to proactively configure their systems to be sufficiently heterogeneous such that they can meet their desired performance objectives. Particularly, for a large heterogeneous

configuration space, such as those offered by public clouds, the proposed heterogeneity measure can be instrumental to proactively configuring a heterogeneous system (instead of trial and error) with respect to the desired QoS and without examining the workload. Moreover, this dissertation develops heterogeneity-aware solutions at the admission control and scheduling levels. In particular, we propose a fair energy- and latency-aware scheduler that considers heterogeneity in the computing system to efficiently assign tasks to heterogeneous machines. To capture uncertainties in the execution and arrival times of tasks, we devise an admission control that proactively drops tasks to maintain the robustness of the heterogeneous computing system against uncertainties.

To my parents, Kourosh and Parvin, my loving wife, Yalda, and my adored son, Hossein, whose love brightens my every day, and in loving memory of my dear brother, Mohammad.

Acknowledgments

I would like to express my appreciation to Professor. Martin Margala for his invaluable support, particularly in his role as the chair of my Ph.D. committee. Furthermore, Dr. Amini's contributions to my research and publications have been noteworthy, and I appreciate his involvement in my academic journey. I wish to express my appreciation to Dr. Pooyan Jamshidi and extend my heartfelt gratitude to Saeid Ghafouri for his indispensable assistance. I would like to thank the members of my dissertation committee, Dr. Li Chen and Dr. Sheng Chen. In addition, I appreciate the support and contributions of SM Zobaed, Razin Farhan Hussein, and Chavit Denninnart to my research. The successful completion of this thesis owes a great deal to the unwavering support and guidance of the Center for Advanced Computer Studies and the Graduate School at the University of Louisiana at Lafayette. Additionally, it is worth noting that this research has been supported by the National Science Foundation under award# CNS-2007209.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Overview

Heterogeneity has been an indispensable aspect of distributed computing throughout the history of these systems. In the modern era, as Moore's law is losing momentum due to the power density and heat dissipation limitations [2, 3], heterogeneous computing systems have attracted even more attention to overcome the slowdown in Moore's law and fulfilling the desire for higher performance in various types of distributed systems. Nowadays, the footprint of heterogeneity can be traced in all forms of distributed systems.

Hyperscaler cloud providers, such as AWS and Microsoft Azure, offer and operate based on a wide range of "machine types" ranging from general-purpose X86 and ARM machines to FPGAs and accelerators. Cloud users can take advantage of this heterogeneity to mitigate their cloud expenditure and to improve QoS. For instance, Amazon SageMaker [4] operates based on heterogeneous Cloud machines to build, train, and deploy machine learning (ML) models. It is reported that the training of ResNet-50 Neural Network model on a heterogeneous cluster with GPUs (`ml.g5.xlarge`) and compute-optimized (`ml.c5n.2xlarge`) machines yields 13% lower cost than on a homogeneous cluster with only `ml.g5.xlarge` GPUs [5].

In the context of Edge computing, domain-specific accelerators (ASICs and FPGA) and general-purpose processors are commonly used in tandem to perform near-data real-time processing. For instance, Google smartglasses Enterprise Edition 2 [6] is equipped with a System-on-Chip (SoC) that includes a multi-core

ARM CPU, a GPU, and a Qualcomm AI Engine to provide onboard computer vision [6, 7].

In the HPC context, deploying architecturally heterogeneous machines has become prevalent to fulfill the power and performance desires [8]. As just one example, HPE Cray EX architecture combines third-generation AMD EPYC CPUs (optimized for HPC and AI) with AMD Instinct 250X accelerators [9].

The heterogeneity of Computing systems can be broadly categorized as consistent or inconsistent [10, 11]. *Consistent heterogeneity* describes a computing system of multiple machines with the same architecture, but different performance characteristics (*e.g.,* different clock speeds). In a consistent heterogeneity, if machine "A" is faster than machine "B" for task 1, it is also faster for all other tasks. The inconsistent heterogeneity describes a system where each task may have different execution times on different machines of the system. Formally, an *inconsistently heterogeneous* system is defined as a computing system in which machine "A" may be faster than machine "B" for task 1 but slower than other machines for task 2 [12].

As an example, ESP32 is a powerful microcontroller chip that is widely used in the development of IoT applications. It has built-in Wi-Fi and Bluetooth connectivity and is equipped with a powerful dual-core processor [13]. The Himax WE-I is a low-power, ultra-small AI processor developed by Himax Technologies. It is designed for running AI algorithms on edge devices, such as wearables, smart home devices, and surveillance cameras [14]. While ESP32 is faster at executing visual wake word datasets compared to the Himax WE-I, but the Himax WE-I

performs better when executing audio wake word datasets. Thus, employing ESP32 and Himax WE-I devices for processing visual and audio wake word datasets forms an inconsistent heterogeneous Edge system.

With the ubiquity of accelerators (*e.g.,* GPUs and TPUs) and domain-specific computing (via ASICs [15] and FPGA [16]), the matter of inconsistent heterogeneity and harnessing it has become a more critical challenge than ever before to deal with. In fact, Heterogeneity plays a key role in improving various performance objectives of distributed systems, such as cost, energy consumption, and QoS. That is why harnessing system heterogeneity has been a longstanding goal in distributed systems. Accordingly, our goal in this dissertation is to take a holistic approach to heterogeneity and harness it as a result of multiple cooperation between middleware solutions. In particular, we theoretically and empirically investigate the impact of heterogeneity on the system performance and devise system-level solutions (*e.g.,* task scheduling, cloud elasticity) that consider heterogeneity to improve the overall system-level objectives (*e.g.,* QoS, cost, or energy consumption).

We formally define a *heterogeneous computing system* as a set of architecturally diverse machines that work together to complete a set of task requests (a.k.a. *tasks*) with different computational requirements. We categorize the tasks arriving in a system based on the type of operation they perform and call them *task types*. For instance, in a system that assists blind and visually impaired people (*e.g.,* [17, 18]), the task types can be obstacle detection, face recognition, and

speech recognition. Moreover, we classify machines of a computing system based on their architectural and performance characteristics and call each one a *machine type*. For instance, a cloud solution architect can form a virtual compute cluster using ARM, x86-based, and GPU machine types. In this work, we consider that the system heterogeneity emanates from the diversity in machine types and computational requirements of task types. That is, the system heterogeneity has two dimensions: (i) *machine heterogeneity*; and (ii) *task heterogeneity*. Variations in the performance (a.k.a. execution time) of a given task type across all the machine types are defined as machine heterogeneity, whereas, variations in the execution time of different task types on a given machine type are represented by the task heterogeneity. Then, the *system heterogeneity* is defined as the compounded heterogeneity of these two dimensions. Thus, for a holistic understanding of how heterogeneity impacts system performance, it is essential to consider both task heterogeneity and machine heterogeneity dimensions, however, the task heterogeneity dimension has been overlooked in many prior works.

1.2 Motivation

Prior research works in this regard have predominantly aimed at optimizing a certain QoS metric (*e.g.,* latency constraint) with respect to the heterogeneity of underlying computing systems. Examples of such works are those for task scheduling [17, 19, 20], load balancing [21, 22, 23, 24], and cloud elasticity [25, 26]. Nonetheless, to the best of our knowledge, *there is no holistic approach that natively deals with the matter of heterogeneity in modern distributed systems.* In particular,

there is no concrete study on the performance of these works upon changing the degree (level) of heterogeneity in the underlying system. That is, the impact of the same system-level solutions (*e.g.,* scheduling methods) across computing systems with various degrees of heterogeneity is unknown.

For instance, in the specific area of task scheduling, numerous heterogeneity-aware scheduling methods (*e.g.,* MinMin [27, 28], weighted Max-Min Fairness [29], SoonestDeadline [19], MaxUrgency [12]) have been developed and widely used. However, our evaluations, shown in Figure 1.1, express that their QoS (on-time completion rate) is not universal and can vary across systems with various degrees of heterogeneity. That is, the same scheduling methods operating on the same input workload can lead to different QoS when applied to two systems with various degrees of heterogeneity. In Figure 1.1, we examined several scheduling methods, namely Min-Min (MM), Min Completion-Soonest Deadline (MSD), Min Completion-Max Urgency (MMU), Min Expected Completion Time (MECT), Min Expected Execution Time (MEET) [19] and First Come First Serve (FCFS), on the same workload against two systems A and B both with four machines, but different heterogeneity levels, as described in Tables 1.1 and 1.2. We observe that the comparative behavior of the scheduling methods varies upon changing the system heterogeneity. For example, MSD outperforms FCFS in the system with heterogeneity level A, but the opposite happens in the system with heterogeneity level B. Also, MECT outperforms other scheduling methods in system A, but that does not hold in system B. These observations indicate that to attain the maximum

performance of heterogeneous computing systems, we need to first be able to *measure* its heterogeneity, and then configure it with the most performant scheduling method accordingly.

Table 1.1. Heterogeneous system A is represented by a matrix whose entries show time units as the expected execution time (EET) of four task types (T_1—T_4) across four machine types (A_1—A_4).

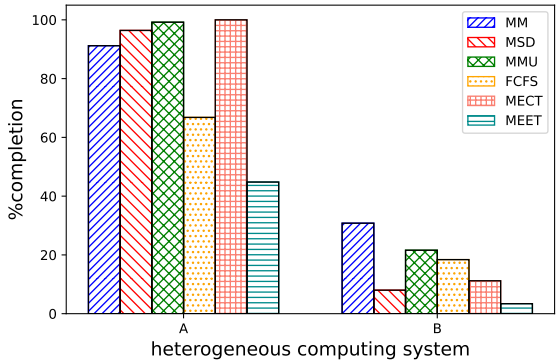
Machines / Task Types	A_1	A_2	A_3	A_4
T_1	0.598	0.739	1.612	4.510
T_2	0.655	0.809	1.765	4.937
T_3	0.686	0.847	1.848	5.171
T_4	1.326	1.639	3.575	10

Table 1.2. Heterogeneous system B is represented by a matrix whose entries include the expected execution time (EET) of four task types (T_1—T_4) across four machine types (B_1—B_4).

Machines / Task Types	B_1	B_2	B_3	B_4
T_1	2.218	2.702	5.692	20.15
T_2	2.448	2.982	6.281	22.24
T_3	2.576	3.138	6.610	23.40
T_4	5.504	6.704	14.122	50

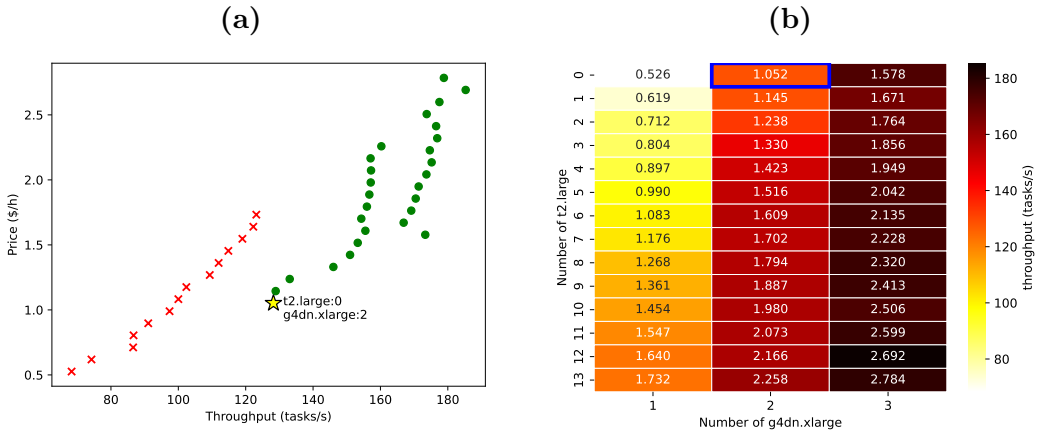
In the context of cloud elasticity, for a given cloud-based application, solution architects need to know the implications of modifying (*i.e.,* adding, removing, or replacing) the allocated machines on the application performance in terms of throughput and cost incurred. The architects have to answer questions like "For a certain arriving workload pattern, what type of machine(s) must be allocated to the existing machines so that the user-defined performance objective (*e.g.,* cost

Figure 1.1. Performance comparison of the same set of scheduling methods with the same workload across two computing systems, A and B, with different levels of heterogeneity (horizontal axis). The vertical axis shows the percentage of tasks completed on time.



and throughput) are satisfied?" They need the ability to *compare* performances of the different (potentially) heterogeneous systems in terms of their performance objectives (*e.g.,* incurred costs and throughput constraint) prior to committing their allocation decisions. Figure 1.2 shows the throughput of different heterogeneous systems, featuring varying numbers of AWS EC2 instances of `t2.large` and `g4dn.xlarge` types, along with the corresponding incurred cost. In Figure 1.2a, configurations (*i.e.,* a certain number of `t2.large` and `g4dn.xlarge` instances) that meet the throughput constraint (*i.e.,* 125 tasks/sec) are shown in green, and those that are not satisfying the throughput constraint are in red. While there are multiple viable configurations that meet the throughput constraint, there is an optimal configuration that minimizes the incurred cost. Figure 1.2b also shows the number of instances with the corresponding price. These observations indicate that to minimize the incurred cost of different heterogeneous computing systems, we need to be able to quantify the impact of heterogeneity on the performance metric (*e.g.,*

Figure 1.2. Illustrating the optimal system configuration that minimizes the cost while meeting the throughput target for a workload of 1000 tasks of speech recognition, image classification, object detection, and question-answering types. Different numbers of AWS EC2 instances of `t2.large` and `g4dn.xlarge` types are used to form heterogeneous system configurations. In Figure 1.2a, the green points represent the system configurations that satisfy the throughput target (>125 tasks/sec), and the red ones violate this constraint. The golden star point is the optimal configuration that meets the throughput target with minimum cost. In Figure 1.2b, numbers in each cell show the cost of each configuration, and colors illustrate the throughput. Darker red represents higher throughput. The cell with a blue frame is the optimal configuration.



throughput), then we can configure a heterogeneous system with the minimum cost. Consequently, in this study, we introduce a novel heterogeneity measure aimed at making the performance (*i.e.,* throughput) of heterogeneous systems predictable. Performance predictability empowers solution architects to estimate the throughput of various system configurations offline. Consequently, they can identify the optimal configuration that minimizes cost while meeting the throughput target.

1.3 Goals and Problem Statement

The influence of system heterogeneity on the system-level solutions calls for a holistic approach that natively considers heterogeneity in all aspects of middleware for modern distributed systems. Even though heterogeneous computing has been

extensively investigated in the past, there is yet to be a concrete approach to harness heterogeneity in a computing system. As such, the overarching **goal** of this dissertation is to take a holistic approach to heterogeneity and harness it as a result of multiple cooperation between middleware solutions. In particular, this dissertation develops heterogeneity-aware solutions at the admission control and scheduling levels. *The core contribution of this dissertation is to propose a "performance-driven heterogeneity measure" that can characterize the impact of the heterogeneity level of a system on its performance objective (QoS) and make the system comparable with its counterparts.*

Figure 1.3 shows our holistic approach to considering heterogeneity in different parts of modern distributed systems. First, we devise an admission control to maintain the robustness of the heterogeneous computing system against uncertainties in tasks' execution times and arrival times. Then, we propose a fair energy- and latency-aware scheduler that considers heterogeneity in the computing system to efficiently assign tasks to the heterogeneous machines. Lastly, we develop a performance-driven heterogeneity measure that can characterize the impact of the heterogeneity level of a system on its performance behavior (a.k.a. QoS) in terms of makespan.

With the aim of harnessing the heterogeneity to improve the system performance, in this dissertation, we address the following problems:

1. How to develop a proactive task-dropping mechanism as an admission control to make the heterogeneous computing system robust against uncertainties in

tasks' execution times?

2. How to quantify the scheduling fairness across different task types in a heterogeneous computing system?

3. How to leverage the multi-objective analysis to develop a fair latency- and energy-aware heuristic for concurrent task types in the heterogeneous computing system?

4. How to provide a measure to quantify the system heterogeneity such that it can be used to determine the overall QoS of the system for a given workload?

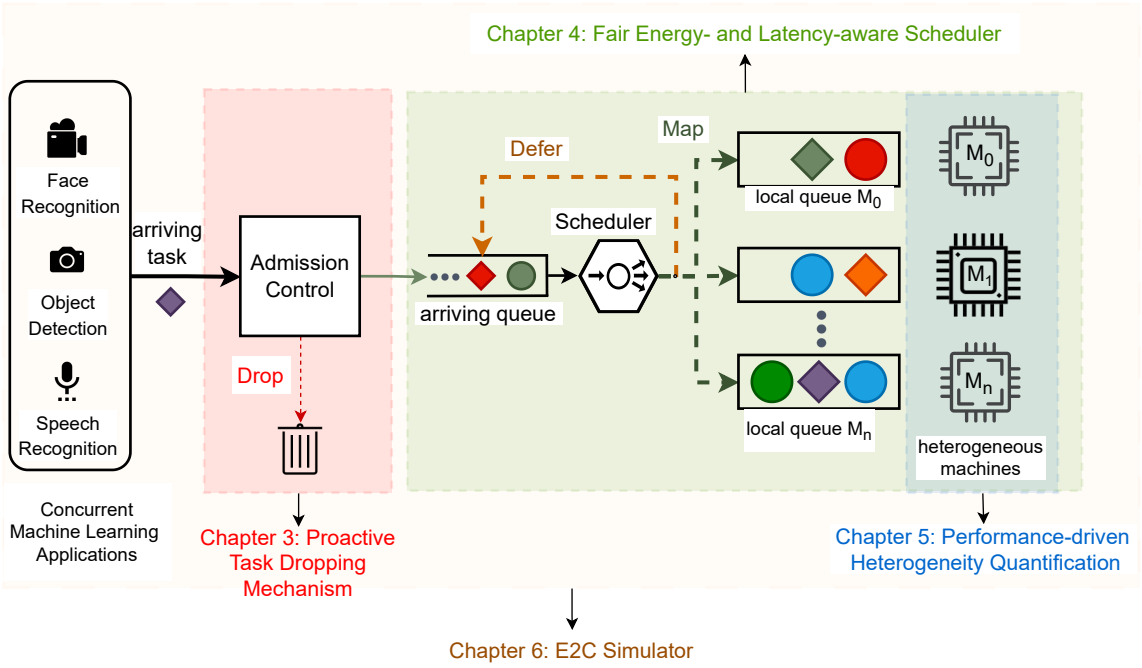
1.4 Contributions and Dissertation Organization

Figure 1.3 depicts the relationships between chapters and the contribution to which they are related to. The core chapters of this dissertation are derived from several research papers published during the course of the PhD candidacy [17, 30, 19, 18].

- Chapter 2 provides background for heterogeneity-aware and fair task scheduling in modern distributed computing systems and explores related research works.

- Chapter 3 studies the robustness of distributed computing systems against uncertainties in tasks' execution time and arrival time. In this chapter, we consider task execution time as a random variable and use probabilistic analysis to develop an autonomous proactive task-dropping mechanism to

Figure 1.3. Illustration of inter-relationship between chapters and contributions



attain our robustness goal. Experimental results demonstrate that the autonomous proactive dropping mechanism can improve the system robustness by up to 20%.

– **Ali Mokhtari**, Chavit Denninnart, and Mohsen Amini Salehi, *Autonomous task dropping mechanism to achieve robustness in heterogeneous computing systems*, in 29th Heterogeneity in Computing Workshop (HCW 2020), in the Proceedings of the IPDPS 2020 Workshops PhD Forum (IPDPSW), 2020, pp. 17–26, © 2020 IEEE.

- Chapter 4 explores fairness in the scheduling of latency-sensitive and concurrent Machine Learning (ML) applications on battery-powered heterogeneous Edge systems. To that end, we investigate edge-friendly

(lightweight) multi-objective mapping heuristics that do not become biased toward a particular application type to achieve the objectives; instead, the heuristics consider "fairness" across the concurrent ML applications in their mapping decisions. Moreover, we study and analyze resource allocation solutions that can increase the on-time task completion rate while considering the energy constraint. Performance evaluations demonstrate that the proposed heuristic outperforms widely used heuristics in heterogeneous systems in terms of latency and energy objectives.

  - **Ali Mokhtari**, Md Abir Hossen, Pooyan Jamshidi, and Mohsen Amini Salehi, *Felare: Fair scheduling of machine learning tasks on heterogeneous edge systems*, in 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), 2022, pp. 459–468, © 2022 IEEE.

- Chapter 5 studies developing a "performance-driven heterogeneity measure" that can characterize the impact of the heterogeneity level of a system on its performance behavior (a.k.a. QoS) in terms of makespan. Performance evaluations across various simulated and real-world heterogeneous systems demonstrate that our proposed mathematical model can accurately characterize the performance behavior of these systems. Particularly, the results show that our proposed heterogeneity measure is able to predict the true makespan of heterogeneous systems without online evaluations with an average accuracy of 84%.

  - **Ali Mokhtari**, Saeid Ghafouri, Pooyan Jamshidi, Mohsen Amini Salehi, *HEET: A Heterogeneity Measure to Quantify the Difference across Distributed Computing Systems*, unpublished manuscript.

- Chapter 6 explains an open-source simulation tool, called E2C, that can help students researchers, and practitioners to study any type of heterogeneous (or homogeneous) computing system and measure its performance under various system configurations. E2C is equipped with an intuitive graphical user interface (GUI) that enables its users to easily examine system-level solutions (scheduling, load balancing, scalability, etc.) in a controlled environment within a short time and at no cost. In particular, E2C is a discrete event simulator that offers the following features: (i) simulating a heterogeneous computing system; (ii) implementing a newly developed scheduling method and plugging it into the system, and (iii) measuring energy consumption and other output-related metrics.

  - **Ali Mokhtari**, Drake Rawls, Tony Huynh, Jeremiah Green, and Mohsen Amini Salehi, *E2C: A Visual Simulator to Reinforce Education of Heterogeneous Computing Systems*, in 13th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar), May 2023,

Chapter 2: Background and Literature Study

This chapter provides background and a survey of other research works undertaken in the fields most related to heterogeneous computing systems.

2.1 Prior Literature for Admission Control in Heterogeneous Computing Systems

In spite of substantial exploration of uncertainty in different areas, ranging from biology to economics, it has not yet been sufficiently explored in the distributed computing literature. The majority of current studies in scheduling assume a static deterministic execution environment [31, 32] or consider predictable and stable performance for distributed computing environments [33, 34, 32]. In practice, these assumptions do not hold. Even in the case of clouds that guarantee certain characteristics (*e.g.,* processor speed and memory capacity) for their services, the actual performance is subject to several underlying factors, such as multi-tenancy, that cause uncertainty. To offer robustness, uncertainty and dynamic performance variations, inherent to heterogeneous and shared infrastructures [35], must be captured.

Optimal task mapping in HC systems and in the presence of uncertain (stochastic) parameters has shown to be an NP-complete problem [36]. Therefore, a large body of research works has been undertaken to capture the stochastic behavior and provide a near-optimal task mapping to fulfill various performance goals (*e.g.,* minimizing average waiting time [37] and maximizing throughput [38, 39]).

With respect to capturing uncertainty in tasks' execution time, Aupy

et al.[40] treat tasks' execution time as a random variable and use probabilistic distributions to model the uncertainty. With the goal of minimizing the incurred cost of using cloud-based reservations, they leverage their proposed strategy to allocate an optimal reservation sequence and schedule tasks on the reserved resources.

Shestak *et al.* [41] investigate and prepare a foundation work for stochastic task execution time modeling using probability mass function (PMF). They establish fundamental tools for the system that use PMF instead of scalar values for task scheduling. Our work builds on top of their findings, adopts their PMF modeling, calculates tasks' completion time based on the convolution of PMFs, and measures robustness in a similar way to their work.

Khemka *et al.* [42] design and evaluate four resource allocation heuristics in oversubscribed HC systems. These heuristics include the use of different utility functions based on urgency, priority, and utility class. Although they utilize PMF-based task execution times, they treat tasks' execution time in a deterministic (*i.e.,* not probabilistic) manner. Their approaches include the use of preemptive task-dropping procedure (*i.e.,* discard task before reaching its deadline). However, their approach relied on a static threshold and only drop tasks, if the task's utility goes below the specified threshold.

Salehi *et al.*[12] mathematically model the impact of task dropping on the completion time PMF of tasks in an HC system. However, task dropping is carried out either based on a static threshold or in a reactive manner (*i.e.,* after a task

misses its deadline). Later, Gentry *et al.* [11] extend the earlier study and presented a task pruning mechanism for HC systems. Denninnart *et al.* [43], show a generalized form of the pruning mechanism and deploy it as a separate component in the system to improve the robustness of homogeneous or heterogeneous systems. The generalized pruning mechanism can work in conjunction with any mapping heuristic to improve the system's robustness. Nonetheless, in all of these works, probabilistic task pruning makes its decisions based on a predefined threshold, which is not necessarily optimal and requires user intervention. Alternatively, the dropping mechanism of this study is optimal and autonomous, *i.e.,* it does not require any predefined threshold and/or user intervention.

2.2 Prior Literature for Fair Energy- and Latency-aware Task Scheduling in Heterogeneous Computing Systems

ML applications are computationally intensive, and deploying them in resource-limited HEC systems would raise two critical challenges: (i) latency and (ii) energy consumption [44, 45, 46]. Prior research efforts have addressed these challenges in two ways: deploying approximate computing techniques and proposing efficient resource allocation algorithms.

Approximate computing techniques can be used to improve the latency and energy consumption of ML applications [47, 48, 49]. Approximate computing, in particular, improves the latency and energy consumption of computationally intensive applications by leveraging their error resistance [50]. In hardware-level approximate computing, the hardware components are modified such that the

computations are performed with less energy consumption and lower accuracy. In [49, 51], the researchers implemented approximate adders and multipliers in deep learning accelerators to improve the performance of the accelerator and save energy. Quantization is another approximate computing technique that can be used to reduce the latency and energy consumption of deep learning applications. In integer quantization, the 32-bit floating-point numbers (*e.g.,* weights) are converted to 8-bit fixed-point numbers to shrink the model, and consequently reduce the latency[52]. In [53], the authors suggested a quantization approach that allows inference to be performed using only integer arithmetic. Their proposed quantization technique could reduce the model size by 4x. In [54], the authors introduce a layer-wise and per-parameter quantization method that could maintain accuracy while the energy consumption and model size are decreased.

The optimal scheduling is proven to be an NP-hard problem [55, 56], thereby, a substantial exploration has been accomplished to propose feasible heuristic-based or sub-optimal solutions. The majority of the proposed solutions in scheduling focuses on one or two objectives like energy [57, 58, 59], makespan [60], or QoS [61]. In [62] a bio-inspired approach was proposed to solve the bi-objective optimization problem for the system makespan and the energy consumption objectives. In [63], the authors employed linear weighted sum techniques to minimize both energy and makespan. In [30, 19], the probabilistic approach is used to determine the probability of on-time completion of tasks on available computing resources. Then, a task-dropping mechanism is deployed to maximize the system's robustness. The

dropping decisions are made such that the overall system performance would increase.

The optimal scheduling and approximate computing techniques are complementary solutions. In fact, the synergy between approximate computing and efficient scheduling can significantly increase the overall system performance. In this work, we propose an efficient resource allocation algorithm that aims to minimize the system's wasted energy due to the unsuccessful completion of tasks while the performance metric is maintained. Furthermore, the proposed method is developed first to detect the unfairness across task types, and then it treats the suffered task types until the fairness criteria are satisfied.

2.3 Prior Literature for Performance-driven Heterogeneity Quantification

Heterogeneous computing systems utilize various computing machines to perform diverse tasks with different computational requirements. The idea of exploiting the system heterogeneity to improve the performance of the system considering different objectives such as energy [64, 65, 66, 67, 68, 17] and QoS [69, 70, 30, 19] has been extensively explored in the literature. Based on these works, it is proven that heterogeneity can play a crucial role in enhancing different system performance metrics, however, these works fall short of providing a concrete metric that can explain the impact of heterogeneity on system performance. Moreover, these works commonly try to exploit heterogeneity in favor of optimizing a performance metric. In contrast, our work takes a different approach such that, for a desired performance metric, it tries to configure the heterogeneity of the

system and its impact on the performance metric.

Expected Time to Compute (ETC) Matrix The idea of characterizing a heterogeneous computing system using the Expected Time to Compute (ETC) matrix was first explored by Ali et al. [1]. They used the coefficient-of-variation of expected execution times as a measure of heterogeneity. Then, they suggested an algorithm that takes the mean and standard deviation of execution times to generate the ETC matrix of the heterogeneous system. However, their method neither characterizes the performance of different heterogeneous systems, nor make them comparable. In contrast, we present a mathematical model to measure the system heterogeneity such that it can characterize the overall system performance behavior and make different systems comparable.

Heterogeneity-aware Task Scheduling Several research works have been undertaken on heterogeneity-aware scheduling algorithms operating based on the ETC matrix. Panda et al. [65] introduced an energy-efficient task scheduling algorithm, called ETSA, for heterogeneous cloud computing to minimize energy consumption and makespan (*i.e.,* the total time to complete a workload). In the proposed algorithm, the trade-off between minimizing energy consumption and maximizing the system performance is achieved by minimizing the linear combination of normalized completion time and total utilization. In their work, the heterogeneity is represented by the Expected Time to Compute (ETC) matrix whose entries show the expected execution time of a particular task type on a specific machine type. Although the proposed scheduling solution has been

evaluated across different heterogeneous computing systems, the impact of the heterogeneity on the system performance has not been discussed. In another work, Panda and Jana [71] presented three scheduling algorithms for heterogeneous multi-cloud environments while the objectives are minimizing makespan and maximizing cloud utilization. Here, the heterogeneous cloud environment is also represented by the ETC matrix. Moreover, Mokhtari *et al.* [17], leveraged the ETC matrix to model the performance behavior of heterogeneous computing systems. They introduced a fair and energy-aware scheduling algorithm, called FELARE, for latency-sensitive tasks in heterogeneous edge systems. To improve fairness across task types, FELARE monitors the performance of the heterogeneous system, and based on the defined fairness metric, it mitigates the suffered tasks by prioritizing them in the next mapping events.

In [20, 19, 30, 12], the authors followed a probabilistic approach to design a robust resource allocation algorithm for heterogeneous computing systems. To capture the uncertainties in the execution times, they introduced the Probabilistic Execution Time (PET) matrix to maintain the Probability Mass Function (PMF) of the execution time of different task types on different machine types in the heterogeneous system. Then, they employed probabilistic convolution techniques to calculate the probabilistic completion time of tasks and, subsequently, make scheduling decisions. In these works, a modified version of the ETC matrix that contains the distribution of execution time for each task type on a machine type was employed to model the heterogeneity. Narayanan *et al.* [29] proposed a throughput

matrix to model the performance behavior of the system. Specifically, each entry (i, j) of the matrix represents the throughput of job i on machine j. The matrix also implies the system heterogeneity, hence, they leveraged it to devise a heterogeneity-aware scheduling method that can be optimized for different performance metrics.

Heterogeneity-aware Machine Learning Inference Serving Systems Several research works have been conducted to devise heterogeneity-aware machine learning inference services considering performance objectives such as cost, QoS, or throughput. Ribbon [72] exploits the cost and performance trade-off in serving deep learning inferences on heterogeneous AWS EC2 instances. They considered different system configurations (different numbers of AWS EC2 instances) as decision parameters. In their work, they assume that the performance metric (*i.e.,* QoS) of diverse configurations cannot be described mathematically. Thus, they employed Bayesian Optimization techniques to find the optimal configuration that minimizes the cost of serving the inference queries while meeting the QoS constraint. Cocktail [73] leverages model ensembling of heterogeneous model variants alongside heterogeneity in the hardware for accuracy improvement and cost optimization. Kairos [74] is a deep learning inference serving framework that maximizes the throughput under cost budget and QoS constraint. For that purpose, they proposed a heterogeneity-aware query distribution mechanism that maximizes the throughput. The core idea is to distribute queries across machines such that it maximizes the idle time in the future for all instances. In these works, it is assumed

that the performance (*i.e.,* throughput) of a heterogeneous system cannot be mathematically described, however, in our work, we propose an analytical approach to accurately estimate the throughput of a heterogeneous system. This would help in finding the optimal configuration that minimizes the cost of service while meeting the throughput target.

In sum, all these works testify that heterogeneity can be employed to improve the system's performance. Nonetheless, there is yet to be a concrete way to characterize the impact of heterogeneity on the system performance. For that purpose, we provide a heterogeneity measure that characterizes the impact of heterogeneity on the performance of heterogeneous systems and can be used to estimate the performance (*i.e.,* throughput or makespan) of heterogeneous systems without online evaluations.

Chapter 3: Admission Control via Proactive Task Dropping Mechanism

3.1 overview

The robustness of a distributed computing system is defined as the ability to maintain its performance in the presence of uncertain parameters. Uncertainty is a key problem in heterogeneous (and even homogeneous) distributed computing systems that perturb system robustness. Notably, the performance of these systems is perturbed by uncertainty in both task execution time and arrival. Accordingly, our goal is to make the system robust against these uncertainties. Considering task execution time as a random variable, we use probabilistic analysis to develop an autonomous proactive task-dropping mechanism to attain our robustness goal. Specifically, we provide a mathematical model that identifies the optimality of a task-dropping decision, so that the system robustness is maximized. Then, we leverage the mathematical model to develop a task-dropping heuristic that achieves system robustness within a feasible time complexity. Although the proposed model is generic and can be applied to any distributed system, we concentrate on heterogeneous computing (HC) systems that have a higher degree of exposure to uncertainty than homogeneous systems. Experimental results demonstrate that the autonomous proactive dropping mechanism can improve the system robustness by up to 20%.

3.2 Problem Statement

Heterogeneous Computing (HC) systems can be categorized as consistent or inconsistent [10, 11] heterogeneous systems. Consistent machine heterogeneity

describes a computing system of multiple machines with the same architecture but different performance characteristics. In an inconsistent HC system, machines are also distinguished by their different architectures [75, 76, 77]. In such a system, each task may have different execution times on different machines of the system. Formally, an *inconsistently heterogeneous* system is defined as a computing system in which machine A may be faster than machine B for task 1 but slower than other machines for task 2 [12]. As a popular example of an inconsistent HC system, we can consider Amazon cloud [78] that offers various machine types (*e.g.,* CPU-Optimized, Memory-Optimized, and GPU).

In the same way, task requests can be categorized as consistently or inconsistently heterogeneous. For instance, a system dedicated for video transcoding [79] receives categorically different tasks (*i.e.,* task types) to change video resolution, bit rate, or compression formats [10]. Each instance of these task types can process a video with a different size, which represents consistent heterogeneity across tasks of the same type. Such variety of tasks are proven to benefit from utilizing an HC system [10].

Robustness of a system is defined as its ability to maintain its performance in the face of uncertainty [12, 80]. Two major uncertain parameters that affect the robustness of a computing system in an inconsistent HC system are, namely task execution time and task arrival [79]. There is uncertainty in the execution times of different task types across different machine types. Uncertainty in tasks' arrival can lead to *oversubscription* situation, which is defined as an overloaded system that

cannot complete all tasks by their deadlines [12].

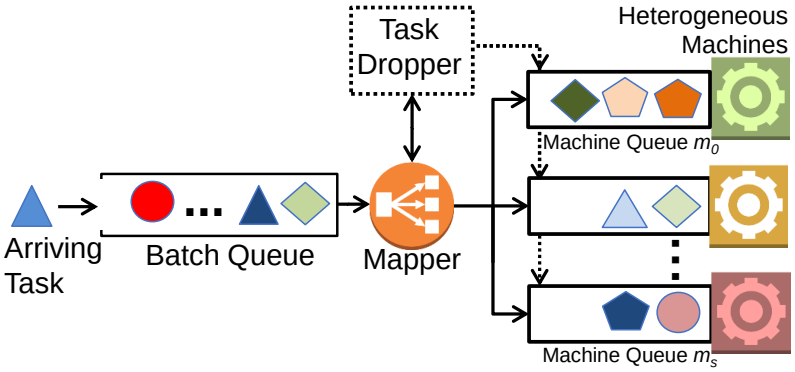
Co-occurrence of both tasks' arrival and execution time uncertainties in a system with inconsistent heterogeneity in their tasks and machines leads to poor resource allocation decisions and lack of robustness [81, 82]. This is particularly crucial when resources are not abundant (*e.g.,* in Edge computing [83]) or the resources cannot be acquired due to budget constraints (*e.g.,* in Cloud environment) [79, 84]. Accordingly, the problem we investigate in this research is: *how to make an inconsistent HC system robust against uncertainties in tasks' execution times and arrival?*

3.3 Solution Statement and Contributions

We address the research question in the context of an HC system used for live video streaming (*e.g.,* [79, 10, 85]). As shown in Figure 4.1, we consider an online (dynamic) batch scheduling system [38] to allocate tasks to heterogeneous machines. Each machine has a limited local queue (termed machine queue) to fetch data for allocated tasks before starting execution. We consider each task in the system as independent and with an individual hard deadline. Then, we measure the robustness of the system based on the number of tasks completed on time within a given time period.

To capture the uncertainty in tasks' execution times, we model the execution times using statistical distributions and leverage them to calculate the likelihood of on-time completion for each task. Also, to capture the uncertainty in tasks' arrival rate, we utilize a task-dropping mechanism that proactively drops (*i.e.,* discards)

Figure 3.1. Overview of a resource allocation system in a heterogeneous edge computing system.



tasks that are unlikely to be completed on time. Smart dropping of unlikely-to-succeed tasks not only reduces the incurred cost of using resources but also increases the chance of success for the remaining tasks and improves the overall system robustness. However, the challenge is in making appropriate task-dropping decisions to achieve the robustness goal. To address this challenge, in this work, we propose a mathematical model that at any mapping event determines the optimal task-dropping decision, so that the overall system robustness is maximized. Next, we leverage the mathematical model to develop a proactive task-dropping heuristic with a feasible time complexity that works along with the mapping heuristic (see Figure 4.1). Although we target HC systems, the proposed model is generic and can improve the robustness of homogeneous systems too.

Prior probabilistic task dropping approaches (*e.g.,* [42, 11, 43]) base their dropping decisions on the chance of completing a task before its deadline (termed *chance of success*) and comparing that against a user-defined threshold. Nonetheless, the dropping threshold is a dynamic parameter depending on

system-level factors, such as task arrival intensity [11]. Such a fine-grained parameter cannot be predetermined and statically applied to the HC system. Alternatively, our proposed dropping mechanism does not rely on any predefined threshold. It can autonomously make optimal dropping decisions such that the overall system robustness is maximized.

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

- Developing a mathematical model for optimal proactive task dropping in an HC system.
- Proposing an autonomous proactive task dropping heuristic in HC systems.
- Analyzing the impact of task-dropping mechanism on the robustness of both heterogeneous and homogeneous systems under varying workload characteristics.
- Analyzing the cost-benefit of using the proactive task-dropping heuristic.

The rest of the chapter is organized as follows: In Section 3.4, we present an overview of the system and our approach. Then, in Section 3.5 we describe our mathematical model and proactive task-dropping heuristic. Next, in Section 3.6, performance evaluation is elaborated. Lastly, the chapter summary is presented in Section 3.7.

3.4 System Model

This research is motivated by an inconsistent Heterogeneous Computing (HC) system used for transcoding live video streaming tasks, such as those explored

in [79, 86, 85]. In this system, each task has an individual deadline and it has to be completed before the deadline. There is no value in executing tasks that have missed their deadlines and such tasks should be dropped to maintain the liveness of the video streaming. In this HC system, a limited number of task types (*e.g.,* transcoding types) are processed. Figure 4.1 shows that arriving tasks are batched in a queue; then each task is mapped to one of the s heterogeneous machines.

There is uncertainty in the execution time of each task type across different machine types. Furthermore, there is uncertainty in the execution time of even one task type on a single machine type, due to factors such as tasks' data sizes and/or resource contention in a multi-tenant system [87]. We consider the uncertain execution time of each task type as a discrete random variable and use a Probability Mass Function (PMF) to model it. Practically, the execution time PMF of task type i on machine type j can be learned and estimated from the historic execution time information of that task type on that machine type. In an HC system, a matrix, called Probabilistic Execution Time (PET) [12], is employed to store the execution time PMFs of all task types on all machine types. Since there are a limited number of known task types and machine types, the PET matrix has a limited size. It is assumed that the PET matrix is available in the HC system.

A mapping event is triggered by completion or arrival of a task to assign unmapped tasks from the batch queue. At each mapping event, first, pending tasks in machine queues that missed their individual deadlines are dropped. Then, Mapper uses a *mapping heuristic* to assign unmapped tasks to available slots in

machine queues. The mapping heuristic creates a temporary queue of machine-task mappings and the *completion time* PMF of each unmapped task on heterogeneous machines is calculated (see Section 3.5.1). Machine queues are to fetch data (*e.g.,* video content) for the assigned tasks, prior to their execution. To restrain the combined effect of execution times uncertainties on a task completion time uncertainty, the size of machine queues are considered to be limited.

We assume that the mapped tasks cannot be remapped, due to the data transfer overhead, and machine queues operate in a first come first serve manner. Similar to [79], tasks are considered to be sequential, independent, and executed in isolation, with no preemption and no multitasking.

Although our model is generic and can be applied to homogeneous systems, in this study, we concentrate on HC systems. The reason is that HC systems have a higher degree of exposure to uncertainty than homogeneous systems. In fact, an inconsistent HC system is not only exposed to uncertainty in the execution time of a certain task type on a given machine type but it is also exposed to the uncertainty of the same task type across different machine types.

3.5 Proactive Task Dropping

Probabilistic task dropping is a double-edged sword for system robustness. On the one hand, we miss the chance to complete a task, hence, it reduces the robustness. On the other hand, dropping improves the chance of success for the tasks behind the dropped task, as they can begin their execution earlier. To attain the maximum robustness, these two effects should be considered for any dropping

decision. In this section, we resolve this issue and determine how task-dropping decision should be made in an HC system so that the robustness is maximized.

In essence, a task should be dropped, if it increases the likelihood of having more tasks completed on time. Therefore, in this section, firstly, we introduce a method to calculate the impact of a task dropping on the chance of success for the remaining tasks. Then, we provide a mathematical model that, at each mapping event, determines the optimal subset of tasks whose dropping can potentially maximize the robustness. Lastly, as the provided model is complex, we leverage it to present a sub-optimal task-dropping heuristic that makes a dropping decision for each individual task, as opposed to collectively considering all tasks.

3.5.1 *Calculating Chance of Success in Reactive Task Dropping*

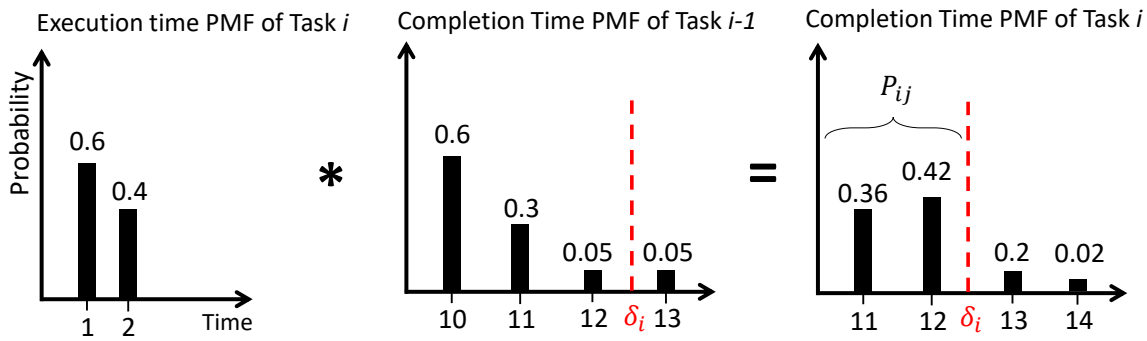
To calculate the chance of success for a task of type i on the local queue of a machine of type j, we first need to determine the stochastic completion time of the task. Recall that, the *execution time* of task type i on machine type j is considered as a discrete random variable, denoted E_{ij}, which is maintained in the form of a PMF in the PET matrix. Let $e_{ij}(t)$ an impulse in the PMF, representing the probability that task type i on machine type j takes t time units to execute (*i.e.,* $e_{ij}(t) = \mathbb{P}(E_{ij} = t)$). Similarly, let C_{ij} be a discrete random variable, representing the *completion time* of task type i on machine j and its PMF is denoted as $c_{ij}(t)$.

As depicted in Figure 3.2, to calculate the completion time PMF of pending task i on the given machine j, its execution time PMF is convolved with the completion time PMF of the task ahead of it (*i.e.,* task $i-1$). Note that, if pending

task i cannot begin its execution before its deadline, denoted δ_i, it is dropped. As this way of task dropping is performed in reaction to missing a task's deadline, we call it *reactive task dropping*. Equation 3.1 shows the way $c_{ij}(t)$ is calculated. In this equation, if the completion time of task $i-1$ occurs at any time after δ_i, task i is reactively dropped, hence, its execution time is considered zero in the convolution process. In this case, $\forall t \geqslant \delta_i$, impulses of $c_{(i-1)j}(t)$ are directly added to $c_{ij}(t)$.

$$c_{ij}(t) = \begin{cases} \sum_{\forall k<t} c_{(i-1)j}(k).e_{ij}(t-k), & t < \delta_i \\ \sum_{\forall k<\delta_i} c_{(i-1)j}(k).e_{ij}(t-k) + c_{(i-1)j}(t), & t \geqslant \delta_i \end{cases} \tag{3.1}$$

Figure 3.2. Execution time PMF of pending task i is convolved with the completion time PMF of task $i-1$ to obtain the completion time PMF of task i.



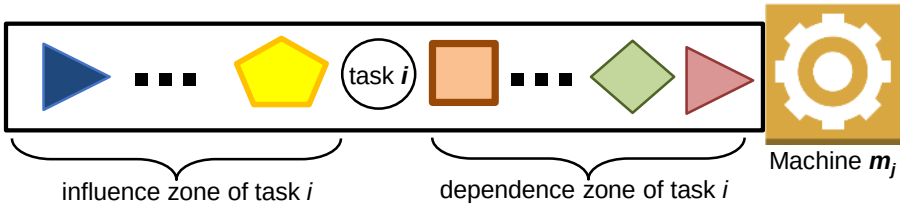
Once we calculate the completion time PMF of task i, its chance of success, denoted p_{ij}, is calculated based on Equation 3.2.

$$p_{ij} = \sum_{\forall t<\delta_i} c_{ij}(t) \tag{3.2}$$

Although task execution time is an independent random variable, task completion time is not. As depicted in Figure 3.3, in a machine queue, the completion time (and subsequently chance of success) of task i not only depends on

its execution time but also on the completion time of the tasks ahead of it, defined as *dependence zone*. Similarly, task i influences the completion time of the tasks behind it in the machine queue, defined as *influence zone*. Note that, upon dropping task i, only its influence zone is affected.

Figure 3.3. Stochastic completion time of task i is dependent on the list of tasks ahead of it (dependence zone). Task i influences the stochastic completion time of tasks behind (influence zone).



3.5.2 *Calculating Chance of Success in Proactive Task Dropping*

In this part, we investigate how predictively deciding to drop a task, known as *proactive task dropping*, can favor the overall system robustness. For that purpose, we need to measure the potential benefit of task dropping on the system's robustness. For a list of q pending tasks in machine queue j, we define *instantaneous robustness*, denoted R_j, as the sum of their chances of success and calculate it based on Equation 3.3. Our hypothesis is that the overall system robustness is likely to be improved, only if instantaneous robustness is improved at each individual mapping event.

$$R_j = \sum_{i=0}^{q} p_{ij} \tag{3.3}$$

Because dropping task i only affects the chance of success for tasks in its

influence zone, dropping task i is considered appropriate, only if it improves the instantaneous robustness of tasks in the influence zone. For task i in the machine queue, we need a method to calculate the instantaneous robustness of its influence zone in two cases: when task i is not dropped versus when it is provisionally dropped.

Upon provisional dropping of task i, the completion time of task $i-1$ is convolved with the execution time of task $i+1$ with respect to its deadline (δ_{i+1}), as explained in Equation 3.1. Let $c^{(i)}_{(i+1)j}(t)$ represent the completion time PMF of task $i+1$ when task i is provisionally dropped. Formally, $c^{(i)}_{(i+1)j}(t)$ is calculated based on Equation 3.4.

$$c^{(i)}_{(i+1)j}(t) = \begin{cases} \sum_{k=0}^{k<t} c_{(i-1)j}(k).e_{(i+1)j}(t-k), & t < \delta_{(i+1)} \\ \sum_{k=0}^{k<\delta_{i+1}} c_{(i-1)j}(k).e_{(i+1)j}(t-k) & \\ +c_{(i-1)j}(t), & t \geqslant \delta_{(i+1)} \end{cases} \tag{3.4}$$

Accordingly, the completion time PMF of the next tasks in the influence zone of task i, $c^{(i)}_{nj}(t)$ for $\forall n \geqslant (i+2)$, is determined using Equation 3.5.

$$c^{(i)}_{nj}(t) = \begin{cases} \sum_{k=0}^{k<t} c^{(i)}_{(n-1)j}(k).e_{nj}(t-k), & t < \delta_n \\ \sum_{k=0}^{k<\delta_n} c^{(i)}_{(n-1)j}(k).e_{nj}(t-k) + c^{(i)}_{(n-1)j}(t), & t \geqslant \delta_n \end{cases} \tag{3.5}$$

Once we have the completion time PMF for task n in the influence zone, its

chance of success, denoted $p_{nj}^{(i)}$, is calculated based on Equation 3.6.

$$p_{nj}^{(i)} = \sum_{t=0}^{t<\delta_n} c_{nj}^{(i)}(t) \tag{3.6}$$

3.5.3 *Optimal Proactive Task Dropping*

Recall that dropping a task has two contradictory effects on the system's robustness. Although it reduces the number of completed tasks by one, it increases the chance of success in its influence zone, and therefore, the instantaneous robustness.

Due to the impact of proactive task dropping on the chance of success of tasks in its influence zone, proactive dropping is not an independent decision to be made for a task in isolation. As an example, assume that task n is located in the influence zone of a large (*i.e.,* compute intensive) task i, such that the chance of success for n tends to zero ($p_{nj} \to 0$). Therefore, instantaneous robustness does not gain from dropping n. However, proactively dropping i can affect the chance of success for n and make it appropriate for dropping. We can conclude that an optimal proactive task dropping must maintain a collective view of the list of tasks of a machine queue, as opposed to deciding for each task in isolation. Thus, the problem of optimal proactive dropping is narrowed down to finding a subset of tasks whose dropping maximizes the instantaneous robustness.

As the influence zone of the last task in a machine queue is null, its dropping does not improve instantaneous robustness, hence, it is excluded from the subset of tasks considered for proactive dropping decisions. Accordingly, in a machine with queue size q, finding the optimal proactive dropping decision requires 2^{q-1} subsets

to be examined for dropping. The subset of tasks whose dropping maximizes the instantaneous robustness represents the optimal proactive dropping decision.

3.5.4 *Proactive Task Dropping Heuristic*

As finding the optimal subset of tasks for proactive dropping includes examining an exponential number of cases, it imposes a considerable overhead at each mapping event. As such, in this part, we propose a task-dropping heuristic that provides a sub-optimal solution within a feasible time. The proposed heuristic does not examine all subsets, instead, it operates on a task-by-task basis and decides about the proactive dropping of each task. Specifically, the heuristic iterates each machine queue and only in one pass decides appropriate tasks for proactive dropping.

The appropriateness of proactively dropping task i can be measured by comparing the instantaneous robustness of machine j when task i is provisionally dropped, denoted $R_j^{(i)}$, versus the circumstance in which task i is not dropped (*i.e.,* R_j). Let Q_j represent the list of pending tasks on machine queue j, then $R_j^{(i)}$ is calculated based on Equation 3.7.

$$R_j^{(i)} = \sum_{\forall n \in Q_j - \{i\}} p_{nj}^{(i)} \tag{3.7}$$

In particular, dropping task i is considered appropriate, if $R_j^{(i)}$ is sufficiently greater than R_j. That is, we should have $R_j^{(i)} > \beta \cdot R_j$, where $\beta \geqslant 1$ is defined as the *robustness improvement factor*. In fact, the value of β dictates the aggression level of proactive task dropping. In spectrum, $\beta \to \infty$ disables proactive task dropping

whereas $\beta \to 1$ enacts dropping even for minor improvements in instantaneous robustness. We study the suitable value for β in the evaluation section of the paper.

Note that, in examining the provisional dropping of task i, only its influence zone has to be considered and the dependence zone of the task can be excluded from the calculations. We argue that there is not much gain in exploring the whole influence zone. Knowing that provisional dropping of task i decreases the instantaneous robustness by p_{ij}. Assuming $\beta = 1$, the gain in the instantaneous robustness of tasks in the influence zone must be greater than p_{ij}, so that proactive dropping of task i is enacted. Theoretically, the gain can occur due to the accumulation of negligible improvements across a large number of tasks that eventually may not increase the system's robustness. To avoid dropping because of such misleading gains, in this heuristic, we enact proactive dropping of task i, if the loss in the instantaneous robustness is compensated only within the first few tasks of the influence zone.

For proactive task dropping heuristic, we define *effective depth*, denoted η, as the number of tasks located immediately after task i in its influence zone. Then, robustness improvement is only examined for tasks $n \in < i+1, ..., i+\eta >$. In summary, the proactive dropping of task i on machine j is confirmed by the heuristic, only if the condition in Equation 3.8 holds.

$$R_j^{(i)} > \beta \cdot R_j \iff \sum_{n=i+1}^{i+\eta} p_{nj}^{(i)} > \beta \cdot \sum_{n=i}^{i+\eta} p_{nj} \tag{3.8}$$

The algorithm in Figure 3.4 explains the proactive task-dropping heuristic.

In the first step, the algorithm iterates through all machine queues and performs reactive task-dropping for those that already missed their deadlines (as noted in Step 2). Then, in Steps 4—9, for each task i, we examine provisionally dropping it and compare the instantaneous robustness for the effective depth of task i with the circumstance that task i is not dropped. In Step 9, task i is proactively dropped if the condition in Equation 3.8 holds. The mapping heuristic is invoked after the proactive dropping heuristic.

Figure 3.4. Pseudo-code for Proactive Task Dropping Heuristic.

$\beta \leftarrow$ Robustness Improvement Factor
$\eta \leftarrow$ Effective Depth

Upon triggering of a mapping event:

(1) For each queue j of machines $\{m_0, m_1, ..., m_s\}$:

(2) Drop all pending tasks that missed their deadlines

(3) For each task i in machine queue j:

(4) For each task n in effective depth of task i:

(5) Calculate p_{nj} based on Equation 3.2

(6) Provisionally drop task i

(7) Calculate $p_{nj}^{(i)}$ based on Equation 3.6

(8) if $\sum_{n=i+1}^{i+\eta} p_{nj}^{(i)} > \beta \cdot \sum_{n=i}^{i+\eta} p_{nj}^{(i)}$

(9) Confirm dropping of task i

(10) Call mapping heuristic

3.5.5 *Complexity Analysis of Proactive Task Dropping*

The time complexity of proactive task dropping in each mapping event depends on two factors: (A) the number of convolutions; and (B) the complexity of

performing each convolution.

As noted earlier, the number of cases that the optimal dropping examines is 2^{q-1}, and for each case, at most q tasks are considered. Hence, in the worst case, the number of convolutions required (factor A) for the optimal solution is $O(q \cdot 2^{q-1})$. Alternatively, heuristic dropping approximates optimal dropping by iterating the machine queue from the head to the tail only once, evaluating the impact of dropping each task on η tasks in its influence zone. Therefore, it requires at most $O(\eta \cdot q)$ convolutions.

Let N_1 and N_2 the set of impulses of two given PMFs. In the worst case, we assume that the PMFs are such that the number of impulses in the convolved PMF is $|N_1| \cdot |N_2|$. Then, the time complexity of the convolution operation (factor B) is $O(N^2)$, where $N = max(|N_1|, |N_2|)$. Accordingly, calculating the completion time of all tasks in a machine queue with size q has a time complexity of $O(N^q)$ where $N = max_{i=1}^{q}(|N_i|)$. As a result, the overall time complexity of the proactive task-dropping heuristic is $O(q \cdot N^q)$. Note that, in practice, the value of q is low and, based on our observations, the number of impulses generated by a convolution is far less than $|N_1| \cdot |N_2|$.

3.6 Performance Evaluation and Analysis

3.6.1 *Experimental Setup*

To evaluate the task-dropping mechanism, we simulate two scenarios: one using four video transcoding as task types and four AWS cloud virtual machines (VM) as the HC system. To study the mechanism further, we simulate a more

diverse HC system with eight machines and twelve task types from SPECint [88] benchmarks. We base our analysis on the latter workload because it provides a wide variety of inconsistent heterogeneous workloads. Then, we use the cloud-based workload for validation of the findings.

The eight machines in the latter scenario contain eight machines[a]. The function describing the execution time of the tasks on a machine is assumed to be an unimodal distribution. Gamma distribution was used to generate the distributions and the mean of the Gamma distribution was determined based on execution time results of SPECint benchmarks on the aforementioned eight machines. We sampled 500 execution times for each application on each machine where the scale parameter of each Gamma distribution was chosen uniformly from the range [1,20]. Once the sample execution times were generated, we applied a histogram to discretize the result and produce PMFs. The PMFs of different benchmarks on the eight heterogeneous machines collectively form the PET matrix.

PET matrix of the eight machines by twelve task types is used throughout the experiments. Each machine is provided with a machine queue which can store up to six tasks, including the task that is currently executing. Task dropping mechanism is engaged each time a system notices a task missing its deadline. All the experiments are performed on Louisiana Optical Network Infrastructure (LONI) Queen Bee 2 HPC system [89].

[a]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.

Each simulation starts and ends when the system is in an idle state. In a simulation, each task arrives based on an arrival time and eventually oversubscribes the system. As we focus on the oversubscribed condition, the first and last 100 tasks in each workload trial are excluded from the results. For each experiment, 30 workload trials with the same intensity level were examined. Workload intensity refers to the number of tasks per time unit arrives in the system. For each experimental result, the mean and 95% confidence interval are reported.

Every workload trial introduces a level of oversubscription to the system, such that all tasks cannot be completed successfully, due to a shortage of resources. However, every single task is individually feasible to process on time. The deadline for any given task i is determined based on $\delta_i = arr_i + avg_i + (\gamma \cdot avg_{all})$, where arr_i is the arrival time, avg_i is the mean execution time for the task type (range from 50 to 200 ms), γ is a coefficient determining the task slack, and avg_{all} is the mean of all task types execution times. To evaluate the system's robustness against task arrival uncertainty, we conduct all experiments with three levels of task arrival intensity, creating workloads with 20K, 30K, and 40K tasks.

3.6.2 *Mapping Heuristics*

The dropping mechanism introduced in this paper is generic and independent from any particular mapping heuristic. In fact, the dropping mechanism can be considered as a separate component in a resource allocation system that can cooperate with any mapping heuristic, such as those widely used in heterogeneous systems (*e.g.,* MinMin [12], MSD [79], and PAM [11]) or homogeneous systems

(*e.g.,* FCFS, SJF, and EDF), to improve the system robustness.

MinCompletion-MinCompletion (MinMin or MM): MinMin (MM) is a popular mapping heuristic in heterogeneous computing literature [90, 12]. In the first phase of this heuristic, for each task in the batch queue, the machine that offers the minimum expected completion time is found, and a pair is formed. In the second phase, for each machine with an available slot in its queue, from the task-machine pairs provisionally mapped to that machine, the pair with the minimum completion time is assigned to it. The process is repeated until all machine queues are full, or until the batch queue is depleted.

MinCompletion-Soonest Deadline (MSD): Similar to MinMin, MSD is also a two-phase mapping heuristic used in several earlier studies (*e.g.,* [42, 12, 11]). The first phase creates task-machine pairs based on the minimum expected completion time for each unmapped task. In the second phase, for each machine with a free slot, the task-machine pair that has the soonest deadline is assigned to that machine.

Ties are broken by choosing the task that has the minimum expected completion time. Similar to MM, after assigning tasks to free slots, the operation is repeated until either there is no unmapped task or there is no free slot in machine queues.

Pruning-Aware Mapping (PAM): PAM [11] is a state-of-the-art heuristic function based on the PET matrix and operates based on the chance of success for tasks. PAM is a two-phase mapping heuristic. In its first phase, for each

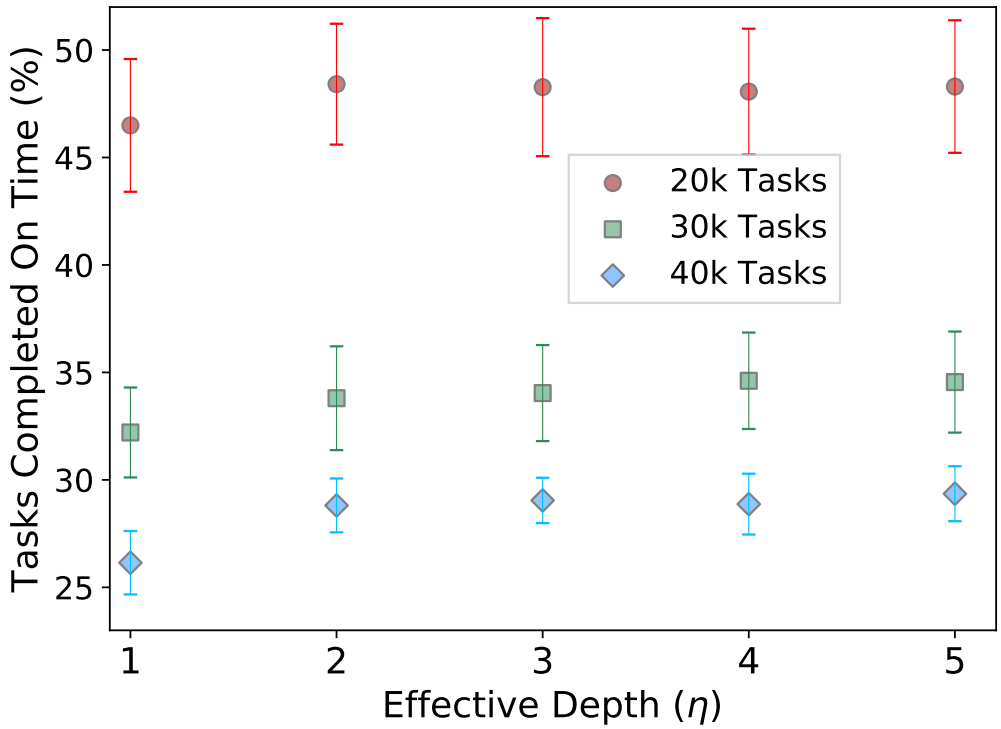
task, it finds the machine provides the highest chance of success. Then, the second phase finds the task-machine pair with the lowest completion time and maps it to that machine queue. Ties are broken by assigning the task that has the shortest expected execution time. PAM performs task dropping (from machine queues) and task deferring (from the batch queue) at each mapping event. However, because this study focuses on the dropping operation, for the sake of comparison, we disabled deferring on PAM.

PAM uses a predetermined threshold for dropping and deferring decisions. We replace the dropping thresholds of PAM with our proposed proactive dropping mechanism. Specifically, we consider two separate cases for evaluation: (A) Combination of PAM with optimal proactive task dropping (shown as $PAM + Optimal$); (B) Combination of PAM with heuristic proactive task dropping (shown as $PAM + Heuristic$).

3.6.3 *Analyzing the Impact of Effective Depth*

In Section 3.5.4, we described that the proactive task-dropping heuristic does not need to examine the whole influence zone of a task to decide about its dropping. In this part, we aim to identify the suitable number of tasks in the influence zone (*i.e.,* effective depth), whose robustness improvement should compensate for the loss of robustness resulting from a task dropping. For that purpose, we analyze how the robustness of an HC system differs by varying the values of effective depth (η). The result of this analysis is shown in Figure 3.5. The horizontal axis shows different values of effective depth and the vertical axis shows the system's robustness in the

Figure 3.5. The impact of varying effective depth on the system robustness resulted from proactive task dropping heuristic with PAM mapping heuristic. The horizontal axis shows the effective depth (η) and the vertical axis shows the system robustness in the form of the percentage of tasks completed on time.



form of the percentage of tasks completed on time. The experiment was conducted for three oversubscription levels.

As shown in Figure 3.5, there is no significant improvement in the system robustness for $\eta > 2$. The reasons are twofold: First, considering too many tasks for effective depth can be misleading to the task-dropping heuristic. This is because the robustness loss resulting from dropping task i can be potentially amortized across multiple tasks in the influence zone, causing a slight (but practically ineffective) improvement in their chances of success. In this circumstance, the task-dropping heuristic malfunctions by suggesting dropping task i, without necessarily improving

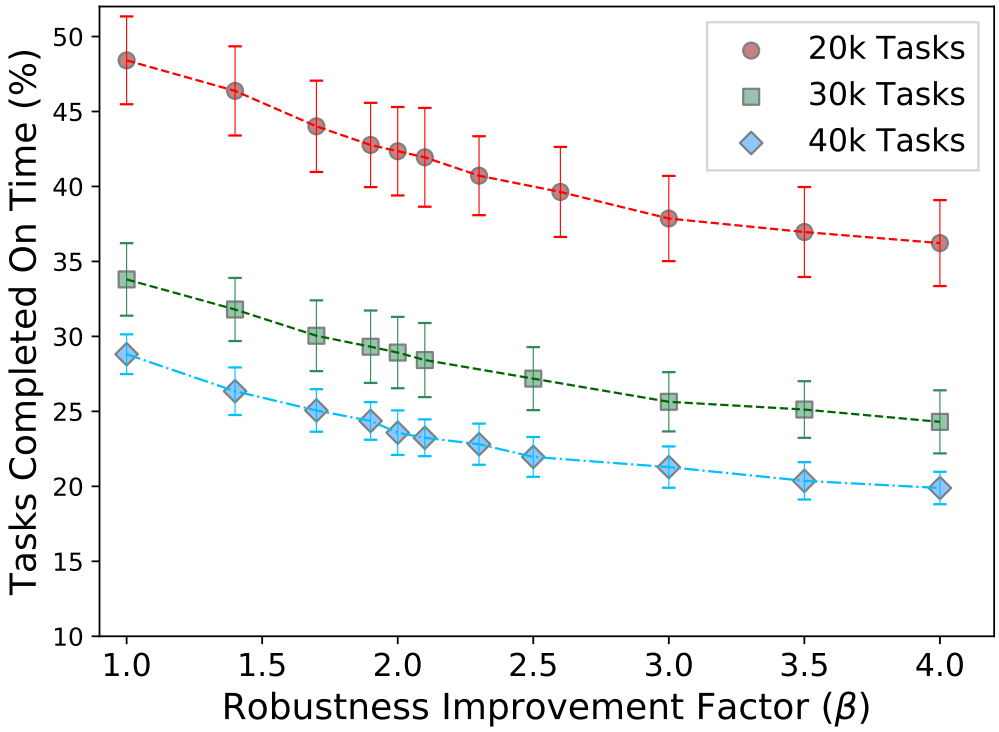
the number of tasks completed on time. This observation confirms our hypothesis in Section 3.5.4Second, from a probabilistic point of view, when we drop task i in an oversubscribed system, the uncertainty exists in the completion time of tasks located immediately after task i gradually absorbs the gain of robustness resulting from dropping task i. We can conclude that, in an oversubscribed system, the impact of dropping task i fades out quickly, within the first couple of tasks in the influence zone of task i.

Although the above justification suggests effective depth to be small, in Figure 3.5, we observe that an effective depth of 1 is not effective. In fact, the case of $\eta = 1$ can be misleading in certain circumstances. For example, consider task i is unlikely to succeed (say $p_{ij} = 10\%$), therefore, it is provisionally dropped. However, task $i + 1$ is already likely to succeed (say $p_{ij} = 95\%$) and provisionally dropping i can improve the chance of task $i + 1$ by at most 5%. Because the robustness improvement cannot compensate for the loss of it (which is 10% by dropping task i), dropping heuristic decides not to drop task i. However, because $\eta = 1$, the heuristic neglect considering task $i + 2$ in the influence zone that can potentially gain significantly from dropping task i. According to this analysis, for the rest of the evaluations, we configure the proactive mapping heuristic to be carried out with $\eta = 2$.

3.6.4 *Analyzing the Impact of Robustness Improvement Factor*

As we described in Section 3.5.4, the proactive task-dropping heuristic decides the appropriateness of a task-dropping based on a Robustness Improvement

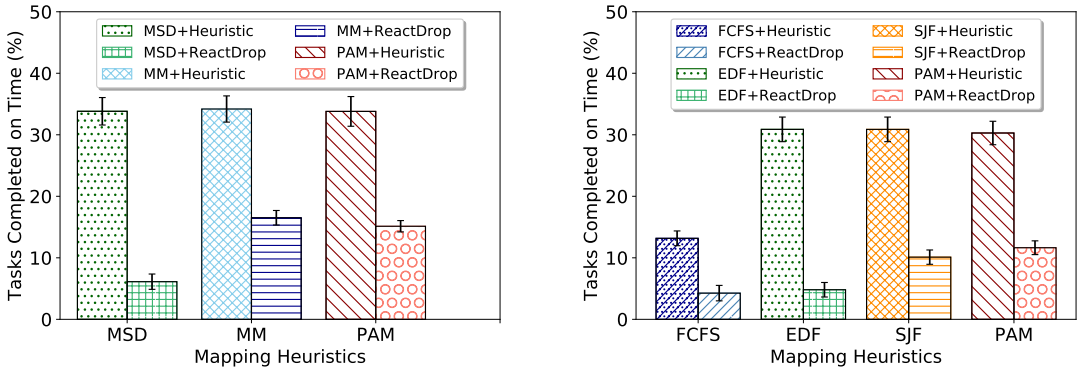
Figure 3.6. The impact of the Robustness Improvement Factor (β in the horizontal axis) on the system robustness resulted from proactive task dropping heuristic with PAM mapping heuristic for different oversubscription levels.



Factor (β). In this part, we experimentally identify the suitable value that should be considered for β, so that the system robustness gain is maximized. To this end, as shown in Figure 3.6, we vary the value of β in the range of [1,4] by step 0.5 and, for each configuration, we measure the system robustness in the form of the percentage of tasks completed on time. We conducted the experiment for all three levels of oversubscription.

As we can see in this figure, the system robustness is maximized for $\beta = 1$ and the system robustness declines, as the β value increases. In fact, by increasing the β value proactive task-dropping heuristic becomes more conservative and is less

Figure 3.7. Evaluating the impact of applying proactive task dropping heuristic to different mapping heuristics. Subfigure (a) shows the results for a heterogeneous computing system and Subfigure (b) shows it in a homogeneous computing system. Horizontal axes show different mapping heuristics—each one deployed with a proactive task dropping heuristic (+Heuristic) and without a proactive task dropping heuristic (+ReactDrop). In each case, system robustness in the form of the percentage of tasks completed on time is reported.



(a) Proactive task dropping in a heterogeneous system

(b) Proactive task dropping in a homogeneous system

often engaged in the task-dropping operation. At the end of the spectrum, very large values for β neutralize the impact of the proactive dropping heuristic. According to this analysis, for the rest of the evaluations, we configure the proactive mapping heuristic to be carried out with $\beta = 1$.

3.6.5 *Analyzing the Impact of Proactive Task Dropping on Various Mapping Heuristics*

Although the proposed task-dropping mechanism is independent of mapping heuristics, the two can have a synergy in achieving robustness against compound uncertainty. To examine both the generality of the dropping mechanism and its impact on the system robustness, in this experiment, we apply the proactive task dropping heuristic on widely-used mapping heuristics of both heterogeneous and

homogeneous systems. Then, for each mapping heuristic, we measure the system robustness (percentage of tasks completed on time) with the proactive task dropping heuristic (+Heuristic) and without the proactive task dropping heuristic involved (+ReactDrop). In this experiment, the oversubscription level of the system is set on 30K tasks.

The results of this experiment are shown in Figure 3.7. Subfigure 3.7a shows the percentage of tasks completed on time (vertical axis) and its horizontal axis shows MSD, MM, and PAM mapping heuristics, each one with and without proactive task dropping heuristic. In this figure, we observe that when proactive task dropping is not applied, MSD performs significantly lower than MM and PAM. This is because in an oversubscribed system, mapping tasks based on their deadline intensity implies allocating tasks with a low chance of success and postponing tasks that have a high chance of success to a later time. However, we observe that when proactive task dropping is in place, all three mapping heuristics provide almost the same robustness. This is because proactive task-dropping prunes tasks whose chance of success is low from machine queues. Interestingly, the results show that, if we put a reasonable dropping mechanism in place, we do not have to deploy a complex mapping heuristic. In this case, simple mapping heuristics can be forgiven for their poor mapping decisions and ultimately provide competitive robustness.

The result of this experiment for homogeneous mapping heuristics is shown in Figure 3.7b. In this experiment, we employed three mapping heuristics that are popular in homogeneous systems, namely FCFS, SJF, and EDF (earliest deadline

first), and a prior work's mapping heuristic named PAM. The figure testifies that the dropping mechanism can significantly improve the robustness of homogeneous systems. We observe that, without dropping, FCFS and EDF provide the lowest robustness. The reason that SJF and PAM provide better robustness is that SJF always maps the shortest tasks and, hence, can increase the number of completed tasks. Also, PAM always maps the ones with the highest chance which leads to completing tasks on time. Similar to heterogeneous systems, we observe that proactive dropping heuristics can compensate for poor decisions made by mapping heuristics and increasing their robustness to almost the same magnitude. The improvement in robustness is less significant for FCFS. This is because, unlike SJF, in FCFS, executing a compute-intensive task can diminish the chance of success for several pending tasks, such that even by proactively dropping them the chance of success for remaining tasks does not improve significantly.

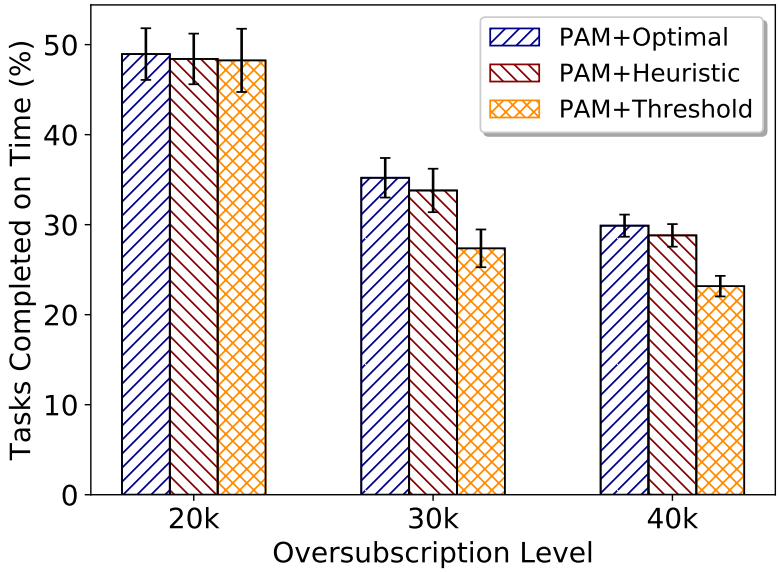
3.6.6 Analyzing the Impact of Proactive Task Dropping on the System Robustness

In this experiment, our goal is to evaluate how proactive dropping can enhance the system's robustness against compound uncertainty in both task execution times and arrival. Based on the previous experiment, we pick PAM as the mapping heuristic for this study and apply the following four variations of task dropping on it: (A) using optimal proactive dropping (termed PAM+Optimal); (B) using proactive dropping heuristic (termed PAM+Heuristic); and (C) using a threshold based approach (termed PAM+Threshold). Case (C) was developed in

[11] and in that the system user needs to be aware of dropping and initially set its threshold. Then, the predetermined threshold is adjusted at each mapping event.

Figure 3.8 shows the result of evaluating variations of task dropping across three oversubscription levels, represented by the number of arriving tasks (as shown in the horizontal axis). In each case, we measure the system robustness in the form of the percentage of tasks completed on time (vertical axis).

Figure 3.8. Comparing the impact of proactive task dropping against other forms of task dropping in terms of system robustness, measured by the percentage of tasks completed on time (vertical axis). The experiment is conducted for various oversubscription levels (horizontal axis).



The experiment results demonstrate that as the systems become more oversubscribed, the system robustness declines. However, we observe that both PAM+Optimal and PAM+Heuristic outperform PAM+Threshold. Specifically, when the system is under 40K task arrival, both PAM+Optimal and PAM+Heuristic outperform PAM+Threshold by around 8%. The results indicate

the efficacy of the proactive dropping approaches. This improvement is particularly remarkable when considering that proactive dropping is also less complicated than PAM+Threshold and it does not require any user involvement in adjusting the dropping threshold.

Further analysis between PAM+Optimal and PAM+Heuristic reveals that, regardless of the oversubscription level, there is no statistically and practically significant difference between these two approaches. Considering the simplicity and competitive performance of PAM+heuristic, we can conclude that it can replace PAM+Optimal without any major loss in robustness.

To analyze the impact of proactive task dropping on the observed robustness, we need to know the percentage of tasks dropped reactively (upon missing the deadline) and proactively. Our analysis shows that after applying the proactive task-dropping mechanism, only around 7% of the task droppings happen reactively. This indicates that proactive task dropping is remarkably effective in avoiding resource wastage and allocates tasks to machines, only if they can complete on time. Proactively dropping tasks with a low chance of success offers a higher chance and certainty of success to the remaining tasks, hence, improving the system robustness.

3.6.7 *Analysis of the Incurred Cost of using Resources*

While the focus of this paper is to maximize the system robustness in an HC system, there are other metrics of success to consider; one of these is cost. Time consumed for computing tasks that eventually fail to be completed on time is a resource wastage that for certain scenarios, such as cloud computing, has associated

costs. As such, the aim of this experiment is to analyze the impact of the proactive task-dropping heuristic on the incurred cost of using such resources. For that purpose, pricing from Amazon cloud [78] was mapped to the simulation machines. To create a normalized view of the incurred costs, the price incurred to process the tasks is divided by the percentage of tasks completed on time. We conduct this experiment for various oversubscription levels.

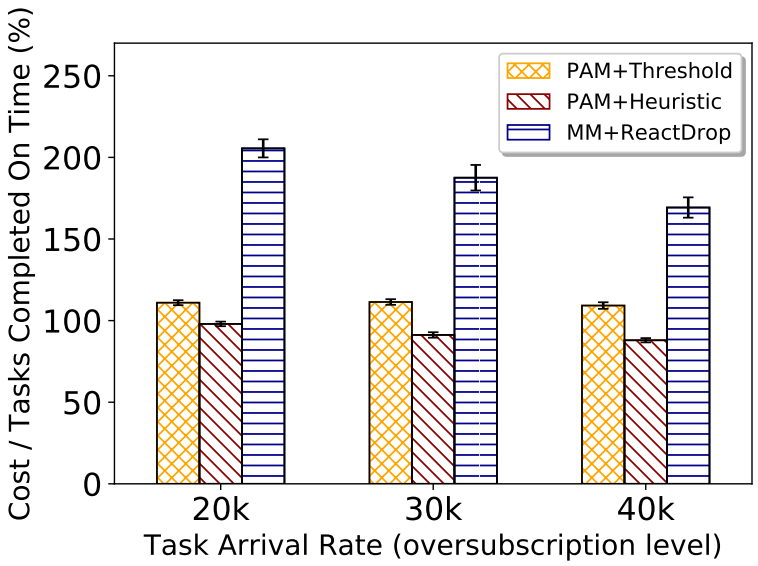
Figure 3.9 shows that in an oversubscribed system, both PAM+Threshold and PAM+Heuristic incur a significantly ($\simeq$50%) lower cost per completed task than MM. In particular, the reason for the improvement in PAM+Heuristic is prioritizing tasks that are most likely to succeed. The significance of this experiment is showing the fact that PAM+Heuristic not only outperforms other dropping-based methods in terms of robustness, but it also performs that with a lower incurred cost, because of not processing tasks needlessly.

3.6.8 *Validating Robustness for Video Transcoding Workload*

To validate our earlier observations, we utilize video transcoding workload traces to measure the impact of the proactive dropping heuristic on the system robustness. The video workload includes four video transcoding (task) types on four heterogeneous machine types (two machines for each type). Execution time variation across different task types is high (*i.e.,* certain task type takes significantly shorter time to execute than the others across all machine types). These video workload traces also have a lower arrival rate and the system is moderately oversubscribed.

The results, shown in Figure 3.10, confirm our earlier observations that

Figure 3.9. The impact of the proactive task dropping on incurred costs of using resources. The horizontal axis shows the oversubscription level.

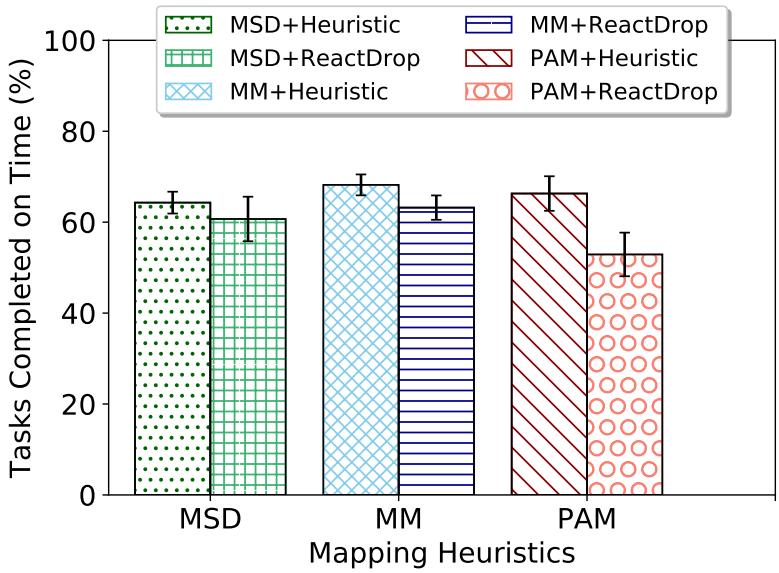


applying the proactive task-dropping heuristic improves the system's robustness, regardless of the mapping heuristic deployed in the system. Further, we observe that when proactive task dropping is plugged into the system, all mapping heuristics exhibit almost the same robustness, which again validates our observations in the earlier experiments.

3.7 Summary

In this chapter, we investigated the robustness of HC systems against the compound uncertainty resulting from both uncertain task execution times and uncertain task arrivals. To attain the robustness goal, we proposed an autonomous dropping mechanism that captures the compound uncertainty and proactively drops tasks whose chance of success is low, to increase the chance of success for the remaining tasks, hence, maximizing the overall system robustness. The dropping

Figure 3.10. The impact of proactive task dropping applied on the video transcoding workload using different mapping heuristics. The oversubscription level is 20k tasks.



mechanism uses a mathematical model to determine the optimal task-dropping decisions in a dynamic resource allocation system. We then utilized the mathematical model and proposed a sub-optimal task-dropping heuristic that provides nearly the same robustness as the optimal one. Experimental results show that the proactive task-dropping heuristic not only improves the system robustness in both heterogeneous and homogeneous systems by around 20%, but also reduces the incurred cost of using resources. In compare to earlier task-dropping works, the proposed proactive task-dropping mechanism provides the following advantages: (A) It is dynamic and does not require user intervention to configure any predetermined threshold; (B) Architecturally, it is less complicated and can cooperate with any mapping heuristic in a resource allocation system; (C) It provides a higher system robustness.

Chapter 4: Fair Scheduling of Machine Learning Tasks on Heterogeneous Systems

4.1 Overview

Edge computing enables smart IoT-based systems via concurrent and continuous execution of latency-sensitive machine learning (ML) applications. These edge-based machine learning systems are often battery-powered (i.e., energy-limited). They use heterogeneous resources with diverse computing performance (e.g., CPU, GPU, and/or FPGA) to fulfill the latency constraints of ML applications. The challenge is to allocate user requests for different ML applications on the Heterogeneous Edge Computing Systems (HEC) with respect to both the energy and latency constraints of these systems. To this end, we study and analyze resource allocation solutions that can increase the on-time task completion rate while considering the energy constraint. Importantly, we investigate edge-friendly (lightweight) multi-objective mapping heuristics that do not become biased toward a particular application type to achieve the objectives; instead, the heuristics consider "fairness" across the concurrent ML applications in their mapping decisions. Performance evaluations demonstrate that the proposed heuristic outperforms widely-used heuristics in heterogeneous systems in terms of the latency and energy objectives, particularly, at low to moderate request arrival rates. We observed 8.9% improvement in on-time task completion rate and 12.6% in energy-saving without imposing any significant overhead on the edge system.

4.2 Motivation

IoT-based systems commonly rely on edge computing to process Machine Learning (ML) applications in the user's proximity, thereby, offering low-latency smart services. However, the edge systems are often battery-powered and have a limited energy supply. In many use cases, the IoT-based systems offer multiple smart services to their users (*e.g.,* object detection and motion capture [91]). Therefore, the corresponding edge system needs to handle multiple compute-intensive ML applications simultaneously (*i.e.,* concurrently) and continuously on its limited resources. These limitations justify making use of inconsistently Heterogeneous Edge Computing (*HEC*) systems [12, 30] where processors are architecturally diverse and offer different compute performance and energy consumption for distinct application types.

An exemplar use case of such an IoT-based system is SmartSight [92] (see Figure 4.1), whose aim is to offer ambient perception to the blind and visually impaired people. The system operates based on a pair of smart glasses and a companion edge system that takes advantage of *inconsistently heterogeneous* processing units [12] (a.k.a. machines) in which each machine type has a different energy consumption and is optimized for fast execution of certain task types. For instance, the HEC system can take advantage of both ESP32 (with IVP-EP 32-way SIMD imaging/video data plane processor) [93] and ARC EM9D/EM11D processors [94] that are optimized for image/video processing tasks (*e.g.,* object detection inference), and audio processing tasks (*e.g.,* speech recognition),

respectively. In this system, the companion edge system has to concurrently and continuously execute multiple ML applications and serve the user's requests within a short latency (in less than 100 milliseconds), thereby, offering services such as object detection to identify obstacles, motion detection to identify approaching objects, face recognition, text recognition, and speech recognition to respond the user's commands. Both energy-efficiency and low-latency are critical metrics for the *usability* and dependability of this system and the day-to-day life of the disabled user. As such, any platform for this system must aim at *maximizing the energy-efficiency* and *minimizing the latency* for *all* the offered services.

4.3 Problem Statement

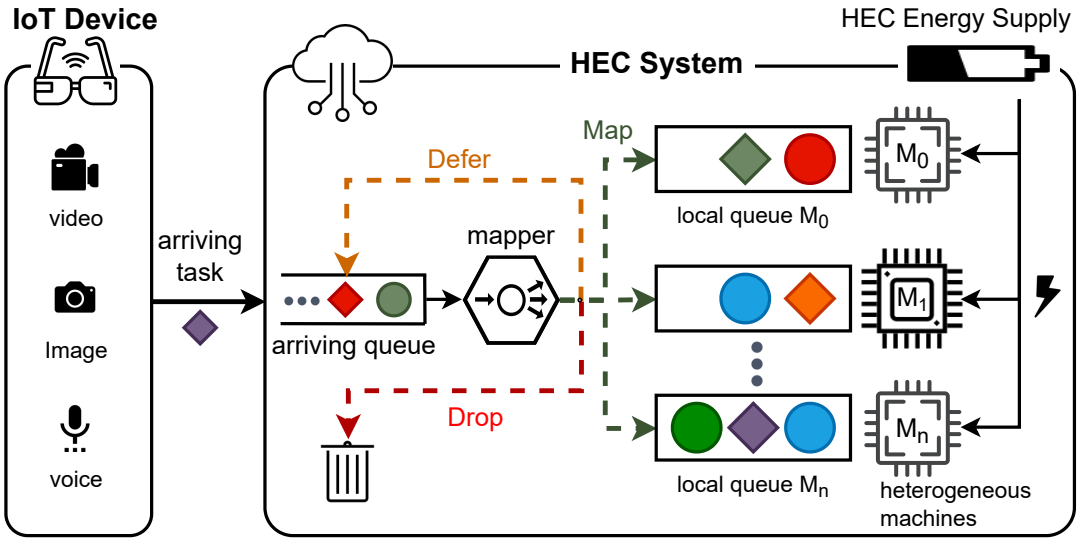
An overview of the HEC system that we consider in this study is shown in Figure 4.1. Multi-modal data (*e.g.,* image, video, voice) are streamed from the sensors of an IoT device and form different types of task requests that dynamically arrive at the HEC system. Subsequently, the mapper module is triggered and allocates the tasks to a set of inconsistently heterogeneous machines. Before executing a task, its data is fetched to the local queue of the machine. Note that the local queues of the machines are limited. The tasks are *independent* and *latency-sensitive*, with *individual hard deadlines*.

In the HEC system, efficiently allocating tasks to the limited resources is decisive on the *latency* and *energy* objectives. Intuitively, resource allocation decisions can minimize energy consumption by mapping user requests to the machine with minimum energy usage (and computing performance). However, such

allocations can potentially undermine the latency objective. In contrast, allocating user requests to the fastest machine (with higher energy consumption) depletes the battery quickly and makes the system unusable. In a system like SmartSight, such an unusability can potentially threaten the user's safety. These extreme cases demonstrate the importance of efficient resource allocation to increase the up-time and usability of an energy-limited HEC system.

The resource allocation decisions must also be *fair* across the concurrent services. That is, the aforementioned objectives cannot be fulfilled by making the system biased toward specific task requests. For instance, to make the system energy-efficient and last for a longer time, a resource allocation method cannot consistently ignore (*i.e.,* drop) the motion detection tasks (that have long execution times) in favor of the object detection ones (that have shorter execution times). In the case of SmartSight, such a biased system makes the blind person incapable of detecting approaching objects, which again undermines its usability. As such, *fairness* across request types is the third objective that has to be considered by the resource allocation of such systems. In sum, the problem we investigate in this chapter can be stated as follows: *how to design a fair resource allocation method for tasks (i.e., requests to concurrent ML applications), such that the on-time task completion rate is maximized within the energy constraint of the HEC system?* It is noteworthy that in such a resource-limited computing system, the resource allocation method solution should be lightweight, and its incurred overhead should not worsen the system performance.

Figure 4.1. Overview of an Edge-based system that offers multiple ML services (*e.g.,* Object detection, speech recognition, and face recognition) concurrently using an energy- and resource-limited Heterogeneous Edge Computing (HEC) system. Sensors of the system capture multi-modal input (*e.g.,* video, image, and voice) and form latency-sensitive task requests.



Resource allocation problem for heterogeneous systems has been widely studied with the objective of either maximizing the on-time task completion [30] or minimizing the energy consumption [95]. Nonetheless, in a usable IoT-based system, both of these objectives in addition to the fairness are desired on a resource- and energy-constraint system.

4.4 Solution Statement and Contributions

In addressing the multi-objective problem, we noticed that often improving the latency objective comes with the cost of energy-inefficiency and vice versa. Hence, we investigate how to efficiently strike a trade-off between the energy and latency objectives. Such a multi-objective resource allocation problem is known to be NP-complete [55]; thus, we develop a spectrum of heuristic solutions that range from latency-aware to energy-aware ones. We note that, in the described HEC

system, executing a task that is unlikely to meet its deadline wastes the available energy for its unsuccessful execution and delays the following tasks, reducing their chance of on-time completion, hence, degrading the system's usability. To mitigate this, prior to the mapping, the mapper should proactively drop tasks that are most probably unable to meet their deadline.

The performance metric for the latency objective is the number of tasks successfully that are completed on-time (*i.e.,* within the latency constraint), and for the energy objective is the amount of energy consumed to execute task requests. Then, we show that using the proposed lightweight mapping heuristic, called *ELARE*, we can meet both the energy and latency objectives that can dominate the widely-used resource allocation heuristics. In the next step, we explore the fairness aspect. For that purpose, we first formally define a measure to quantify the fairness in a HEC system. Next, we leverage the measure to propose a Fair Energy- and Latency-Aware Resource allocation on heterogeneous Edge systems, called *FELARE*, that can maintain fairness across task types while the energy and latency objectives are satisfied. FELARE aims at minimizing the energy consumption for the mapping decisions that are expected to result in successful task completion (*i.e.,* feasible scheduling decisions). FELARE also identifies the suffered task types (*i.e.,* those with the low fairness value) and dynamically adjusts its decisions to mitigate the unfairness.

To evaluate the efficacy of the ELARE and FELARE heuristics, we

conducted a simulation study[a] and noticed that these methods could improve the on-time task completion by 8.9% and reduce the wasted energy due to inefficient task scheduling by 12.6%. These heuristics could also achieve fairness while fulfilling the energy and latency objectives. In sum, the main contributions of this chapter are as follows:

- Developing a measure to quantify the scheduling fairness across different task types in a computing system.
- Performing a multi-objective analysis of the resource allocation in the HEC system.
- Leveraging the multi-objective analysis to develop fair latency- and energy-aware heuristics for concurrent ML applications in the HEC systems.
- Investigating the energy consumption, latency, and fairness resulting from the proposed heuristics against widely used scheduling heuristics in the HEC systems.

Although we conduct this research in the context of machine learning tasks and edge computing, our analysis and findings are not limited to this context and can be adapted to similar contexts where a set of pre-known independent task types are deployed on energy-limited heterogeneous machines.

The rest of this chapter is organized as follows: In Section 4.5, we provide an overview of the system. Then, in Section 4.6, we describe our latency- and

[a]https://github.com/hpcclab/E2C-Sim.git

energy-aware heuristic. In Section 4.7, we propose a measure to quantify fairness and fair resource allocation. Next, in Sections 4.8 and 4.9, the performance evaluation of the methods is described. Lastly, we present the chapter summary in Section 4.10.

4.5 System Model

This chapter encompasses scenarios where a single-user HEC system is employed to host multiple ML applications, such as SmartSight [92] and those explored in[96, 97, 98]. In these scenarios, ML applications are latency-sensitive and have to process the user's requests in a real-time manner. As an example, a request to an obstacle detection application in SmartSight has to detect the objects (obstacles) for a disabled user within a short hard deadline. There is no use in completing a task after the deadline has passed and doing so makes the solution unusable for the disabled user. Moreover, these HEC systems host only a *limited* and pre-known ML applications (a.k.a. task types) with different data modalities (*e.g.,* video, image, or voice) that are assumed to have the same priority. As shown in Figure 4.1, tasks with various modalities are queued upon arrival. Then, the resource allocator (a.k.a. scheduler) uses a mapping heuristic to make one of the following decisions for each queued task: (A) *mapping* it to an available slot in the local queue of a machine; (B) discarding it, via *dropping* it or *deferring* its mapping to a later time. A mapping could happen in two situations: (i) completion of an executing task or (ii) arrival of a new task. The local queue on each machine is to fetch the required data (*e.g.,* image, audio, or video) and prepare the assigned tasks

for execution. Due to uncertainty in the execution time of tasks and its compound impact on the mapping decisions, the local queues are considered to have a limited size, and they are equal across different machines in the system [11, 99]. Furthermore, task requests dynamically arrive to the system and the order of arrival is unknown.

We categorize the arriving tasks based on the ML application they belong to, and call them *task types*. Similarly, heterogeneous machines in the HEC systems are distinguished by their performance characteristics and architectures and are considered as different *machine types*. Profiling the execution time of task types on the machines provides information about the execution time of task type i on machine type j. Then, the expected values of the execution times for all task types on different machine types are utilized to form a matrix, called *Expected Execution Time (EET)* matrix [12]. The number of task types and the number of machine types in the HEC system determine the number of rows and columns of the EET matrix. In this work, we assume that the EET matrix is available via leveraging task profiling data of the HEC system.

To determine the energy consumption of the system, we use the idle and dynamic powers of each machine. Specifically, the amount of energy used by a machine of type j to process a task of type i is determined by multiplying the expected execution time of task type i by the dynamic power of that machine j. For an idle machine, the amount of energy used is determined by multiplying the idle time by its idle power. Due to the data transfer overhead and latency constraint of

the tasks, we assume that a mapped task cannot be remapped or preempted. Machine queues are also served in a first come, first served manner.

4.6 ELARE: Energy- and Latency-aware Task Scheduling in HEC Systems

In the HEC systems, the energy consumption due to processing a latency-sensitive task is wasted in two ways: (i) unsuccessful task completion; and (ii) inefficient resource allocation. The former is caused by mapping a task to a machine where the task misses its deadline (*i.e.,* the expected completion time of the task on that machine is greater than its deadline). The latter explains the extra energy consumed (wasted) by a machine to successfully complete a task that could be otherwise successfully completed on another machine with less consumed energy.

To mitigate energy wastage, we propose a two-phase Latency- and Energy-aware resource allocation method called *ELARE*, to map tasks in the HEC system. To tackle the wasted energy due to unsuccessful task completion, in the first phase, the mapper identifies the unlikely-to-succeed tasks in the arriving queue and defers their assignment to the next mapping events with the hope that a better matching machine would be available at that time. However, it is possible that the task deferral continues in the following mapping events until the task's deadline is violated. In this case, the task is dropped and the system would not further process that task. To mitigate the wasted energy due to inefficient resource allocation, the mapper minimizes the energy consumption in its mapping decisions for the feasible tasks in the arriving queue. Algorithm 1 provides the pseudo-code of the ELARE heuristic. In the following sections, we elaborate on the details of each phase of this

heuristic.

Algorithm 1: ELARE Heuristic

```
   Input: ArrivingQueue: {T_0, T_1, ··· , T_p}
   Machines: {M_0, M_1, ··· , M_m}
   EET: Expected Execution Time Matrix
   Output: Mapping Pairs: A list of [task, machine]
 1 Function EE(ArrivingQueue, Machines, EET)
 2     Call Phase-I with the ArrivingQueue, Machines & EET as inputs to
         generate feasible task-machine pairs,
 3     and the set of infeasible tasks
 4     for each task ∈ infeasible tasks do
 5         if task.deadline < current time then
 6             defer(task)
 7         else
 8             drop(task)
 9     Call Phase-II with feasible task-machine pairs & Machines as inputs
10     return set of task-machine pairs for mapping
```

4.6.1 *Phase-I: Latency- and Energy-Awareness*

Recall that the Phase-I of ELARE is responsible for recognizing the infeasible tasks in the arriving queue. A $[task, machine]$ pair is deemed as *feasible* pair if the machine can successfully complete the task. A task that appears in at least one feasible $[task, machine]$ pair is identified as a feasible task. To determine the feasibility of a $[task, machine]$ pair, the expected completion time of the task on the machine is required. Let task i with deadline δ_i map to slot q of machine j and is given a start time, denoted by $s_{ij(q)}$, when the machine is being idle. The expected execution time of processing task i on machine j is e_{ij} time units, which is extracted from the EET matrix. Thus, the expected completion time of task i when it is mapped to the queue slot q of machine j, denoted by $c_{ij(q)}$, is calculated based

on Equation 4.1:

$$c_{ij(q)} = \begin{cases} s_{ij(q)} + e_{ij}, & s_{ij(q)} + e_{ij} < \delta_i \\ \delta_i, & s_{ij(q)} + e_{ij} > \delta_i \ and \ s_{ij(q)} < \delta_i \\ s_{ij(q)}, & s_{ij(q)} \geqslant \delta_i \end{cases} \tag{4.1}$$

In Equation 4.1, the first row belongs to feasible pairs while two other rows belong to infeasible pairs. In case of missing the deadline, the completion time of the task could be either its deadline (*i.e.,* task is dropped during execution immediately when it passes its deadline) or the start time (*i.e.,* the task is dropped before execution started because the task has already passed its deadline).

In Phase-I, to prevent inefficient scheduling, for each task in the arriving queue, the feasible $[task, machine]$ pair that incurs minimum energy usage is singled out and considered as the feasible and efficient pair for that task. To that end, the expected energy consumption for executing task i when it is mapped to queue slot q of machine j is determined as follows:

$$ec_{ij} = \begin{cases} p_j^{dyn} \cdot \ (\delta_i - s_{ij(q)}), & c_{ij(q)} > \delta_i \ and \ s_{ij(q)} < \delta_i \\ p_j^{dyn} \cdot \ e_{ij}, & c_{ij(q)} < \delta_i \\ 0, & s_{ij(q)} \geqslant \delta_i \end{cases} \tag{4.2}$$

In Equation 4.2, the first row describes the wasted energy due to the unsuccessful completion of the task. In the case of successful task completion, the middle term of Equation 4.2 gives the amount of energy consumed by the machine. However, this energy consumption is not always the optimal value for completing the task. In other words, a scheduler that makes an inefficient scheduling decision would result in higher energy consumption, thus, wasting energy. In the case that a feasible task appears in multiple $[task, machine]$ pairs, all the pairs except the one

with the minimum energy consumption are considered as "inefficient pairs" that acting upon them increase the energy consumption, thus, reducing the usability of the HEC system. To avoid this energy wastage, for each task, its feasible pair with minimum expected energy consumption (efficient feasible pair) is selected in Phase-I.

The pseudo-code for Phase-I is shown in Algorithm 2. Lines 6-11 generate a list of feasible machines for each task in the arriving queue and their corresponding Expected Energy Consumption. In Line 13, the feasible machine with minimum expected energy consumption (EEC), ec_{ij}, is selected. Then, in Line 14, task T_i and its matching machine with minimum EEC (*i.e.,* efficient and feasible machine for completing the task) create a pair, and it is appended to the list of feasible efficient pairs. In Line 16, a task that has no chance of being completed by any available machines is considered an infeasible task and stored in the list with the same name.

4.6.2 *Phase–II: Minimizing the Energy Consumption*

Recall that the output of Phase-I is the set of feasible and efficient $[Task, Machine]$ pairs. It is possible that multiple tasks become feasible for a machine in the system. To avoid wasting energy due to inefficient resource allocation, Phase-II of the ELARE heuristic is responsible for mapping the feasible pair that incurs the minimum expected energy consumption.

Algorithm 3 shows the pseudo-code for the Phase-II of the ELARE heuristic. In Line 6, all tasks that match Machine j are retrieved. Then, in Line 7, the most efficient task (*i.e.,* with minimum energy consumption) is selected, and appended to the mapping pairs in Line 8. Next, the mapping pairs are returned to the main

Algorithm 2: Phase-I of ELARE Heuristic

Input: ArrivingQueue: $\{T_0,\ T_1,\ \cdots,\ T_p\}$
Machines: $\{M_0,\ M_1,\ \cdots,\ M_n\}$
EET: Expected Execution Time Matrix
Output: Feasible Efficient Pairs, Infeasible Tasks

```
Function Phase-I(ArrivingQueue, Machines, EET)
    for each T_i ∈ ArrivingQueue do
        for each M_j ∈ Machines do
            Calculate c_ij using Equation 4.1
            if c_ij ⩽ δ_i then
                Calculate ec_ij using Equation 4.2
                Append [M_j, ec_ij] to FeasibleMachines
        if FeasibleMachines ≠ NULL then
            M_eff = Select machine with min ec_ij in FeasibleMachines
            Append [T_i , M_eff] to Feasible Efficient Pairs
        else
            Apppend T_i to Infeasible Tasks
    return Feasible Efficient Pairs, InfeasibleTasks
```

Algorithm 3: Phase-II of the ELARE Heuristic

Input: Machines: $\{M_0,\ M_1,\ \cdots,\ M_n\}$,
Feasible Efficient Pairs: A list of $[T_i, M_j, EEC_{ij}]$
Output: Mapping Pairs: A list of [task, machine]

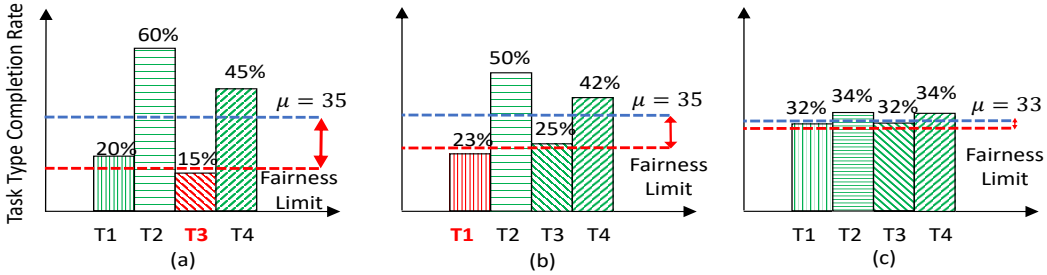
```
Function Phase-II(Machines, Feasible Efficient Pairs)
    for each M_j ∈ Machines do
        Nominees_j ← Call Get_Nominees(Feasible Efficient Pairs, M_j)
        T_i ← Select the task with min ec_ij from Nominees_j
        Append [T_i, M_j] to the list of Mapping Pairs
    return Mapping Pairs
```

ELARE heuristic.

4.7 Fairness in Completing Task Types

A resource allocation method is deemed fair if it is unbiased in allocating resources to the tasks of the same priority. That is, the resource allocation should not prioritize task types based on their execution time or any other system-level metric other than those explicitly defined by the user. Recall that we assumed no precedence across task types in HEC. Therefore, we can use the successful completion rate of different task types as the metric to measure the fairness across all task types. The completion rate of task type i, denoted cr_i, represents the portion of the tasks of type i that are successfully completed within a given time interval. In other words, task type completion rate is the ratio of the number of completed tasks on-time for a certain task type to the total number of tasks of that type arrived to the system. In an ideal and fair resource allocation, the completion rate of all task types is one ($\forall i \; cr_i = 1$). However, due to non-optimal resource allocation or shortage of resources, some tasks may not meet their deadlines, thus, their task type completion rate decreases with missing each task. To quantify fairness, we continuously monitor the task types' completion rates. An observed completion rate distribution could lie in one of the following categories: (i) Co-existence of high and low values for task type completion rates; (ii) Similar but low completion rate values for all task types; or (iii) Similar and high completion rate for all the task types. The first observation describes the situation where the mapping method favors certain task types (with high completion rates) over others

Figure 4.2. Illustration of the fairness limit method at different mapping events. The completion rate variance is diminishing from left to right, as a result of FELARE mapping heuristic. Suffered task types (T1 and T3) are gradually treated by FELARE heuristic.



(with low completion rates). In contrast, the third one represents a mapping heuristic that exhibits fairness across all the task types. Accordingly, improving the fairness in a biased mapping system is translated as moving from category (i) to (iii).

Algorithm 4: Suffered Task Types

Input: TaskTypes: $\{TT_0,\ TT_2,\ \cdots,\ TT_s\}$
Output: Suffered Task Types
1 **Function** `Suffered(TaskTypes)`
2 Calculate μ and σ of the task type completion rates
3 Calculate fairness limit, ϵ using Equation 4.3
4 **for each** $TT_i \in TaskTypes$ **do**
5 cr_i: completion rate of TT_i
6 **if** $cr_i \leqslant \epsilon$ **then**
7 Append TT_i to Suffered Task Types
8 **return** Suffered Task Types

In the case of co-existing high and low completion rates, the task types with low completion rates are the *suffered task types*. To identify the suffered task types, we define a *fairness limit*, denoted ϵ, such that any task type whose completion rate is lower than this limit is considered as suffered task type. To calculate this limit, we introduce the *fairness factor*, denoted f, that represents the aggressiveness of the fairness method. Let μ and σ, respectively, represent the mean and standard

deviation of the completion rates across all task types. Then, the fairness limit is calculated based on Equation 4.3. A higher value for the fairness factor results in a less aggressive fairness method. In the extreme case where f is enough large, the fairness limit approaches zero, thus, does not identify any suffered task types (*i.e.,* the fairness method is disabled).

$$\epsilon = \mu - f \cdot \sigma \quad \text{where } 0 \leqslant f \leqslant \frac{\mu}{\sigma} \tag{4.3}$$

Upon calculating the fairness limit, any task type whose completion rate is below the limit is identified as a member in the set of suffered task types, *i.e.,* $cr_i < \epsilon \iff i \in$ *suffered task types*. Figure 4.2 further clarifies the way to identify the suffered task types using the fairness limit method. In subfigure (a), the completion rate for task types $\{T_1, T_2, T_3, T_4\}$ is 20%, 60%, 15%, and 45%, respectively. To identify the suffered task types, the mean and standard deviation of the completion rates across all task types is calculated; we have $\mu = 35$ and $\sigma = 18.4$. Assuming the fairness factor be one (*i.e.,* $f = 1$), based on Equation 4.3, the value of the fairness limit becomes $\epsilon = 16.6$. Because the completion rate of T_3 is less than the fairness limit ($cr_3 = 15\%$), it is identified as a suffered task type. In the next mapping events, the FELARE method gives T_3 a higher priority, thus, its completion rate increases ($cr_3 = 25\%$). From subfigure (b) to (b), Although the mean of the completion rates does not change ($\mu = 35$), the standard deviation ($\sigma = 11.4$) decreases as a result of applying FELARE mapping heuristic. Therefore, the fairness limit is shrinking and T_1 is identified as a suffered task type ($cr_1 = 23 < 23.6$), thus, higher priority is given to T_1 in the next mapping events.

Eventually, the standard deviation converges to zero and the the gap between the mean and fairness limit diminishes, as depicted in subfigure (c).

Algorithm 4 explains the pseudo-code to identify suffered task types. In Line 4, the mean and standard deviation of the task type completion rates are calculated. Then, in Line 5, the Equation 4.3 is used to determine the fairness limit. Eventually, in Lines 6-9, the completion rate of each task type, cr_i is compared against the fairness limit to identify the suffered task types.

Once we know the suffered task types, we leverage it to make the system fair. For that purpose, we extend the ELARE heuristic and propose a new heuristic, called Fair Energy- and Latency-aware Resource Allocation (FELARE). It follows the following approaches to address fairness:

- Prioritizing the suffered tasks in the mapping events.
- Leveraging task dropping for non-suffered tasks in favor of infeasible suffered tasks to make them feasible.

Using FELARE, in each mapping event, the suffered task types are prioritized in allocation. Moreover, for a suffered task that is infeasible, the pending tasks in the local queue of the fastest (*i.e.,* best-matching) machine are dropped one-at-a-time, until the suffered task becomes feasible on that machine. These two strategies ultimately enhance the completion rate for the suffered tasks and gradually diminish the dispersion in the completion rates of the task types. It is noteworthy that, once the completion rate of task type i becomes greater than the

fairness limit, it is removed from the list of suffered task types. To prioritize the suffered task types in mapping events, we introduce *high-priority pairs* that include the feasible efficient pairs of suffered task types. To construct the high-priority pairs, we first generate the feasible efficient pairs as explained in Phase-I of ELARE. Next, each pair with a task not identified as suffered is removed from the list. The resultant list contains high-priority pairs.

The high-priority pairs are passed to Phase-II of FELARE, which is the same as Phase-II of ELARE, to make the mapping decisions.

4.8 Experimental Setup

To evaluate the performance of the proposed heuristics, we examine two scenarios: (i) using two deep learning applications (namely, face recognition [100] and speech recognition [101]) as the task types running on two AWS Virtual Machines (VMs) (t2.xlarge and g3s.xlarge)[b]) as the machine types; and (ii) simulating the HEC system with four machine types and four task types with synthesized expected execution time (EET) matrix.

In the first scenario, the `T2` instances of AWS are general-purpose machines that can be used for various workloads. They utilize Intel Xeon processors (Haswell E5-2676 v3 or Broadwell E5-2686 v4). The `t2.xlarge` instance has 4 vCPUs and 16 GB memory. The `G3` instances are best-matched for applications with intensive graphic and equipped with NVIDIA Tesla M60 GPUs. The Thermal Design Power (TDP) of Haswell E5-2676 v3 and NVIDIA Tesla M60 is 120 W and 300W,

[b]https://aws.amazon.com/ec2/instance-types/\#Intel

respectively.

To perform the face recognition, we first use MTCNN model [102] to detect the faces, then FaceNet model [100] is used to extract the embeddings. Eventually, Support Vector Machine (SVM) is used to classify the face embeddings. The input images consist of 30 images sampled from the LFW dataset [103]. Each AWS instance is used to process all input images. In addition, the experiment is repeated 30 times, and eventually, we collected the end-to-end latency for 900 inferences. For speech recognition, we used the DeepSpeech model [101]. The test dataset consists of 900 recorded audio which sums up to 118.9 hours of speech. Lastly, we use the collected execution times to construct the EET matrix.

To study the mapping heuristics behavior, we simulate an HEC system with four machine types and four task types. The four machines ($\{m_1, m_2, m_3, m_4\}$) have dynamic power consumption of $\{1.6 \cdot p, 3.0 \cdot p, 1.8 \cdot p, 1.5 \cdot p\}$ and idle powers of $0.05 \cdot p$ where p represents the unit power. The four heterogeneous task types are $\{T_1, T_2, T_3, T_4\}$. To model the heterogeneity of the HEC systems, we use the Coefficient-of-Variation-Based (CVB) technique[1] to populate the Expected Execution Time (EET) matrix. In the CVB method, the Coefficient of Variation (CV) of execution time values is used to measure the heterogeneity. Then, based on the task and machine CVs, two Gamma distributions are utilized to generate the expected execution times. The EET matrix is shown in Table 4.1. Next, the expected values in EET matrix are used to sample the execution time for each individual task from a Gamma distribution. For each task type, we calculate the

average of expected execution times on machine types, denoted $\bar{e}_i$. Then, we take the average of $\bar{e}_i$ for all task types, denoted $\bar{e}$, as the average of the collective expected execution time of all task types on all machine types. Finally, we determine the deadline of the task k of type i, $\delta_i(k)$, that arrives at arr_k by adding task type and collective mean values to the task's arrival time, as shown in Equation 4.4.

$$\delta_i(k) = arr_k + \bar{e}_i + \bar{e} \tag{4.4}$$

We assume the inter-arrival between tasks follows the Poisson distribution [104].

Table 4.1. Expected Execution Time (EET) matrix. Each entry (i, j) represents the expected execution time of task type i on machine type j. The CVB technique[1] was used to generate the EET matrix.

Tasks \ Machines	m_1	m_2	m_3	m_4
T_1	2.238	1.696	4.359	0.736
T_2	2.256	1.828	4.377	0.868
T_3	2.076	1.531	5.096	0.865
T_4	2.092	1.622	4.388	0.913

4.8.1 *Baseline Mapping Heuristics*

Two-phase mapping heuristics have been extensively studied in heterogeneous systems [12, 105]. Here, we focus on Minimum Completion Time-Minimum Completion Time (MM), Minimum Completion Time-Soonest Deadline (MSD), and Minimum Completion Time-Maximum Urgency (MMU) as baseline heuristics. The first phase of MM, MSD, and MMU are similar. The mapper selects a [task, machine] pair with a minimum expected completion time in

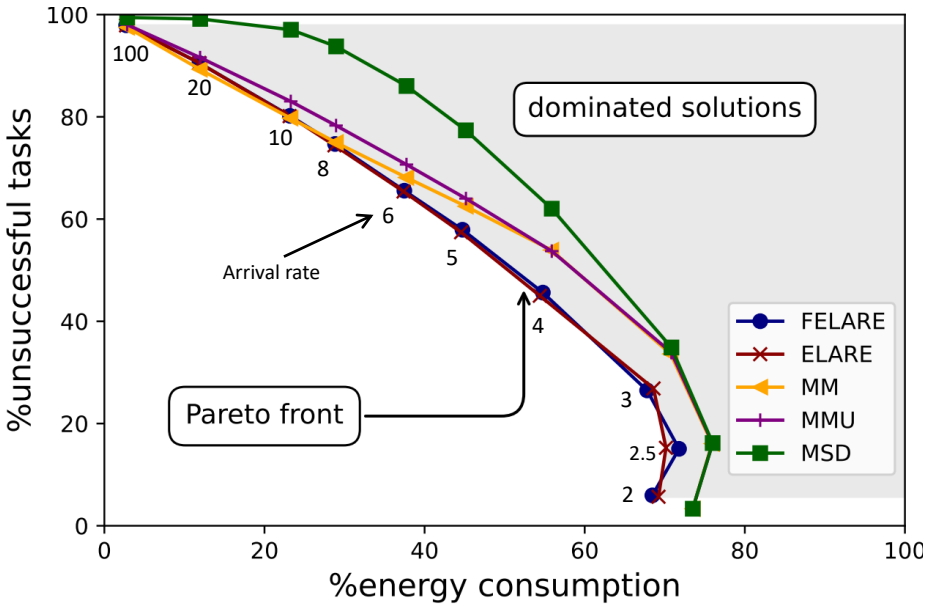
the first phase. Then, the list of [task, machine] pairs are used in the second phase to select an individual task for each of the available machines. The methods are distinguished based on the second phase of the algorithms. In MM, in the second phase, if there exist multiple tasks for a machine, the [task, machine] pair that offers minimum expected completion time is chosen, and then the task is allocated to the local queue of that machine. In MSD, it chooses the pair based on the soonest deadline. That is, for each available machine, it explores all the pairs generated in the first phase, and then it chooses the task with the earliest deadline. In case of the same deadline for multiple tasks, the task with the minimum expected completion time is selected and assigned to the queue of the machine. In MMU, it uses the urgency metric for selecting a task. The urgency of task k of type i is defined as $1/(\delta_i(k) - e_{ij})$. So, the task with maximum urgency is selected and assigned to the local queue of the machine.

4.9 Performance Evaluation

4.9.1 *Energy and Latency Trade-off*

Recall that energy- and latency make the resource allocation problem of HEC systems a bi-objective optimization problem. Minimizing the energy consumption conflicts with maximizing the completion rate and/or minimizing the deadline miss rate of tasks. As a result, there is not a single optimal solution, instead, there could be a set of solutions that dominate other solutions. Figure 4.3 shows the energy consumption and deadline miss rate for the mapping heuristics at different arrival rates. Moving from right to left on each curve increases the arrival

Figure 4.3. The trade-off between energy consumption and latency across ELARE, FELARE, and other baseline heuristics. Each curve belongs to one mapping heuristic at different arrival rates. The gray zone describes the solutions that were dominated by the Pareto-front curve. Both ELARE and FELARE are non-dominated solutions that form the Pareto-front.



rate. We can observe that at an extremely high arrival rate (*e.g.,* 100 tasks per second), all methods exhibit similar performance (high miss rate with low energy consumption). In fact, at a very high arrival rate, a resource-limited system is highly oversubscribed and tasks are missed regardless of the applied mapping heuristic. However, the proposed heuristics, ELARE and FELARE, dominate other heuristics at lower arrival rates. In other words, we can say that ELARE and FELARE at low to moderate arrival rates belong to the set of non-dominated solutions or the Pareto-front. This analysis recommends us to employ ELARE and FELARE, particularly, at low to moderate arrival rates.

4.9.2 *Analyzing the Wasted Energy*

This experiment is to examine the performance of ELARE and FELARE on

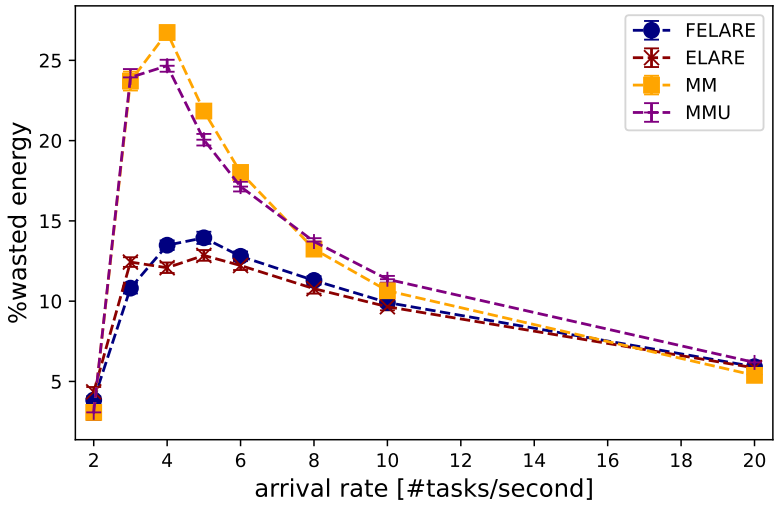
minimizing the energy wasted due to processing infeasible tasks. To this end, we used 30 synthesized workload traces with different arrival rates where each workload trace included 2,000 tasks. We also measure the wasted energy as the percentage of energy consumed by machines to process the missed tasks with respect to the initial available energy of the HEC system.

Figure 4.4 shows the results of the wasted energy analysis. We observe that the wasted energy for ELARE and FELARE at low to moderate arrival rates is much less than the other heuristics. Specifically, deploying ELARE shows 12.6% less wasted energy at an arrival rate of 4 tasks per second than the MM heuristic. However, for the high arrival rates, all the heuristics converge to a low energy wastage. The reason is that, at the high arrival rates, most of the tasks become infeasible and there is no chance to make them feasible. Hence, they miss their deadline before even being assigned, regardless of the mapping heuristic being used. A similar trend is observed in Figure 4.5 where the deep learning applications are executed on the AWS instances.

These observations can be explained by considering how the energy is consumed in each one of the heuristics. We explore this granular behavior of heuristics in the next paragraph. However, the observed results confirm that ELARE and FELARE methods definitely waste less energy than others for the low to medium arrival rates.

Figure 4.6 shows the percentage of unsuccessful tasks, due to missing deadline or cancellation (dropping) before the assignment, for both MM and

Figure 4.4. The wasted energy due to deadline miss at different arrival rates for different mapping heuristics



ELARE at different arrival rates. We observe that ELARE outperforms MM for the lower arrival rates. Specifically, ELARE reduces the unsuccessful tasks by 8.9% for the arrival rate of 3. We see that ELARE proactively cancels most of the unsuccessful tasks, whereas, the majority of the unsuccessful tasks for MM are due to missing deadlines, which implies energy wastage. That is why the wastage for MM is remarkably higher than ELARE in low to moderate arrival rates. However, the canceled-to-missed ratio for MM gradually increases, because the system becomes oversubscribed and the arriving tasks cannot be allocated and they are eventually dropped from the arriving queue.

4.9.3 *Analyzing Fairness Across Task Types*

Although ELARE considers both the energy consumption and deadline constraints, it is biased towards certain task types. To address this problem, we proposed the FELARE heuristic to improve fairness across task types. In this part, we conduct an experiment to compare the fairness of FELARE against other

Figure 4.5. The wasted energy due to deadline miss of face recognition and speech recognition applications on AWS instances at different arrival rates for MM and EE

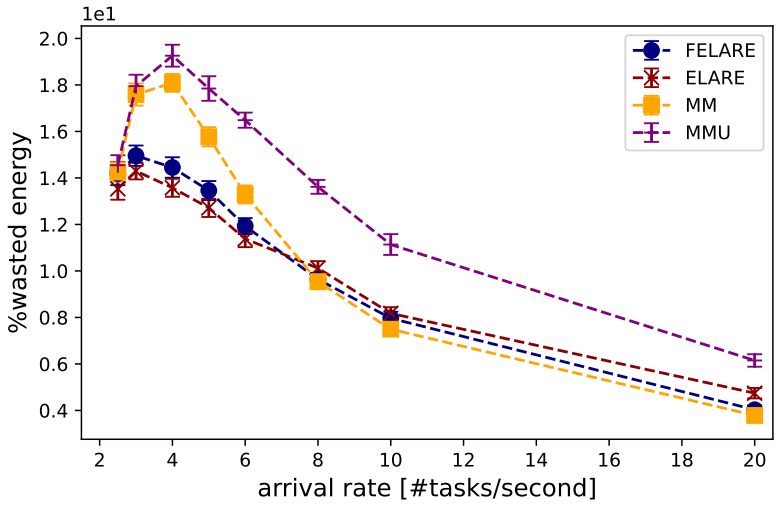


Figure 4.6. The percentage of unsuccessful tasks at different arrival rates for MM and ELARE. Unsuccessful tasks are those that are either cancelled (not assigned to the machines) or dropped due to missing deadlines.

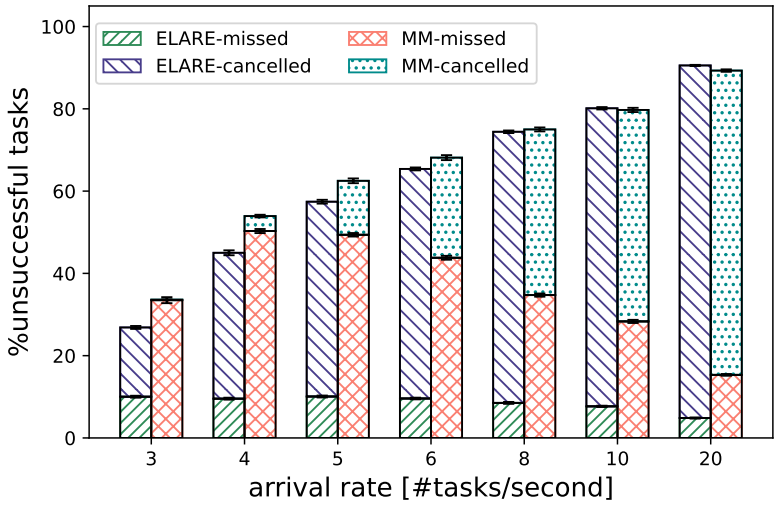
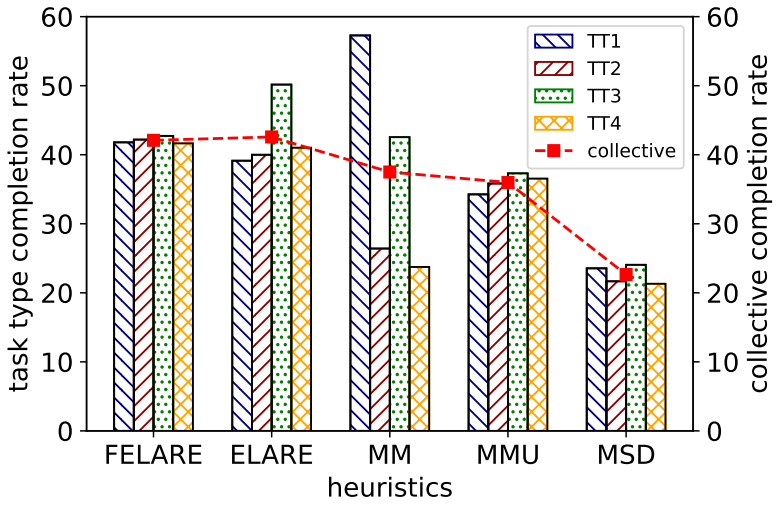


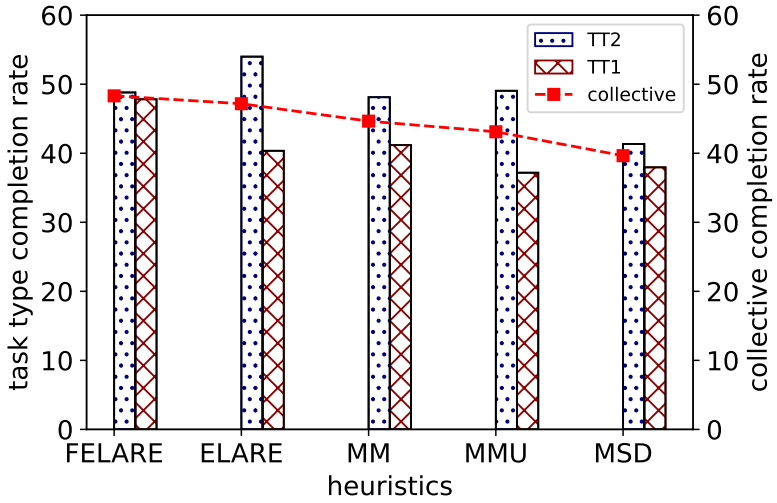
Figure 4.7. The fairness across task types TT_1 to TT_4 for FELARE, ELARE, MM, MMU, and MSD heuristics. Also, the right-side vertical axis and red data points represent the collective completion rate resulting from each heuristic.



heuristics. To this end, we utilize 30 synthesized workload traces with an arrival rate of 5.0 tasks per second where each workload contains 2000 tasks. The task types and machines are characterized as described in Section 4.8

Figure 4.7 shows the results of this experiment. The x-axis represents the heuristics. The left and right y-axes also represent the task type and collective completion rates, respectively. Here, the collective completion rate represents the ratio of the successfully completed tasks to the total number of tasks that have arrived in the system. We observe that ELARE is biased towards T_3 and MM towards T_1 and T_3. However, FELARE could considerably improve the fairness with negligible degradation in the total completion rate. In the case of AWS workload trace, Figure 4.8 shows the fairness of mapping heuristics across face and speech recognition applications at an arrival rate of 2 tasks per second. The results are in agreement with the previous experiment on the synthesized workload, where the

Figure 4.8. The figure shows the fairness across task types, face recognition, and speech recognition on AWS instances for FELARE, ELARE, MM, MMU, and MSD. Also, the right y-axis and red curve represent the collective completion rate for each heuristic.



FELARE method exhibits substantially higher fairness than the other heuristics.

4.10 Summary

In this chapter, we investigated the fair and energy-efficient allocation of concurrent and latency-sensitive ML applications in Heterogeneous Edge Computing (HEC) systems. We proposed a two-phase heuristics, called ELARE, to address both energy and latency objectives. This heuristic proactively drops tasks that are unlikely to meet their deadlines, thereby, avoiding wasting energy. In the experiments, we showed that ELARE could considerably reduce the wasted energy via proactively dropping infeasible tasks and choosing the machines with the minimum energy usage for each task. To address the bias to certain task types in ELARE, we extended it and proposed the FELARE heuristics that treats all task types fairly. FELARE measures the completion rate per task type to determine the suffered task types. Then, it prioritizes the suffered task types in each mapping

event. Moreover, in the case of observing infeasible suffered task types in a mapping event, FELARE drops tasks from the non-suffered task types to make the infeasible suffered tasks, feasible. The evaluation results showed that devising customized mapping heuristics for the HEC systems can noticeably improve fairness and energy consumption.

Chapter 5: Performance-driven Quantification of Heterogeneity in Distributed Computing Systems

5.1 Introduction

We define a heterogeneous computing system as a set of architecturally diverse machines that work together to complete a set of requests (a.k.a. *tasks*) with different computational requirements. We categorize the tasks arriving to a system based on the type of operation they perform and call them *task types*. For instance, in a system that assists blind and visually impaired people [17, 18], the task types can be obstacle detection, face recognition, and speech recognition. Moreover, we classify machines of a computing system based on their architectural and performance characteristics and call each one a *machine type*. For instance, a cloud solution architect can form a virtual compute cluster using ARM, x86-based, and GPU machine types. In this work, we consider the system heterogeneity that emanates from the diversity in machine types and computational requirements of task types. In other words, system heterogeneity has two dimensions: (i) *machine heterogeneity* and (ii) *task heterogeneity*. Profiling various task types on heterogeneous machine types can describe their execution time behavior. Variations in the performance (a.k.a. execution time) of a given task type across all the machine types are defined as machine heterogeneity, whereas, variations in the execution time of different task types on a given machine type are defined as task heterogeneity. Then, the *system heterogeneity* is defined as the compounded heterogeneity of these two dimensions.

To quantify the heterogeneity of such computing systems in a manner that can describe the system performance behavior in terms of makespan and throughput, we need to address the following research questions: (i) *How to find a measure that can quantify the heterogeneity of any given computing system?* (ii) *How to exploit the heterogeneity measure to predict the performance behavior of a computing system?*

We define *Expected Execution Time (EET)* as a matrix to store the expected execution time of each task type on each machine type. In this manner, entry $EET[i, j]$ (denoted e_{ij}) represents the expected execution time of task type i on machine type j. It is needless to say that, for systems with multiple instances of the same machine type, the corresponding columns of those machines in the EET matrix are identical. The EET matrix, as a whole, represents the expected performance of the entire system in terms of the tasks' execution times. As such, the matrix can be leveraged as a guide to understand the throughput that the system can potentially achieve. We propose a mathematical model to measure the task heterogeneity and machine heterogeneity levels separately as two dimensions describing the system heterogeneity level. Then, we devise a mathematical model to determine the overall system heterogeneity based on these dimensions. For that purpose, we examine various measures of central tendency, namely arithmetic, geometric, and harmonic means, and propose a single measure, called Homogeneous Equivalent Execution Time ($HEET$), that, for a given EET matrix and workload, describes the expected execution time of a hypothetical homogeneous system whose

throughput is similar to the heterogeneous system represented by EET matrix. Subsequently, we employ HEET measure to determine the throughput of the heterogeneous computing system. We evaluate the HEET measure for various heterogeneous systems and workloads and show that it can precisely describe the impact of heterogeneity level on the desired throughput. In summary, the specific contributions of my research work in this chapter are as follows:

- Providing a measure to quantify the system heterogeneity such that it can be used to determine the throughput of the system for a given workload.
- Proposing a systematic approach to analyzing heterogeneity of a computing system by means of decoupling the heterogeneity into machine and task heterogeneity dimensions, and characterizing each dimension separately by making use of a custom-defined speedup metric.
- Proving the appropriateness of arithmetic mean and harmonic mean to measure the central tendency of speedup values due to machine heterogeneity and task heterogeneity, respectively.
- Validating how the system performance (makespan and throughput) can be derived as a function of the proposed heterogeneity measure.

The rest of this chapter is organized as follows: In Section 5.2, we describe the mathematical model to quantify the system heterogeneity based on the overall speedup a system obtains by employing the heterogeneity. Section 5.3 provides an overview of the real-world implementation we have developed to validate the

accuracy of HEET measure. Section 5.4, examine the proposed heterogeneity measure. The chapter summary is presented in Section 5.5.

5.2 Analyzing Performance Characteristics of Heterogeneous Systems

System heterogeneity is the result of the synergy between machine heterogeneity and task heterogeneity dimensions. Accordingly, our approach for quantifying system heterogeneity is to measure the heterogeneity of each dimension individually and then fuse them to quantify the overall system heterogeneity. To this end, we base our analysis on the notion of EET matrix that is representative of the system performance. Considering that each row of EET represents a task type and each column represents a machine type, the row-wise variations illustrate the machine heterogeneity, and similarly, column-wise variations express the task heterogeneity. Note that, in the event that some machines in the system are homogeneous, their corresponding columns in the EET matrix are repeated.

We exploit the notion of speedup to characterize the impact of heterogeneity on the system execution time behavior. The gained "speedup due to machine heterogeneity" can be described by the row-wise analysis of the EET matrix. We represent this speedup by a row vector, denoted $\vec{\alpha}^{(i)}$, that has the same dimension as i$^{\text{th}}$ row of EET matrix. Each entry $\alpha_j^{(i)}$ denotes the speedup that the system can achieve when machine type j executes task type i as opposed to executing it on the slowest machine type. Accordingly, given the EET matrix of size $m \times n$, the value of

$\alpha_j^{(i)}$ is determined based on Equation 5.1.

$$\alpha_j^{(i)} = \frac{\max_{j=1}^{n} e_{ij}}{e_{ij}} \tag{5.1}$$

In the above equation, $\alpha_j^{(i)} = 1$, if and only if machine type j is the same as the slowest machine for task type i.

Similarly, we define "speedup due to task heterogeneity", denoted $\vec{\beta}_{(j)}$, as a column vector with the same dimension of j$^{\text{th}}$ column of the EET matrix. Entry $\beta_{(j)}^i$ indicates the speedup achieved by executing a task of type i on machine type j, as opposed to executing the slowest task type on that machine. Formally, $\beta_{(j)}^i$ is calculated based on Equation 5.2.

$$\beta_{(j)}^i = \frac{\max_{i=1}^{m} e_{ij}}{e_{ij}} \tag{5.2}$$

The value of $\beta_{(j)}^i$ is one, if and only if task type i is the slowest task type on machine type j.

Given the speedup vectors due to machine heterogeneity and task heterogeneity, we need to represent each speedup vector in the form of a scalar value. This scalar value for $\vec{\alpha}^i$ ($\vec{\beta}_{(j)}$) accurately describes how much all machine types (task types) together can gain speedup because of heterogeneity in the machines (tasks). Based on the mean-field method [106, 107], the interaction of variables in a complex stochastic system can be replaced by the average interactions between those variables. As such, we can use mean to represent the speedup behavior of all $(task, machine)$ pairs in both $\vec{\alpha}^{(i)}$ and $\vec{\beta}_{(j)}$. For that purpose, we

employ statistical measures of central tendency (arithmetic, geometric, and harmonic mean) to accurately represent $\vec{\alpha}^{(i)}$ and $\vec{\beta}_{(j)}$. However, the challenge is that there is no consensus on the appropriateness of these measures to capture the central tendency of a specific use case [108, 109] and it has to be investigated on a case-by-case basis. An appropriate mean speedup value derived from the EET matrix must precisely depict the "real speedup", a.k.a. *true speedup* (denoted Γ), that the heterogeneous system can achieve for a given workload. We exploit the notion of makespan (*i.e.,* the total time of executing workload) to calculate the true speedup. According to Equation 5.3, the true speedup is determined based on the makespan of executing the workload on the heterogeneous system with respect to its slowest "counterpart homogeneous system", as the base system (a.k.a. baseline).

$$\Gamma = \frac{\text{homogeneous system makespan}}{\text{heterogeneous system makespan}} \tag{5.3}$$

Note that the counterpart homogeneous system is represented by an EET matrix whose entries are all equal to the maximum value of the EET matrix of the heterogeneous system. We use Γ_M and Γ_T to represent true speedup with respect to the machine heterogeneity and task heterogeneity, respectively.

Once we know the true speedup for a given workload, we can compare it against the calculated mean speedup to know how accurate the calculated one is. We provide three lemmas to introduce appropriate central tendency measures that can represent speedup due to machine heterogeneity and task heterogeneity.

In the first step, to quantify the system heterogeneity, for each machine type,

we utilize the EET matrix to generate the speedup vectors due to task heterogeneity using Equation 5.2. These speedup column vectors construct a speedup matrix due to task heterogeneity, denoted by S^T. In the second step, we represent each speedup vector due to task heterogeneity by its mean value. Note that, in this step, we fuse all task types into a single hypothetical equivalent task type that can describe the execution time behavior of all task types. Taking into account that we consider the execution time of the slowest task type as the baseline, in the third step, for each machine type, we utilize the mean speedup values due to task heterogeneity and the baseline execution time to determine the execution time behavior of the hypothetical equivalent task type. We repeat this step for all machine types to construct a row vector depicting the execution times of the hypothetical equivalent task type on the machine types. In step four, based on the row vector generated in the last step (as the EET of the representative task type), we generate the speedup vector due to machine heterogeneity based on Equation 5.1. Eventually, we reduce the speedup vector due to machine heterogeneity for the hypothetical equivalent task type into a single scalar value. We use this mean speedup value and the execution time of the slowest machine for the hypothetical task type to determine a single-value execution time that represents the execution time behavior of the entire system. We use this representative execution time of the hypothetical equivalent task type on the hypothetical equivalent machine type to define a heterogeneity measure.

Provided that heterogeneity affects the system performance by means of the execution time (reflected in the EET matrix), we define a measure based on the

execution time behavior, called *Homogeneous Equivalent Execution Time (HEET)*, to estimate the impact of the system heterogeneity on throughput. In fact, HEET represents how fast the heterogeneous machines of the system are to process heterogeneous tasks. Our hypothesis is that systems with the same HEET measure expose comparable execution time behavior, thus, for a given workload, they exhibit similar makespan and throughput. Therefore, the HEET score is able to properly characterize the system heterogeneity with respect to the throughput it can offer without running the workload against the heterogeneous system. In the rest of this section, we elaborate on characterizing the machine heterogeneity and task heterogeneity dimensions. Then, we discuss how to fuse these dimensions to characterize the system heterogeneity.

5.2.1 Characterizing Machine Heterogeneity

For task type i, we process row i of the EET matrix with respect to the slowest machine for that task type to form a *row speedup vector*, denoted $\vec{\alpha}^{((i)}$, representing the speedup values due to machine heterogeneity. For task type i, we use the central tendency measure (mean) of the row speedup vector components, denoted $\bar{\alpha}^{(i)}$, to aggregate the speedup behavior of all machine types.

According to [110], in the circumstances where the performance is expressed as a rate (*e.g.,* flops), generally, harmonic mean can accurately express the central tendency. Also, the central tendency can be usually represented by the arithmetic mean when the performance is of a time nature (*e.g.,* makespan or total execution time of a benchmark). Lastly, they suggest avoiding using geometric mean when the

performance is of the time or rate nature. In another study [109], the speedup is considered as the performance metric to compare an enhanced system against a baseline one. To that end, they used a benchmark suite to evaluate the performance of each system. Then, based on the makespan of each individual benchmark in the benchmark suite, the speedup for the enhanced system is calculated. Next, the authors discussed different measures of central tendency (*i.e.,* arithmetic, harmonic, and geometric mean) to summarize the speedup results of benchmarks into a single number such that it appropriately describes the overall speedup for the entire benchmark suite. To validate the appropriateness of the mean speedup measure, they compared it against the speedup achieved by considering the makespan of the entire benchmark suite on both systems.

In this research, we also follow the same approach as [109] to validate the accuracy of the central tendency measure of the speedup matrix. Particularly, we use the notion of true speedup (Γ) to verify that the central tendency measure accurately represents the row speedup vector. In addition, the intensity of task arrival to the system impacts machines' utilization that, in turn, affects the true speedup. In Lemma 5.1, we study an extreme case when the task arrival rate is large enough such that all machines in the system have a task for execution at all times. We show that for such a system, arithmetic mean can appropriately summarize $\vec{\alpha}^{((i)}$ and represent the mean speedup due to machine heterogeneity. For the other side of the spectrum, where the task arrival rate is low such that only one machine in the system is executing a task at a time, we present Lemma 5.3 to prove

that the harmonic mean should be used to accurately summarize $\vec{\alpha}^{((i)}$ and represent the mean speedup due to machine heterogeneity. In these lemmas, we assume that there is a single unbounded FCFS queue of tasks that are all available for execution (like bag-of-tasks [38]). Whenever a machine becomes free, it takes the next task from the queue to execute it.

Lemma 5.1. *Let $EET = [e_{ij}]$ $(1 \leqslant j \leqslant n)$ denote EET vector of a heterogeneous computing system consisting of a set of machine types, $M = \{M_1, M_2, ..., M_n\}$, and a workload with $c > n$ tasks of type T_i that are all available for execution (like bag-of-tasks [38]). Tasks are queued upon arrival into a single unbounded FCFS queue. Whenever a machine becomes free, it takes a task from the queue and executes it. Then, the true speedup is calculated as follows:*

$$\Gamma_M = \bar{\alpha}^{(i)(A)} \tag{5.4}$$

where

$$\bar{\alpha}^{(i)(A)} = \frac{1}{n} \cdot \sum_{j=1}^{n} \alpha_j^{(i)} \tag{5.5}$$

Proof. We assume that machine type k is the slowest for task type T_i, that is, we have $\max_{j=1}^{n} e_{ij} = e_{ik}$. Then, the baseline system consists of n machines with the expected execution time of e_{ik}. For a single FCFS queue, c tasks are equally distributed across n homogeneous machines. Hence, each machine has to handle $\frac{c}{n}$ tasks where the expected execution time of each task is e_{ik}. This means that the

total time to complete those c tasks on the homogeneous system is $\lceil \frac{c}{n} \rceil \times e_{ik}$. We know that $0 \leqslant \lceil \frac{c}{n} \rceil - \frac{c}{n} < 1$. If we replace $\lceil \frac{c}{n} \rceil$ with $\frac{c}{n}$, the error in calculating total time is at most $(\lceil \frac{c}{n} \rceil - \frac{c}{n})/\lceil \frac{c}{n} \rceil$, which is negligible for large number of tasks ($c \gg n$). Thus, for the sake of simplicity, we assume that the makespan of completing c tasks on the counterpart homogeneous system is $\frac{c}{n} \times e_{ik}$.

In the heterogeneous system, however, the proportions of the tasks handled by different machine type are not equal, because faster machines can execute more tasks. Specifically, machine type j that has $e_{ij} \leqslant e_{ik}$ executes e_{ik}/e_{ij} tasks while machine type k executes only one task. As a result, the proportion of the total number of tasks executed by the slowest machine type to the total number of tasks, denoted by p_k, is calculated as follows:

$$p_k = \left\lceil \frac{c}{\sum_{j=1}^{n} \frac{e_{ik}}{e_{ij}}} \right\rceil, \quad c \geqslant n \tag{5.6}$$

Then, the makespan of executing c tasks on the heterogeneous system is $p_k \times e_{ik}$. In Equation 5.6, we know that $e_{ik}/e_{ij} = \alpha_j^{(i)}$, therefore, the speedup of the heterogeneous system is calculated as follows:

$$\Gamma_M = \frac{1}{n} \cdot \frac{c}{\left\lceil \frac{c}{\sum_{j=1}^{n} \alpha_j^{(i)}} \right\rceil} \tag{5.7}$$

Assuming that $\frac{c}{\sum_{j=1}^{n} \alpha_j^{(i)}} \in \mathbb{Z}$, we have $\left\lceil \frac{c}{\sum_{j=1}^{n} \frac{e_{ik}}{e_{ij}}} \right\rceil = \frac{c}{\sum_{j=1}^{n} \frac{e_{ik}}{e_{ij}}}$ and Equation 5.7 can be represented as follows:

$$\Gamma_M = \frac{1}{n} \cdot \frac{c}{\frac{c}{\sum_{j=1}^{n} \alpha_j^{(i)}}} = \frac{1}{n} \cdot \sum_{j=1}^{n} \alpha_j^{(i)} = \bar{\alpha}^{(i)(A)} \tag{5.8}$$

Note that $\bar{\alpha}^{(i)(A)}$ is the arithmetic mean of $\vec{\alpha}^{(i)}$ components. However, if $\frac{c}{\sum_{j=1}^{n} \alpha_j^{(i)}} \notin \mathbb{Z}$, the true speedup is not exactly the arithmetic mean of the row speedup vector components ($\bar{\alpha}^{(i)(A)} \neq \Gamma_M$), and the corresponding error, denoted $\epsilon^{(i)}$, is determined as follows:

$$\epsilon^{(i)} = \frac{\bar{\alpha}^{(i)(A)} - \Gamma_M}{\Gamma_M} < \frac{\sum_{j=1}^{n} \alpha_j^{(i)}}{c} \tag{5.9}$$

It is also noteworthy that, for a large number of tasks ($c >> \sum_{j=1}^{n} \alpha_j^{(i)}$), we have $\epsilon^{(i)} \approx 0$

∎

In Lemma 5.1, we made the assumption that tasks are queued into a single unbounded FCFS queue. Whenever a machine becomes available, it selects a task from the queue and processes it. In support of this scheduling approach, we introduce Lemma 5.2, demonstrating that it yields the minimum makespan for bag-of-tasks on heterogeneous computing systems.

Lemma 5.2. *Let $EET = [e_{ij}]$ $(1 \leqslant j \leqslant n)$ denote EET vector of a heterogeneous computing system consisting of a set of machine types, $M = \{M_1, M_2, ..., M_n\}$, and a workload with c tasks of type T_i that are all available for execution (i.e., bag-of-tasks). Then, the minimum makespan (total time to complete the workload) is*

obtained by using a Round-Robin load balancer across available machines. That is, whenever a machine becomes free, it takes a task from the queue and executes it.

Proof. Based on the Round-Robin load balancer for available machines, whenever a machine becomes free, it takes a task from the queue and executes it. Let $e_k = \max_{j=1}^{n} e_{ij}$, representing the slowest machine type for task T_i. Based on the proof in Lemma 5.1, the number of tasks completed on each machine is as follows:

$$n_j = \frac{e_k}{e_{ij}} \cdot \frac{c}{\sum_{j=1}^{n} \frac{e_k}{e_{ij}}} = \frac{c}{e_{ij} \sum_{j=1}^{n} \frac{1}{e_{ij}}} \tag{5.10}$$

These fractions have the following characteristics:

$$\sum_{j=1}^{n} n_j = \sum_{j=1}^{n} \frac{c}{e_{ij} \sum_{j=1}^{n} \frac{1}{e_{ij}}} = c \tag{5.11}$$

Also, based on Lemma 5.1, the makespan, denoted τ^*, is determined as follows:

$$\forall j \quad n_j \cdot e_{ij} = \frac{c}{\sum_{j=1}^{n} \frac{e_k}{e_{ij}}} \cdot e_k = \tau^* \tag{5.12}$$

We assume that there are other fractions of tasks completed on machines, denoted n'_j $(1 \leqslant j \leqslant n)$, such that the resultant makespan, denoted τ', is less than τ^*. Thus, for each machine, n'_j must be less than n_j. If there exists a machine with $n'_j > n_j$, then the corresponding makespan of the tasks completed on that machine will be $\tau' = n'_j \cdot e_{ij} > n_j \cdot e_{ij} = \tau^*$, which is in contradiction with primary assumption, $\tau' < \tau^*$. Thus, we have:

$$n'_j < n_j \quad 1 \leqslant j \leqslant n \tag{5.13}$$

Based on Equation 5.13, for the total number of tasks completed on machines, we have:

$$\sum_{j=1}^{n} n'_j < \sum_{j=1}^{n} n_j \tag{5.14}$$

Based on Equation 5.11, $\sum_{j=1}^{n} n_j = c$. As a result,

$$\sum_{j=1}^{n} n'_j < c \tag{5.15}$$

Thus, we cannot obtain a makespan less than τ^* with the same number of tasks. This proves that assigning tasks that are all available for execution on available machines in a Round-Robin manner results in a minimum makespan.

∎

In Lemma 5.1, we studied the case where the tasks are available for execution a priori so that the impact of task arrival is abstracted. On the other side of the spectrum, we can consider a system with sparse task arrival (*i.e.,* large inter-arrival times between tasks) where some machines may become idle while others are busy. In an extreme case with a very low arrival rate, the entire system can potentially become idle. Under certain (threshold) arrival rates, in between the two extremes, only one machine in the system is busy and the rest are idle during the workload

processing. Any arrival rate lower than the threshold leads to system idling, whereas, any higher arrival rates lead to more than one busy machine at a time.

Lemma 5.3. *Let $EET = [e_{ij}]$ $(1 \leqslant j \leqslant n)$ denote EET vector of a heterogeneous computing system consisting of a set of machine types, $M = \{M_1, M_2, ..., M_n\}$, and a workload with $c > n$ tasks of type T_i. The arrival rate is such that the number of busy machines is exactly one throughout the processing of the workload. Tasks are queued upon arrival into a single unbounded FCFS queue. Whenever a machine becomes free, it takes a task from the queue and executes it. In this case, the mean speedup of using a heterogeneous system for task type T_i is the "harmonic mean" of the $\vec{\alpha}$ components, denoted by $\bar{\alpha}^{(i)(H)}$, and it is calculated as follows:*

$$\Gamma_M = \bar{\alpha}^{(i)(H)} = \frac{n}{\sum_{j=1}^{m} \frac{1}{\alpha_j^{(i)}}} \tag{5.16}$$

Proof. Let k be the slowest machine type in the heterogeneous system that executes task type T_i with the expected execution time of e_{ik}. We consider a baseline homogeneous system counterpart with n machines of type k and with the FCFS scheduler. Recall that, we assume the task arrival is such that only one machine is busy at any given time. Hence, the makespan is the sum of execution times for all tasks, and for c tasks the makespan is $c \times e_{ik}$ and each machine executes $\frac{c}{n}$ tasks. Also, makespan for the heterogeneous system is $\frac{c}{n} \cdot \sum_{j=1}^{m} e_{ij}$. Considering these, the speedup of using a heterogeneous computing system is calculated as follows:

$$\Gamma_M = \frac{c \cdot e_{i,k}}{\frac{c}{n} \cdot \sum_{j=1}^{m} e_{ij}} = \frac{n}{\sum_{j=1}^{m} \frac{1}{\alpha_j^{(i)}}} = \bar{\alpha}^{(i)(H)} \tag{5.17}$$

As we can see, the speedup of using a machine heterogeneous system that has only one machine busy at any given time aligns with the harmonic mean formula. ∎

5.2.2 Characterizing Task Heterogeneity

In Section 5.2.1, we considered the row speedup vector to characterize machine heterogeneity for a given task type. Likewise, to characterize task heterogeneity, for machine type j, we define *column speedup vector*, denoted $\vec{\beta}_{(j)}$. Then, we summarize the column speedup vector into a representative mean value, denoted $\bar{\beta}^{H}_{(j)}$.

In Lemmas 5.1 and 5.3, based on the true speedup, we proved that arithmetic mean (for high task arrival rate) and harmonic mean (for low task arrival rate) are representative measures for central tendency of the row speedup vector due to machine heterogeneity. Next, in Lemma 5.4, we shift our attention to the dimension of task heterogeneity and prove that harmonic mean is an accurate measure for central tendency of $\vec{\beta}_{(j)}$.

Lemma 5.4. *Let $EET = [e_{ij}]$ $(1 \leqslant i \leqslant m)$ denote the expected execution time (EET) vector of a set of task types, $T = \{T_1, T_2, ..., T_m\}$ on machine type M_j. A workload trace of c tasks $(c > m)$ of type $T_i \in T$ arrive to the system. Tasks are queued upon arrival into a single unbounded FCFS queue and executed by machine*

M_j. Then, the true speedup is determined as follows:

$$\Gamma_T = \frac{1}{\sum_{i=1}^{m} \frac{\omega_i}{\beta^i_{(j)}}} = \bar{\beta}^H_{(j)} \tag{5.18}$$

Proof. Assume that k^{th} task type is the largest task type. That is, e_{kj} is the maximum value in the j^{th} column of EET. Then, for a homogeneous workload that contains only task type T_k, the total time consumed by machine M_j to execute those tasks is $c \times e_{kj}$. However, for a heterogeneous workload with ω_i as the proportion of each task type to the total number of tasks, the total time required to execute each task type by machine M_j is $c \times \omega_i \times e_{ij}$. Thus, the total time consumed to process all tasks is $\sum_{i=1}^{m} c \cdot \omega_i \cdot e_{ij}$. Then, the speedup of executing the heterogeneous workload, as opposed to the homogeneous workload of type T_k, is determined based on Equation 5.19.

$$\Gamma_T = \frac{c \cdot e_{kj}}{\sum_{i=1}^{m} c \cdot \omega_i \cdot e_{ij}} = \frac{1}{\sum_{i=1}^{m} \omega_i \cdot \frac{e_{ij}}{e_{kj}}} \tag{5.19}$$

Based on Equation 5.2, we know that $\beta^i_{(j)} = \frac{e_{kj}}{e_{ij}}$. In addition, the weighted harmonic mean of the column speedup vector due to task heterogeneity, denoted by $\bar{\beta}^H_{(j)}$, is calculated based on Equation 5.20.

$$\bar{\beta}^H_{(j)} = \frac{1}{\sum_{i=1}^{m} w_i \frac{1}{\beta^i_{(j)}}} \tag{5.20}$$

Finally, based on Equations 5.19 and 5.20, we prove that true speedup due to

task heterogeneity is the weighted harmonic mean of $\vec{\beta_j}$ as noted in Equation 5.21.

$$\Gamma_T = \frac{1}{\sum_{i=1}^{m} \omega_i \cdot \frac{1}{\beta^i_{(j)}}} = \bar{\beta}^H_{(j)} \tag{5.21}$$

■

5.2.3 Homogeneous Equivalent Execution Time (HEET) Measure

From Lemmas 5.1 and 5.3, we learn that, for a high arrival rate, arithmetic mean represents the mean speedup ($\bar{\alpha}^{(i)(A)}$), whereas, for a low arrival rate, harmonic mean ($\bar{\alpha}^{(i)(H)}$) is representative. However, we are typically interested in studying system efficiency under high arrival rates. That is why, in the rest of this paper, we use arithmetic mean to represent speedup due to machine heterogeneity. Moreover, Lemma 5.4, proves that the weighted harmonic mean represents the mean speedup due to task heterogeneity. Taking these into account, as shown in the example of Figure 5.1, we can reduce each column vector $\vec{\beta}_{(j)}$ to its mean value $\bar{\beta}^H_{(j)}$ to construct a row vector, denoted $\bar{\beta}^H = [\bar{\beta}^H_{(1)}, \bar{\beta}^H_{(2)}, ..., \bar{\beta}^H_{(n)}]$, whose contents summarize the execution time behavior of all task types into an equivalent hypothetical task type (denoted T^*). We use Equation 5.21 to determine $\bar{\beta}^H_{(j)}$ for machine type j. Note that, the mean speedup due to task heterogeneity for machine type j is calculated with respect to the slowest task type for machine type j (homogeneous counterpart). Thus, to determine the expected execution time of T^* on machine type j, we consider that T^* is $\bar{\beta}^H_{(j)}$ times faster than the slowest task type on machine type j. The resultant row vector of the expected execution time of T^* on machine types describes the machine heterogeneity. In a similar approach, we

can aggregate machine types into a single hypothetical machine type (denoted M^*) whose execution time behavior is representative of the entire set of machine types in the heterogeneous system. To that end, we construct the speedup vector due to machine heterogeneity for T^*, denoted by $\vec{\alpha}^{(*)}$, based on Equation 5.22.

$$\alpha_j^{(*)} = \frac{\max_{i=1}^{m} e_{ij}}{\bar{\beta}_{(j)}^{H}} \tag{5.22}$$

Then, we use the arithmetic mean of $\vec{\alpha}^{(*)}$ to determine the mean speedup due to machine heterogeneity, denoted by $\bar{\alpha}^{(*)(A)}$ for T^*. In fact, $\bar{\alpha}^{(*)(A)}$ represents how much M^* is faster than the slowest machine type for the hypothetical equivalent task type (T^*). As a result, we can represent the execution time behavior of the heterogeneous computing system with Homogeneous Equivalent Execution Time, HEET, using the expected execution time of T^* on M^* as follows:

$$HEET = \frac{\max_{j=1}^{n} \alpha_j^{(*)}}{\bar{\alpha}^{(*)(A)}} \tag{5.23}$$

For the sake of clarification, we use Figure 5.1 to illustrate the derivation of HEET by means of an example. In Stage (a) of the figure, an EET matrix is considered. Then, in Stage (b), the EET matrix is used to derive the speedup matrix due to task heterogeneity based on Equation 5.2. Column j of this matrix demonstrates the speedup due to task heterogeneity for machine type j, denoted by $\vec{\beta}_{(j)}$. Next, in Stage (c), we employ harmonic mean (Equation 5.20) to represent each column vector $\vec{\beta}_{(j)}$ in the form of a scalar value. In this way, the set of speedup vectors due to task heterogeneity (that constructs a matrix) are reduced to a single

row speedup vector. In Stage (d), we calculate the expected execution time of T^* on machines by using the execution time of the slowest task type and $\overline{\beta}^H_{(j)}$. In Stage (e) we determine the speedup vector due to machine heterogeneity for T^*, based on Equations 5.1. In stage (f), we use the arithmetic mean to determine the mean speedup due to machine heterogeneity for T^*. Lastly, in stage (g), we calculate the HEET measure by considering the speedup value obtained in stage (f) considering the slowest machine type for T^* (M_1 in stage (d)) as the baseline.

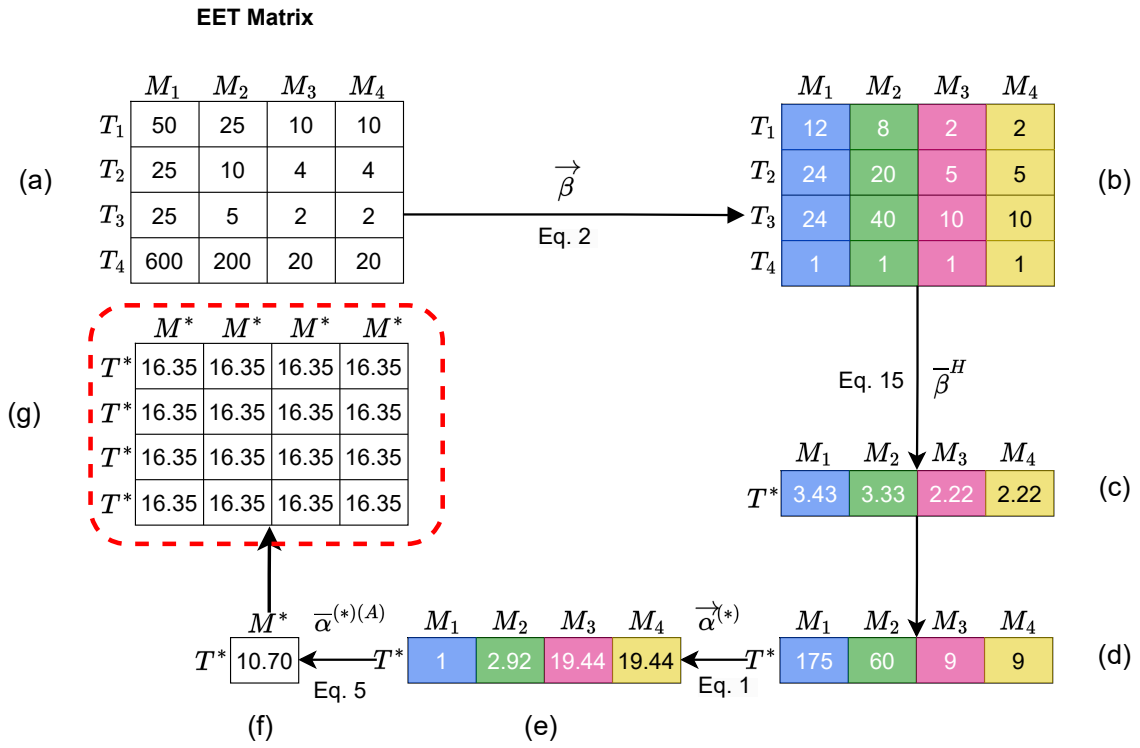
While HEET is able to accurately characterize the system heterogeneity, our hypothesis is that, for a given workload, systems with similar HEET scores exhibit similar makespan too. For a given workload trace with T task types, system A with the set of heterogeneous machine types M_A offers a bigger makespan than heterogeneous system B with machine types M_B ($|M_A| = |M_B|$), if and only if $HEET_A > HEET_B$.

5.2.4 HEET: Space Mapping From Heterogeneity to Homogeneity

As shown in Figure 5.1, the mathematical approach employed to obtain HEET is actually transforming the EET matrix representing the heterogeneous system (stage(a)) to a homogeneous EET matrix whose elements are HEET values (stage(g)). In other words, we proposed a mathematical formulation that transforms a heterogeneous computing system to a hypothetical homogeneous system such that both perform similarly in terms of performance metrics such as makespan.

Recall that HEET metric is the expected execution time of the hypothetical equivalent task type (T^*) on the hypothetical equivalent machine type (M^*).

Figure 5.1. An example illustrating the stages to calculate the heterogeneity measure (HEET). (a) The EET matrix representing a heterogeneous system. (b) The speedup matrix due to task heterogeneity is derived from EET matrix based on Equations 5.2. (c) Based on Equation 5.20, we consolidate each column into a mean value via harmonic mean. (d) Calculate the expected execution time of T^* on machines using the execution time of the slowest task type and $\overline{\beta}_j^H$. (e) The speedup vector due to machine heterogeneity for T^*, based on Equations 5.1. (f) we employ Equation 5.5 on the resultant speedup vectors to determine the mean speedup value due to machine heterogeneity. (g) HEET score represents the execution behavior of the heterogeneous system.



Replacing the machine types with M^* and task types with T^*, results in a homogeneous EET matrix, denoted by EET^*, whose elements are HEET values. Lemmas 5.1 and 5.4 prove that the homogeneous system represented by the EET^* matrix exhibits a similar makespan as the heterogeneous system represented by the EET matrix. Given a workload of c tasks, its makespan, denoted by τ, on a heterogeneous computing system can be estimated by the makespan of the same

workload on the homogeneous equivalent system represented by EET^*. For a homogeneous system of n machines (M^*), the expected makespan of executing c tasks of the same type (T^*) is the number of tasks distributed on each machine ($\frac{c}{n}$) multiplied by the expected execution time of that task type on the machine type (*i.e.,* HEET value). As a result, the makespan is determined as follows:

$$\tau = \frac{c}{n} \cdot HEET \tag{5.24}$$

Similarly, we can use the HEET score to estimate the throughput of the system. To that end, we estimate the throughput metric, denoted by θ, as the ratio of the number of tasks to the makespan of completing those tasks on the machines. Thus, the throughput is determined as follows:

$$\theta = \frac{n}{HEET} \tag{5.25}$$

5.3 System Design

In this section, we present an overview of the components we have designed to evaluate the HEET metric. Our implementation includes a real-world end-to-end "inference system", tailored to match real-world production scenarios [111]. Throughout our experimentation, we employed AWS EC2 instances as machines. It should be noted that, with slight modifications, the system can be readily deployed on alternative cloud platforms as well. The primary objective of our system is to validate the accuracy of the estimated makespan based on Equation 5.24 by comparing them against the actual makespan across various system configurations.

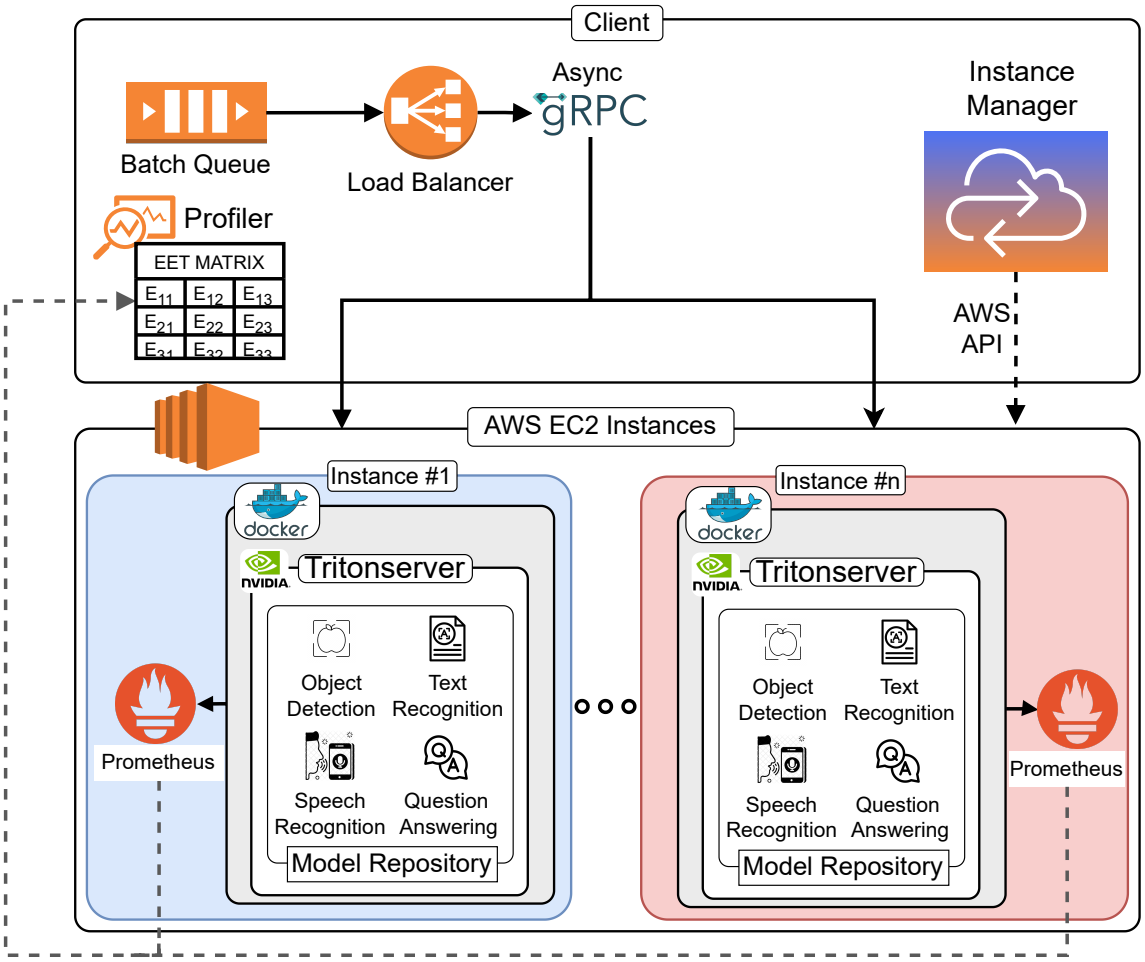
Figure 5.2 illustrates the overview architecture of the system. The components of the system are explained in the following paragraphs:

Model Loader To encompass a diverse range of model types, we utilized the extensive model repository offered by HuggingFace [112]. We used four different models in our experiment: (1) Image classification: Resnet50 [113], (2) Object detection: Yolov5 [114], (3) Question answering: DistilBERT [115], and (4) Speech recognition: Wav2vec2 [116]. Given the variety of deep learning frameworks from which these models were sourced, we sought consistency in our experiments. To achieve this, we converted all models to the ONNX (Open Neural Network Exchange) format [117] using the PyTorch ONNX converter [118]. Utilizing ONNX grants us the advantage of a unified model server setup, applicable across all model types.

Model Server Each of the models used during the experiments should be deployed as a machine learning service. We have leveraged the multi-model serving capability of modern inference systems [119] to encapsulate multiple models into a single inference service. Each of the services is backed by an AWS EC2 instance. In each machine, we spin up a containerized version of NVIDIA Triton Inference Server [120] and load all the model variants on top of it. Communications to the model servers are implemented using gRPC [121] due to its superior performance compared to other transfer protocols [122].

Workload We synthesized the workload traces assuming that the inter-arrival time between tasks in the workload traces follows an exponential distribution with the

Figure 5.2. Performance comparison of the same set of scheduling methods with the same workload across two computing systems, A and B, with different levels of heterogeneity (horizontal axis). The vertical axis shows the percentage of tasks completed on time.



mean arrival rate as its parameter [123]. In accordance with the bag-of-task [17] assumption, we have also designed a workload of tasks that are all available for execution from the beginning (*i.e.,* arrival time is zero). Tasks are sent asynchronously to the machines hosting the ML services based on their arrival times.

Monitoring The monitoring component in each of the EC2 instances is equipped with Prometheus [124], a highly-available time-series database. It records the

inference time of all the model servers during experiments and stored them in the database for later analysis. To construct the EET matrix, the Profiler benchmarks each task type on the machine types and employs Prometheus to obtain the expected inference times.

Instance Manager During the experiments, it is needed to reconfigure the heterogeneous computing systems with a different number of instances of each type. To support this, we have automated the process of reconfiguring the system in a central instance manager. The process of transitioning between two instance configurations includes (1) removing the current instances and cleaning the cluster (2) setting up the new sets of machines with required dependencies (3) bringing up the Triton container on top of the EC2 instances (4) and finally load the models to the Triton inference server. We have automated all the 1-4 steps through AWS Python SDK [125]. In addition, the EC2 instance types we used during our experiments are (1) t2.large, (2) c5.2xlarge, and (3) g4dn.xlarge. These instance types are resembling slow, medium, and fast machines in inferring the selected machine learning tasks.

Load Balancer Tasks are queued in a single unbounded FCFS queue upon arrival. Then, the load balancer assigns tasks to available machines in a Round Robin manner. Mapping events are triggered by either the completion or arrival of a task. In case a task arrives to the system while there is not any available machine, the load balancer defers the mapping event until a machine becomes free.

5.4 Experimental Validation

5.4.1 Experimental Setup

To validate the developed heterogeneity measure, in this section, we execute a workload of four deep learning applications on various combinations of three Amazon EC2 instance (machine) types to verify HEET score under real-world settings. Specifically, we used four different application/models in our experiment: (1) Image classification implemented using Resnet50 [113] model, (2) Object detection implemented according to Yolov5 [114] model, (3) Question answering based on DistilBERT [115] model, and (4) Speech recognition using Wav2vec2 [116] model. For machine types, we utilize GPU-based (`g4dn.xlarge`), compute-optimized (`c5.2xlarge`), and general-purpose CPU-based (`t2.large`) Virtual Machines offered by AWS EC2 services. To obtain the expected execution time of the image classification task on these machines, we processed 1000 sample images on each instance type. We repeat the experiment 10 times, and finally, we use the expected value of these 10,000 inference operations in the EET matrix. Similarly, for the object recognition task, we ran the object recognition task for 1000 sample images 10 times to determine the expected execution time of the object recognition task. For the speech recognition task type, we execute recorded audio of length 4 seconds on the machine types. Then, the average value of the inference times is used to fill the EET matrix. For question answering, we provide a sample context and question as the input of the inference task and run the inference task 1000 times. We aggregate the inference times and determine the expected execution

time for use in EET matrix. Upon the paper's acceptance, we will make all the datasets and the simulation source codes publicly available for reproducibility purposes.

To synthesize the workload trace, we assume that the inter-arrival time between tasks in the workload traces follows an exponential distribution with the mean arrival rate as its parameter [123]. For the bag-of-tasks, we assume that all tasks are available for execution from the beginning (arrival time is zero). Tasks are queued in a single unbounded FCFS arrival queue upon arrival, and assigned to the available machines in the Round Robin manner. Tasks are considered to be latency-sensitive with hard deadlines. The performance metric (makespan) is defined as the length of time the system requires to complete all tasks in the workload trace.

Recall that we employed hybrid central tendency measures (column-wise harmonic mean and row-wise arithmetic mean) to obtain a meaningful representative of the execution time behavior of heterogeneous computing systems. To illustrate the effectiveness of our method, we compare the experimental results with the following baselines as representatives of the execution time behavior of heterogeneous systems: (1) Arithmetic mean, (2) Harmonic mean, and (3) Geometric mean of the EET matrix elements. To that end, we determine three different heterogeneity measures based on the mean value of expected execution times of task types on machine types using different techniques as follows:

$$\overline{eet}_{arithemtic} = \frac{1}{m \cdot n} \cdot \sum_{i=1}^{m} \sum_{j=1}^{n} e_{ij} \tag{5.26}$$

$$\overline{eet}_{geometric} = \left[\prod_{j=1}^{n} \prod_{i=1}^{m} e_{ij} \right]^{\frac{1}{m \cdot n}} \tag{5.27}$$

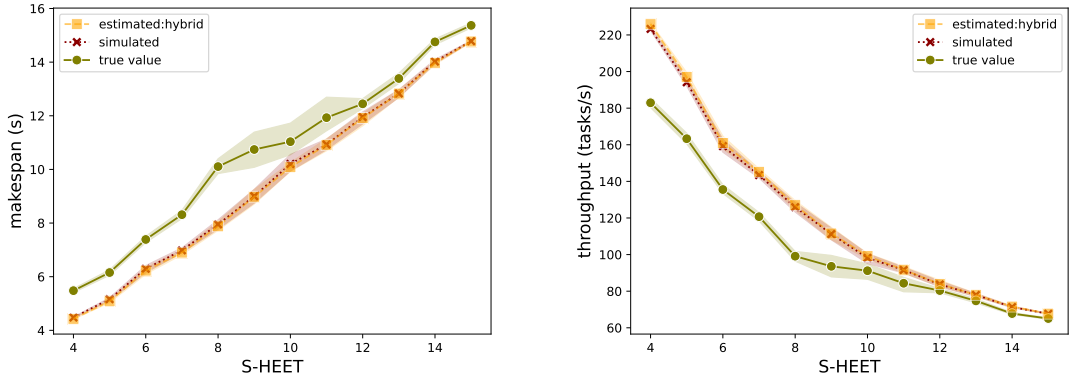
$$\overline{eet}_{harmonic} = \frac{m \cdot n}{\sum_{i=1}^{m} \sum_{j=1}^{n} \frac{1}{e_{ij}}} \tag{5.28}$$

5.4.2 Validating HEET as a Performance-Driven Heterogeneity Measure

In this experiment, our goal is to assess the ability of the HEET score to estimate true throughput and true makespan of the heterogeneous computing systems using Equations 5.25 and 5.24, respectively. To study the behavior of the heterogeneous systems upon varying HEET scores, we simulate a variety of heterogeneous systems with three machine types and four task types. To that end, we generate 228 distinct system configurations with a different number of instances of each type (*i.e.,* `t2.large`, `c5.2xlarge`, and `g4dn.xlarge`). Then, for each system configuration, we feed it with a bag of 1000 tasks of four types (*i.e.,* image classification, object detection, question answering, and speech recognition). Eventually, we average the makespan and throughput over different system configurations of the same HEET score.

Figure 5.3 shows the results of the true and estimated makespan and throughput with varying S-HEET score which is HEET score scaled with the number of machines (*i.e.,* $\frac{HEET}{n}$). Each (makespan, S-HEET)/(throughput,

Figure 5.3. The system performance in terms of the makespan (left) and throughput (right) (vertical axis) upon varying S-HEET scores (horizontal axis) for workloads 1000 tasks. Each individual point represents the average result of multiple computing systems with the same S-HEET score. In addition, the colored area illustrates the 95% confidence interval of the results.



(a) makespan **(b)** throughput

S-HEET) point shows the average makespan/throughput over different system configurations with the same S-HEET value. The colored area illustrates the 95% confidence interval of the results. As shown in Figure 5.3, we can observe that the systems with lower S-HEET scores generally perform better (*i.e.,* lead to a smaller makespan or higher throughput) than those with the higher HEET scores values. This statement itself means that the S-HEET score can be effectively used as a measure to "*compare different heterogeneous computing systems*" in terms of makespan or throughput. Moreover, we can observe that the results exhibit a narrow confidence interval, that is, systems with the same S-HEET score will perform similarly in terms of makespan and throughput. The number of different configurations with S-HEET scores equals to 9, 10, or 11 is small (less than 9), thus, we observe a wider confidence interval for these S-HEET scores in the results.

Moreover, the results show that the estimated values (makespan and throughput) using HEET score based on Equations 5.25 and 5.24, respectively, can predict the true values with an average accuracy of 84%. Note that we use the expected values of execution times to determine the HEET score. As a result, the accuracy of the estimation depends on the degrees of uncertainty that exist in the execution times. Accordingly, we run a simulation with zero uncertainty in the expected execution times to demonstrate the root cause of the error in estimating makespan and throughput using the HEET score. The results show that the makespan calculated using HEET score accurately estimates the makespan of the simulation. Thus, we can say that in heterogeneous systems with low levels of uncertainty in execution times, estimating makespan using HEET score is an accurate method.

Given a user-defined throughput threshold, we can determine the corresponding HEET score and use it to configure a heterogeneous system that can fulfill that desired throughput. For a desired throughput, the HEET score enables solution architects to proactively configure a heterogeneous system that can fulfill that objective (instead of try and error) with minimum cost.

In sum, the result of this experiment validates the applicability of the HEET score for real-world scenarios. In particular, when the HEET score is applied across systems, it can accurately compare different heterogeneous systems with respect to their performance metrics (makespan and throughput) without examining the workload on these systems.

5.4.3 Evaluating arithmetic, harmonic, and geometric means as heterogeneity measures

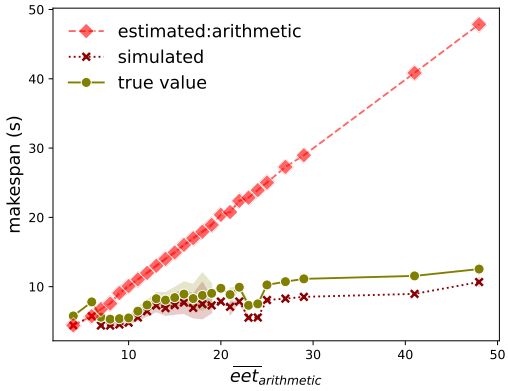
In this experiment, we investigate the effectiveness of arithmetic, harmonic, and geometric mean of expected execution times of task types on machine types as heterogeneity measures representing the execution behavior of heterogeneous systems. To that end, we conducted a similar experiment as Section 5.4.2 to study the performance (in terms of makespan) of 228 heterogeneous computing systems. An appropriate heterogeneity measure should be able to identify similarity and superiority in terms of performance metrics (*e.g.,* makespan or throughput) across heterogeneous systems.

Figure 5.4 shows the results of the makespan for different heterogeneous systems with varying mean expected execution times calculated based on Equations 5.26, 5.28, and 5.27. In Figure 5.4a, the results show that the estimated makespan using $\overline{eet}_{arithmetic}$ is unable to follow the true makespan and is considerably inaccurate. Similarly, we observe that harmonic mean and geometric mean heterogeneity measures are also inaccurate.

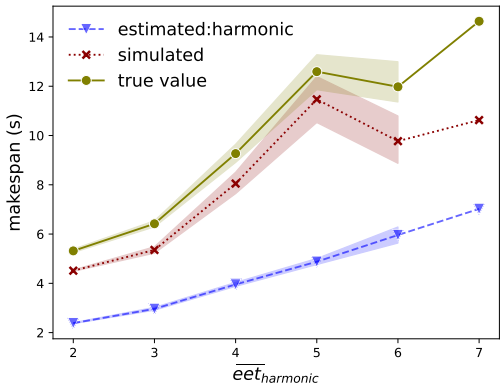
In sum, comparing the results for the baseline heterogeneity measures, as shown in Figure 5.4, with the results for HEET score, as shown in Figure 5.3, verifies that heterogeneous systems are well characterized with HEET measure, and can be used for estimating the performance of heterogeneous systems accurately.

5.4.4 Analyzing the behavior of the True Speedup due to Heterogeneity upon changes in the task arrival rates

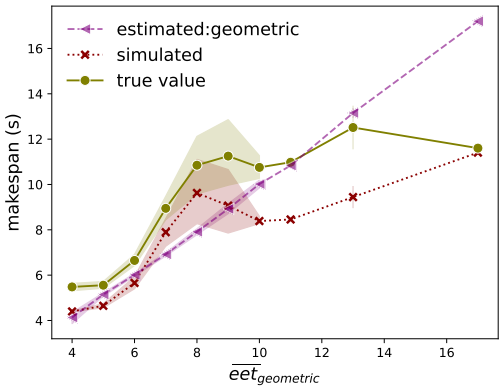
Figure 5.4. The system performance in terms of the makespan (vertical axis) upon the varying mean of EET matrix (horizontal axis) for a workload of 1000 tasks. In (a)–(c), $\overline{eet}_{\text{arithmetic}}$, $\overline{eet}_{\text{harmonic}}$, and $\overline{eet}_{\text{geometric}}$ are the arithmetic, harmonic, and geometric mean of the expected execution times of task types on machines types as represented in EET matrix. Each individual point represents the average result of multiple computing systems with the same mean EET value. In addition, the colored area illustrates the 95% confidence interval of the results.



(a) arithmetic mean



(b) harmonic mean



(c) geometric mean

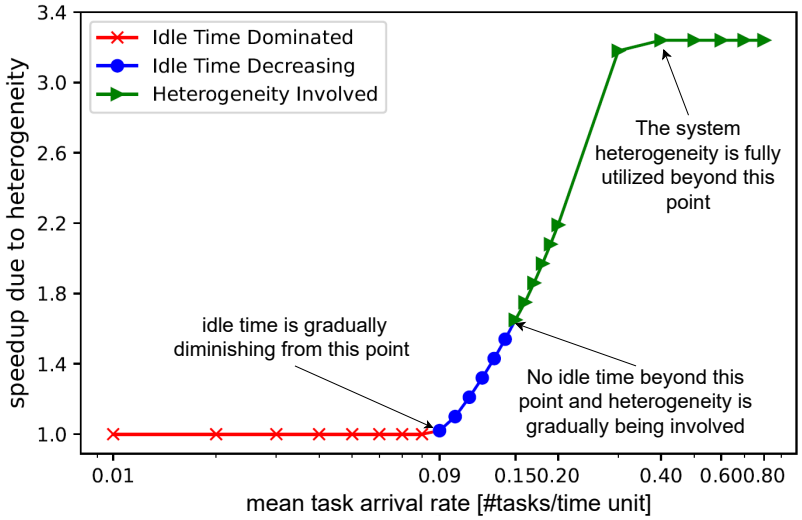
In Lemmas 5.1 and 5.4, we assume a high task arrival rate such that all tasks are available for execution. We showed that the mean speedup due to machine heterogeneity and task heterogeneity can be calculated using arithmetic mean and harmonic mean, respectively. In this experiment, our goal is to relax this

assumption and examine the impact of arrival rate on the system speedup and compare the true speedup against the calculated speedup due to system heterogeneity using HEET measure. For that purpose, we synthesized workload traces with different arrival rates where each workload trace includes 1,000 tasks of 10 types. Then, for each workload, we experimentally determine the makespan of processing that workload on an example heterogeneous system with four machine types and the EET matrix shown in Table 5.1. Based on the table, the system includes one fast machine (M_1) and three slower ones (M_2, M_3, and M_4). The estimated speedup due to heterogeneity for this EET matrix is 3.25. We repeat the simulation on a counterpart homogeneous system where the values of all entries in the EET matrix are replaced with the maximum expected execution time in the heterogeneous EET matrix (50.0). Then, we define the true speedup as the ratio of makespan on the homogeneous system to the one on the heterogeneous system.

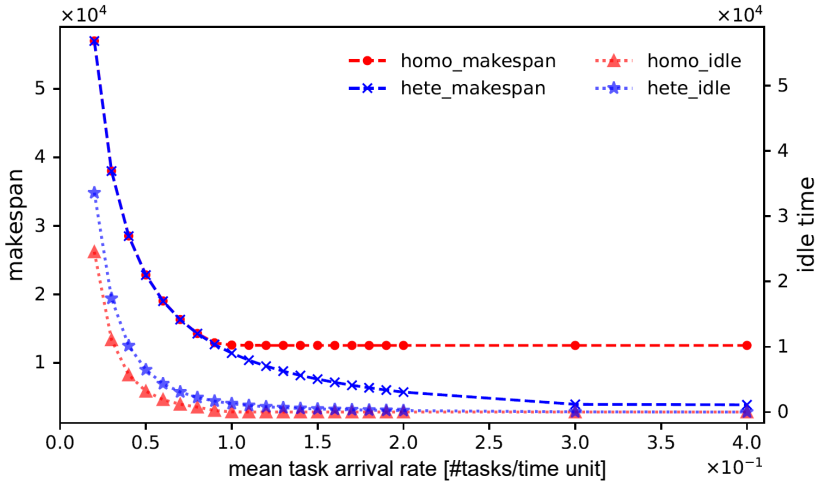
Table 5.1. EET matrix of the heterogeneous computing system used in the experiment 5.4.4The matrix represents a heterogeneous system with $\{M_1, M_2, M_3, M_4\}$ as its machine types and $\{T_1, T_2, ..., T_{10}\}$ as the task types arriving to the system.

Machines / Tasks	M_1	M_2	M_3	M_4
T_1	4.028	28.187	28.187	28.187
T_2	4.418	30.919	30.919	30.919
T_3	4.544	31.797	31.797	31.797
T_4	5.010	35.061	35.061	35.061
T_5	5.116	35.803	35.803	35.803
T_6	5.305	37.122	37.122	37.122
T_7	5.893	41.240	41.240	41.240
T_8	6.738	47.153	47.153	47.153
T_9	6.759	47.303	47.303	47.303
T_{10}	7.145	50.0	50.0	50.0

Figure 5.5. (a) The true speedup across different mean task arrival rates for the system represented by EET matrix shown in Table 5.1. We can see that for very low arrival rates, the speedup is negligible. By increasing the task arrival rates, heterogeneity is involved, and the speedup is increasing. (b) Makespan and idle times (vertical axes) for processing workloads with different mean task arrival rates on a heterogeneous computing system and its homogeneous counterpart. The idle time determines the total time that all machines are idle and no task is available for execution. For very low arrival rates, the makespan is dominated by idle time, hence, the impact of heterogeneity is negligible. However, upon increasing the task arrival rates, the makespan of heterogeneous systems decreases.



(a) True speedup across different mean task arrival rates



(b) Makespan and idle time for different mean task arrival rates

Figure 5.5a shows the true speedup upon changing the mean task arrival rates. We observe that, for very low arrival rates (red points in the figure), the speedup due to heterogeneity is one and it is not changing significantly. Figure 5.5b explains this phenomenon. It shows that, at low arrival rates, the makespan is dominated by idle times, which is because of high inter-arrival times between the tasks. We can conclude that, for underloaded systems, the matter of homogeneity and heterogeneity is unimportant. In Figure 5.5a, however, we observe that the speedup due to heterogeneity gradually increases at an arrival rate of 0.09. In fact, from this point onwards, the makespan is not dominated by idle time anymore. Although idle time is decreasing in the blue region, it is still considerable. Nonetheless, the starting point of the green region illustrates the task arrival rate where the idle time becomes negligible. We realized that in the green region, there are still some idle times in the system, therefore, the system is not fully utilizing its heterogeneous machines at all times. In fact, the start point of the green region demonstrates the situation where Lemma 5.3 is held to calculate the speedup due to machine heterogeneity. The value of estimated speedup due to heterogeneity, calculated using harmonic mean row- and column-wise, is 1.65, which is the same as the true speedup shown in Figure 5.5a for arrival rate 0.15. Upon increasing the arrival rate more than 0.15, the system makes use of more machines at all times, so that there is no idle machine in the system at any time. Ultimately, at arrival rate 0.4, in Figure 5.5a, we can see that the speedup converges to the calculated speedup using Lemmas 5.1 and 5.4 and its value is 3.25. In sum, we observe that the

proposed mathematical model can accurately describe the true speedup due to system heterogeneity. More specifically, we can use Lemma 5.1 under high task arrival rates and Lemma 5.3 under low task arrival rates to accurately determine the mean speedup due to machine heterogeneity, and Lemma 5.4 to represent the mean speedup due to task heterogeneity. Then, we utilize Equation 5.24 to calculate the makespan of the heterogeneous system. Finally, the overall speedup due to system heterogeneity is determined as the ratio of the largest expected execution time in the EET matrix (as the homogeneous baseline) to the HEET value.

5.5 Summary

Numerous system-level solutions (*e.g.,* load balancer, scheduler, etc.) have been developed to harness the heterogeneity and boost the performance of computing systems. However, the performance of such solutions across systems with different levels of heterogeneity is often unknown, because there is no concrete way to measure the system heterogeneity. As such, we provided a measure to quantify the system heterogeneity with respect to the performance metric (makespan or throughput) of the system and for a given set of task types. We characterized the system heterogeneity by decoupling the heterogeneity into machine and task dimensions. To quantify the performance impact of each heterogeneity dimension, we devised a speedup vector due to machine heterogeneity and task heterogeneity. We proved that the mean speedup due to machine heterogeneity under high workload arrival is measured by arithmetic mean. We also proved that harmonic mean can measure the mean speedup due to task heterogeneity. Then, we leveraged

the mean speedup due to task heterogeneity to reduce all task types in a hypothetical equivalent task type that can characterize the execution time behavior of the set of task types. Next, we employ the mean speedup due to machine heterogeneity to characterize the set of machine types into a single hypothetical equivalent machine type. Eventually, we introduce the Homogeneous Equivalent Execution Time (HEET) score as a heterogeneity measure representing how fast a heterogeneous system is for a given set of task types. In this way, we transform a heterogeneous system into a hypothetical equivalent homogeneous system with similar makespan. HEET can be used to globally compare the performance of different heterogeneous systems. We observed that the HEET score is effective and can approximate the makespan with an average and minimum accuracy of 84% and 80.0%, respectively. For a desired throughput objective, the HEET score enables solution architects to proactively configure a heterogeneous system (instead of trial and error) that can fulfill that objective.

Chapter 6: E2C – A Visual Simulator to Reinforce Education of Heterogeneous Computing Systems

6.1 Introduction

harnessing system heterogeneity has been a longstanding challenge in distributed systems (*e.g.,* [30, 126, 11]), and educating it to Computer Science/Engineering (and more broadly STEM) students, and researchers has become necessary. Making use of real infrastructure (such as those offered by the public cloud providers) for benchmarking the performance of heterogeneous systems, for different applications, with respect to different objectives, and under various workload intensities is cost- and time-prohibitive. As an example, consider an IoT-based system that offers multiple smart applications to its users (*e.g.,* object detection, face recognition, speech recognition, etc.); there exists a wide range of machine types with different architectures (such as x-86 or ARM-based multi-core CPUs, different types of GPUs, FPGAs, and ASICs) that can process these services. To find an optimal configuration, a student must examine all permutations of these configurations. Moreover, there can be multiple workload intensities and scheduling policies that can affect performance of the system and the student must examine them too. Last but not least, learning about the energy consumption of the heterogeneous computing system in question adds another dimension to the evaluation process that needs to be conducted by the student.

To avoid the burden of examining all cases, we need simulation tools that can help the students and researchers to study the performance of various system

configurations and effectively learn about impacts of heterogeneity in a distributed system. To that end, in this chapter, we introduce E2C that is an open-source discrete event simulator that simulate any type of heterogeneous (and homogeneous) computing system. By using E2C, the students can easily examine their system-level solutions (scheduling, load balancing, scalability, etc.) in a controlled environment within a short time and at no cost. In particular, E2C offers the following features: (i) defining user-defined workload generation scenarios with various number of applications (a.k.a. task types) and arrival intensities; (ii) simulating a heterogeneous computing system; (iii) implementing a newly developed scheduling method and plugging it into the system, (iv) measuring power and other output-related things, and (v) visual aspects to ease the learning curve for students. These features help students who study resource allocation solutions in distributed systems to test and evaluate their solutions easier and faster. Moreover, the graphical user interface would help students to gain a deeper knowledge of resource allocation procedures in distributed computing systems.

We used E2C as an assignment in our Distributed and Cloud Computing class to examine various types of scheduling methods for heterogeneous (and homogeneous) systems under various workload intensities. We conducted a survey on the learning outcomes of the simulator and its usability aspects. Analysis of the survey results showed that the students on average rated E2C with the score of 8.7 out of 10 for its usefulness in comprehending scheduling methods for heterogeneous and homogeneous computing systems under different workload intensities.

Table 6.1. Positioning of E2C with respect to other simulation tools for distributed systems.

Simulator	**Prog. Language**	**GUI**	**supporting heterogeneous computing**	**workload generator**
CloudSim	Java	✗	✗	limited
iFogSim	Java	✗	✗	limited
EdgeCloudSim	Java	✗	✗	✓
iCanCloud	C++	✓	✗	✗
TeachCloud	Java	✓	✗	limited
E2C	Python	✓	✓	✓

Moreover, based on the survey results, students assessed that E2C is easy to use with the average score of 8.3 out of 10, and they evaluated their willingness for recommending E2C to others with the average score of 8.3 out of 10.

In the rest of this chapter, in Section 6.2, we first position E2C with respect to other existing simulators. Next, we elaborate on the features of E2C in more detail in Section 6.3. Then, in Section 6.4, we describe our experience of using E2C as a class assignment for Computer Science and Engineering students. In Section 6.5, the evaluation of E2C and the results we obtained are discussed.

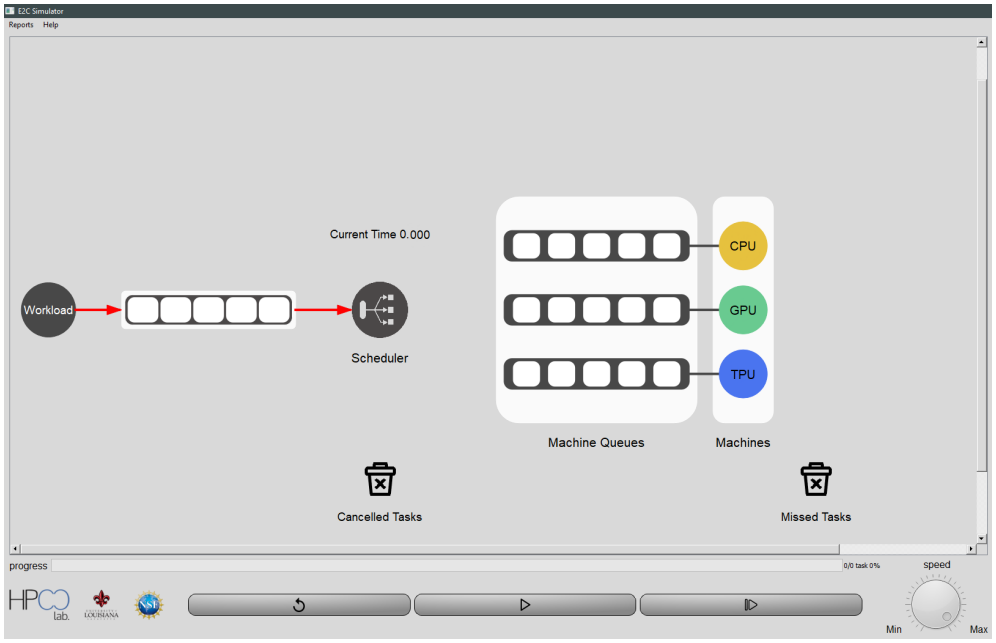
6.2 Positioning E2C with respect to other existing simulators

There are several existing cloud simulators that have been developed to provide researchers and developers with a platform to simulate and test cloud computing environments. Some of the popular cloud simulators include CloudSim [127], EdgeCloudSim [128], iFogSim [129], iCanCloud [130], and TeachCloud [131]. Table 6.1 provides a quick positioning of E2C with respect to these simulation tools.

CloudSim is a popular open-source framework used for modeling and simulating cloud computing environments and applications. With its modular and extensible architecture, CloudSim allows users to customize and configure different aspects of the simulation, such as virtual machine management, workload scheduling, and resource allocation policies, to suit their research needs. However, as a Java-based framework, it needs the user-input in the form of Java lines of the code within the back-end for configuring and customizing the environment. As a results, the users (e.g. students) should already have background experience and knowledge of Java and object-oriented programming (OOP). While this can provide its own kind of learning experience, it is not necessarily helpful for teaching about cloud computing and distributed systems, and may only slow down education concentrated in that area. EdgeCloudSim is another simulation platform that is tailored to Edge computing systems. EdgeCloudSim is based on the CloudSim with more functionalities in network modeling and load generator. iFogSim is another CloudSim-based simulation tool that is utilized for modelling and simulation of Fog computing environments, and evaluating the efficiency of different resource management policies in terms of latency (timeliness), energy consumption, network congestion and operational costs. iCanCloud is a cloud computing simulation framework for generating and customizing a large distributed computing system written in C++. iCanCloud comes with a user-friendly GUI which is useful in managing pre-configured virtual cloud systems and generating graphical reports. Although iCanCloud supports consistent heterogeneity in terms of configuring VMs

with varying number of CPU cores, it does not support inconsistent heterogeneity by having VMs with accelerators (e.g. GPUs and FPGAs). Jararweh et al. developed a simulation toolkit, called TeachCloud [131], for cloud computing environment equipped with a GUI that allow students easily create the main components in the cloud system. However, TeachCloud lacks supporting the heterogeneous computing systems. In general, these simulators provide valuable insights into the performance

Figure 6.1. Overview of the E2C Simulator that includes major components, being the source workload, a batch queue of arriving tasks, scheduler (a.k.a. load balancer), and a set of heterogeneous machines, represented with different colors. Each machine has a "machine queue" where the assigned tasks are queued for the execution.



and efficiency of cloud computing systems, enabling researchers and developers to make informed decisions when designing and implementing cloud-based solutions. However, there are limitations in using these simulators as an educational tool for teaching heterogeneous distributed computing systems. As per limitations of existing simulators, they lack either a user-friendly graphical interface to make use

of the simulator easy and intuitive for the students or supporting heterogeneous computing systems. To overcome these limitations, we developed E2C, a simulator explicitly designed for heterogeneous computing systems with an intuitive graphical user interface (GUI). E2C is intended to facilitate the study of heterogeneous computing systems for students by enabling them to simulate and explore the characteristics of this type of computing systems through a user-friendly interface.

E2C comes with a GUI, that requires no programming input from the user. All inputs can be done directly from the GUI. In addition, the GUI displays simulations in live time, making it well suited for education, as many students perform better through visual learning. E2C also aims to be granular, allowing the user to configure the system in many specific ways. This is also important for researchers, given that a simulated system should be highly configurable in order to handle a wide variety of workloads.

6.3 Simulating a Heterogeneous Computing System via E2C

Figure 6.1 shows an overview of the E2C simulator that includes the following major components: (i) workload, (ii) batch queue, (iii) scheduler, (iv) machine queue, and (v) a set of (homogeneous or heterogeneous) machines. In addition, there are two more components that contain canceled and dropped tasks. This is to support circumstances where tasks have hard deadlines and there is no value in executing them beyond their deadline.

A workload is defined as a large group of tasks where each task is a request for an application (task type). In the real world, a heterogeneous computing system

can be configured to execute several task types. For instance, a heterogeneous system processing satellite images should support task types for object detection, noise removal, and image enhancements to be performed on the received images. E2C enables us to define the task types, arrival distribution for each task type, and their arrival duration. Each task in the generated workload of E2C has an arrival time and deadline as well.

The machines in the distributed system can be identical (homogeneous) or non-identical (heterogeneous). Note that the heterogeneity of the system is modeled by a matrix, called the Expected Execution Time (EET) matrix [1, 65, 71]. This matrix defines the expected execution time of each task type on each machine. This is to model a real world heterogeneous system, where any given task type (e.g., object detection, noise removal, etc.) is expected to have a differing execution time across heterogeneous machines. The opposite holds true for a homogeneous system where any given task type has identical execution time across all machines. As shown in Figure 6.2, the user has access to the EET matrix by selecting the workload component. Users can either modify the EET matrix manually or load the desired one as a CSV file.

As shown in Figure 6.2, the user can load the desired workload trace as a CSV file in this section. The user must keep in mind that the workload trace must conform to the EET matrix. That is, there can be no task type within the workload that is not defined within the EET. Upon the arrival of a task, the simulator transfers the task to the batch queue. The batch queue is where tasks are held

before being scheduled. Next, based on the selected scheduling method, the scheduler selects a task from the arrival queue.

Figure 6.3 shows the scheduler options. The user can choose between immediate scheduling or batch scheduling [132]. Immediate scheduling is when incoming tasks are immediately scheduled to a machine upon arrival, whereas, with batch scheduling, tasks are buffered in the batch queue so the scheduler can make a more informed decision. Typically, immediate mode scheduling methods impose a lower overhead and generally load balancers use this type of scheduling [132]. The following immediate policies are currently implemented into E2C as options: FirstCome-FirstServe (FCFS), Min-Expected-Completion-Time (MECT), and Min-Expected-Execution-Time (MEET). For batch policies, E2C currently implements: ELARE, FELARE, MinCompletion-MinCompletion (MM), MinCompletion-MaxUrgency (MMU), and MinCompletion-SoonestDeadline (MSD). An explanation of these methods can be found in [30].

There exist two options for the scheduled tasks: (i) it might be canceled because of missing its deadline before assignment; or (ii) it might be mapped to one of the available machines. The status of a canceled task is set to "canceled" and no more process is needed. The canceled tasks component shows the number of tasks have been canceled so far. In the case of mapping decisions, the task is appended to the local queue of the assigned machine until the machine queue is saturated. Tasks are executed on the assigned machine in a sequential manner by default. If a task missed its deadline while executing on the machine, it is dropped from the machine.

Figure 6.2. Workload component. Here, the user can load EET and Workload CSV files. The user can also modify the EET matrix and arrival times of the task with the "Edit" button. Upon loading new CSV files or editing values, the user must press the "Submit" button. EET and Workload files must be compatible. `T1, T2, T3` represent different task types in this simulation.

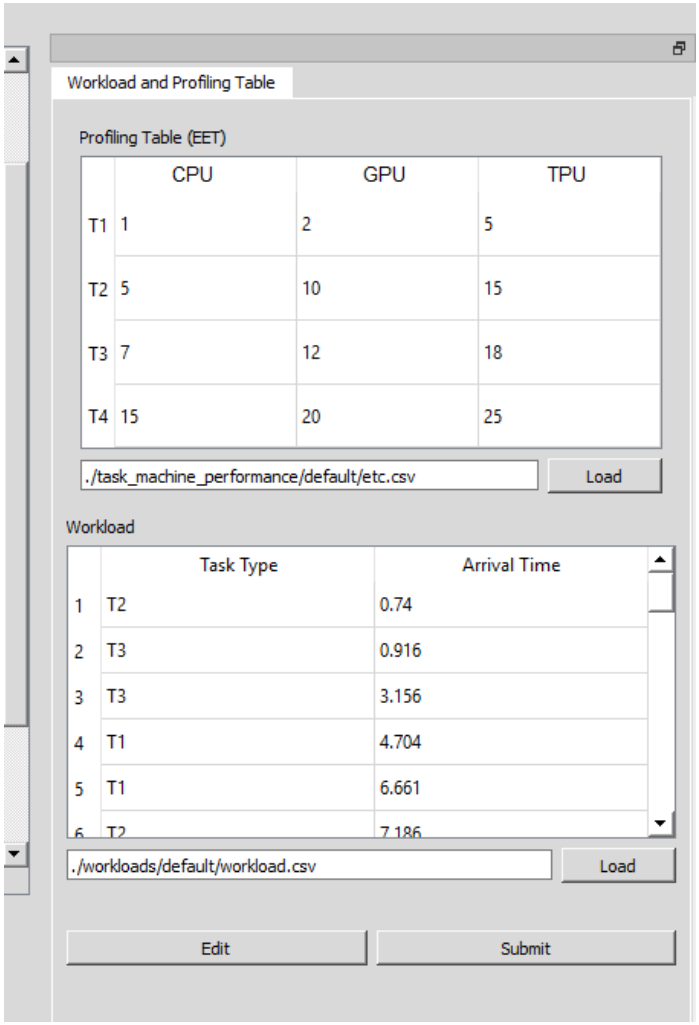


Figure 6.3. Scheduler component. Here the user may select between the various immediate or batch scheduling policies, along with setting the machine queue size. The machine queue size is limited to infinite for immediate policies, but can be changed for batch policies.

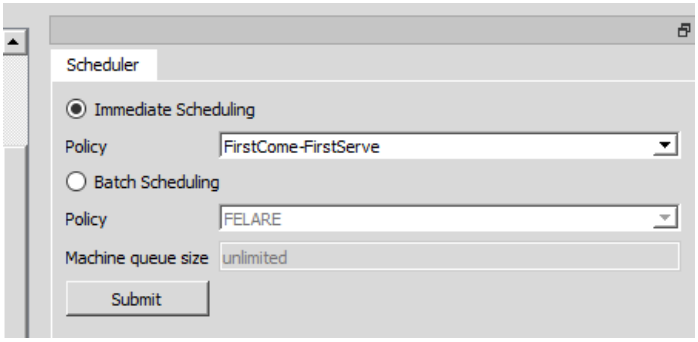
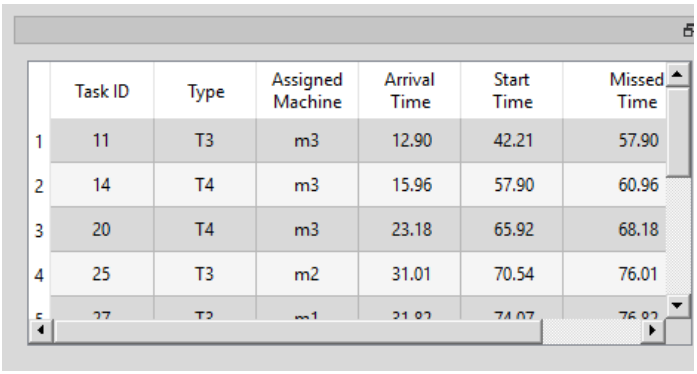


Figure 6.4. Missed Tasks component shows the task ID that missed its deadline, along with its task type, assigned machine, arrival time, start time, and the time when it missed.



	Task ID	Type	Assigned Machine	Arrival Time	Start Time	Missed Time
1	11	T3	m3	12.90	42.21	57.90
2	14	T4	m3	15.96	57.90	60.96
3	20	T4	m3	23.18	65.92	68.18
4	25	T3	m2	31.01	70.54	76.01

As shown in Figure 6.4, the Missed Tasks component shows the tasks that missed their deadline.

Importantly, E2C is designed to be modular, hence, providing the ability for the user to modify the existing scheduling methods or adding their own custom-designed scheduling methods. This feature is particularly helpful for researchers to examine new methods under various conditions and configurations.

After the user selects and submits the EET and workload, they will press the "Play" button near the bottom-middle of the GUI. This will begin the animation of

tasks flowing from the incoming workload to scheduler to machines, along with the "Current Time" which will update continuously during simulation. If you press the "Play" button again during the simulation run-time, the simulation will be paused. The button right of the play button is the "Increment" button, which when pressed while the simulation is paused will perform the next individual step that would performed (i.e. a task being submitted to a machine by the scheduler, or a task's execution being completed by a machine, etc.). This can be helpful if you wish to analyze each specific action of the simulation. To the left of the "Play" button is the "Reset" button, which can be used either during a pause or after completion of a simulation. This will allow you to begin a new simulation, also allowing you load in a new EET and/or workload should you choose. Along with these three options, during the simulation run-time, you can choose to alter the speed at which the simulation runs by using the speed dial located at the bottom right. This can be useful for either getting quicker results or for better visibility of the animated simulation.

Upon completion of a simulation within E2C, the user may view a report, and optionally, save the report as a CSV file. There is an option for a "Full Report," "Task Report," "Machine Report," and "Summary Report." The Full Report displays the majority of relevant information regarding the simulation - this is the option to view all data related to each task and and how each machine performed on it. The Task Report displays information that is more centric to the individual tasks of the workload, whereas the Machine Report displays data more

relevant to the machines of the system. Lastly, the Summary Report displays a summary of the workload data without the specifics of each individual task.

The E2C simulator can be implemented as a learning tool for undergraduate and graduate students, and also serve practical solutions for researchers and practitioners. Through E2C, students can gain the ability to analyze, design, implement, and test distributed computer systems and components. They can deeply investigate scheduling methods, how they work, and gain insights into their advantages and disadvantages. Along with this, they can develop their own scheduling method(s) and use E2C as a means to implement it. Students can also learn how heterogeneity can improve the performance of the system through defining machines that have better performance for executing specific task types. Moreover, they can study the energy consumption of the system once a certain scheduling method is applied, allowing them to learn about resource management. So far, we have used the E2C simulator for students in "Distributed and Cloud Computing" courses to examine the impact of different scheduling policies on homogeneous and heterogeneous systems with various workload intensities. Similarly, the simulator can be used for the "Operating Systems" and "Computer Networks" courses at the undergraduate and graduate levels to teach students about the impact of scheduling at different levels.

Researchers in the resource allocation area and cloud solution architects can employ the E2C simulator to test their solution prior to implementation. Being highly customizable, they can configure E2C to represent its real world counterpart.

Through this, they may test the outcome of a heterogeneous system with different scheduling methods without spending real resources, saving both money and time. The outcome to be tested can be things such as QoS through task completion percentage (versus missed and cancelled tasks), energy consumption of machines (resource management), and how different scheduling methods perform on any given system. This way, researchers can apply practical use of E2C in order to help design and compare their own real world distributed systems or clusters. As an example, in [133], we have used E2C to examine energy efficiency and fairness of scheduling methods on a heterogeneous edge. Also, in [134], we extended E2C to simulate the memory allocation policies of multi-tenant applications on a homogeneous edge computing system.

6.4 Class Assignment for Computer Science and Engineering Students

The E2C was used, and will continue to be used, by undergraduate and graduate students of the University of Lafayette's Distributed Cloud Computing course. Before E2C was implemented as an assignment for the students, there were no assignments for evaluating the students' understanding of heterogeneous systems and scheduling methods through simulation. Now, with the addition of E2C, students have a means to learn these subjects through coursework. In this assignment, E2C was used to teach students about the impact of various scheduling methods in heterogeneous and homogeneous computing systems operating under various workload intensities. It also asked the graduate students to develop and implement their own scheduling policies and compare it with the existing solutions.

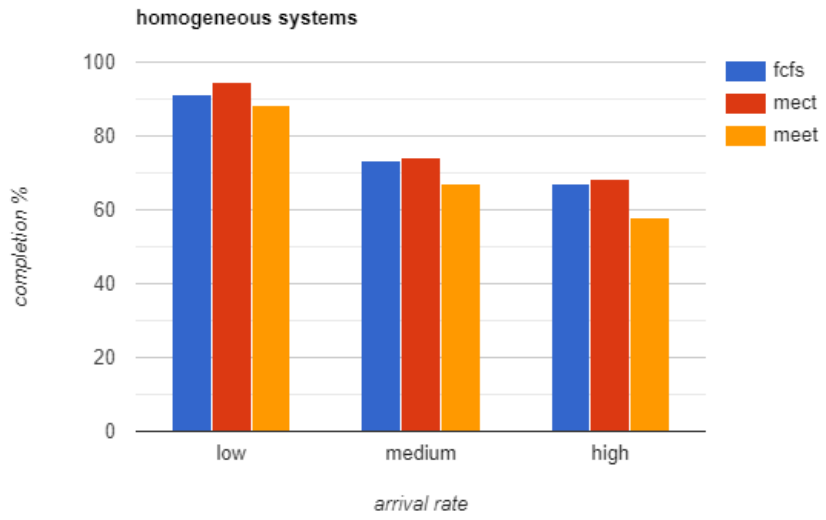
The installation and graphical user-interface of E2C is user friendly and works on any operating system, which makes it easy to pick up and use for projects or assignments. We have created a web-based documentation[a] where all the features of the simulator—from installation to reporting—are explained.

In this assignment, students were to read and learn about the basic components of E2C, being task types, machines, EET matrix, workload trace, and task deadlines. The students would then use the simulator to evaluate the different scheduling methods currently implemented by E2C on both a homogeneous and a heterogeneous system. For the homogeneous system, students were to use three workload traces with arrival intensities ranging from `low, medium,` to `high` to stress the system at different levels. For each arrival intensity level, they ran the simulation and saved the CSV output files, provided by E2C, summarizing all the data related to the simulation for three different immediate scheduling methods, namely FirstCome-FirstServe (FCFS), Minimum-Expected-Completion-Time (MECT), and Minimum-Expected-Execution-Time (MEET). Students then created a bar graphs to depict the percentage of completed tasks that each scheduling method results under each intensity level. The expected results is that higher intensity workloads lead to a lower completion rate (i.e., more tasks missing their deadlines). In addition to observing this behavior, the students had to analyze and report the behavior of different scheduling methods.

For the next part of the assignment, they would do similarly but with a

[a]E2C documentation can be accessed at: `https://hpcclab.github.io/E2C-Sim-docs/`

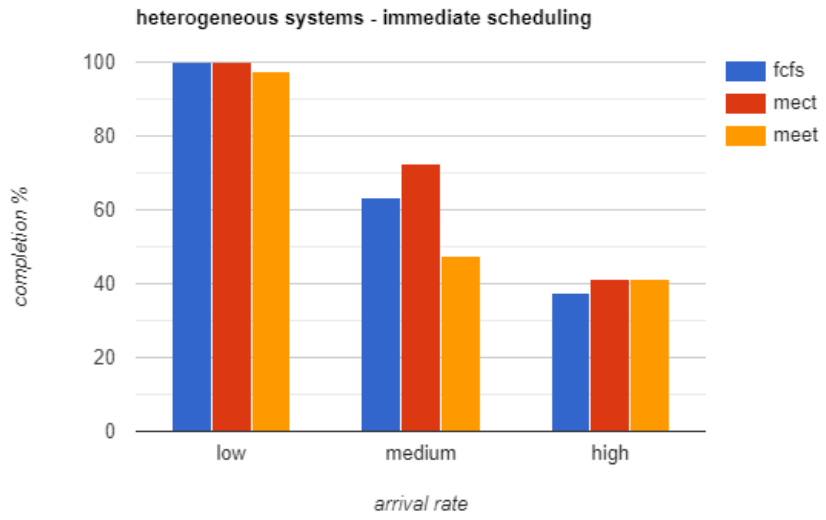
Figure 6.5. A bar graph with completion % for immediate scheduling methods on a homogeneous system, showing results for varying intensities using FCFS, MECT, and MEET policies.



heterogeneous system instead. For this part, in addition to the immediate scheduling policies, they would also be testing the batch mode policies: MinCompletion-MinCompletion (MM), MinCompletion-MaxUrgency (MMU), and MinCompletion-SoonestDeadline (MSD). Required by the graduate students and optional to undergraduates as a bonus, the third part of this assignment was to create and implement their own scheduling method for the heterogeneous system that enabled fairness across various task types in the system. After these simulations and implementations were complete, students were to perform an analysis of their findings on both the homogeneous system and heterogeneous system, and answer questions that show what they have learned about scheduling and its related methods.

The creation of graphs to evaluate their findings is straightforward due to the

Figure 6.6. A bar graph with completion % for immediate scheduling methods on a heterogeneous system, showing results for varying intensities using FCFS, MECT, and MEET policies.

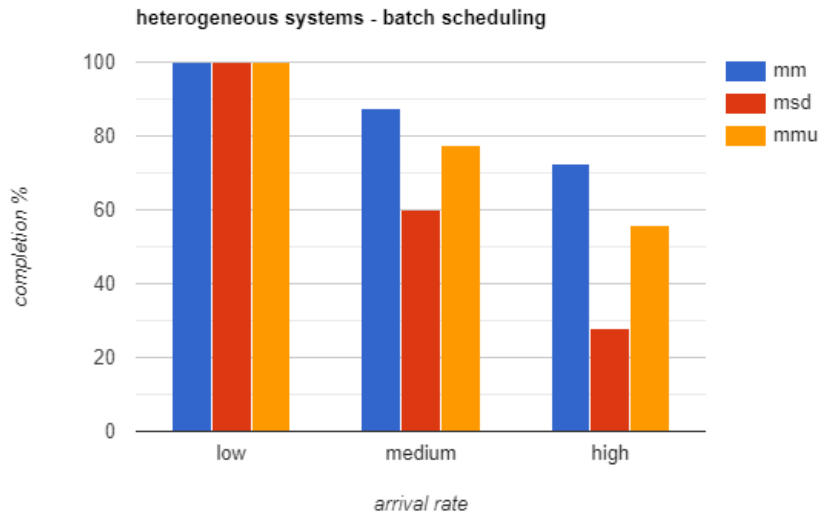


way saving data from simulations is within E2C. Once a simulation is complete, all students needed to do is go to the reports menu and save the report as a CSV file.

For the bar graphs that the students create for both their findings on homogeneous and heterogeneous systems, they plot the completion percentage (completed tasks/total tasks in workload) for each scheduling method. Some examples of their findings show a bar graph depicting completion percentage for immediate scheduling policies on a homogeneous system (Figure 6.5), immediate scheduling policies on a heterogeneous system (Figure 6.6), and batch scheduling policies on a heterogeneous system (Figure 6.7).

The learning outcomes of this assignment was to understand the impact of different scheduling methods in face of homogeneous and heterogeneous systems, and to analyze the advantages and disadvantages of each. For instance, they

Figure 6.7. A bar graph with completion% for batch scheduling methods on a heterogeneous system, showing results for varying intensities using MMU, MSD, and MMU policies.

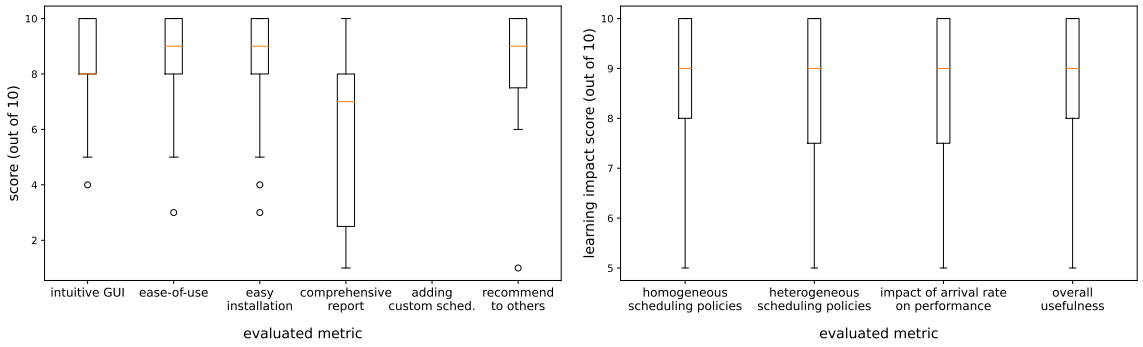


analyzed why Minimum-Expected-Completion-Time (MECT) performs better than FirstCome-FirstServe (FCFS) method, and why the batch policies outperform immediate scheduling policies for heterogeneous systems.

6.5 evaluating learning outcomes of E2C

As mentioned in Section 6.4, E2C have been examined as an assignment in the Distributed and Cloud computing course. After the assignment, we conducted a survey across the students to evaluate the impact of E2C on their learning. 23 students (14 undergraduate students and 9 graduate students) participated in this survey study. The demography of the 23 students are as follows: (i) Gender: 73.9% students were male and 26.1% of them were female; (ii) Degree level: 60.9% students enrolled in Bachelor's degree (undergraduate) and 39.1% were pursuing higher level of education including master and doctoral degree (graduate); (iii)

Figure 6.8. Illustration of the survey on the students' experience in accomplishing their distributed systems assignment via E2C (a) This subfigure demonstrate the HCI experience of the students with E2C (b) This subfigure shows how much E2C simulator could help students in understanding the characteristics of task scheduling policies in the homogeneous and heterogeneous configurations.



(a) Evaluation of user experience with E2C **(b)** Evaluation of learning objectives

Programming experience: The mean and median values of the students' programming experience are 3.8 and 3 years, respectively; and (iv) Passed Operating System (OS) course: 43.5% of the students have already completed the OS course and 56.5% of them have not previously passed that course. The questions of the survey[b] were in two categories: (i) Those related to the user interactions (experience) with the E2C simulator that is shown in Figure 6.8a; and (ii) Those focuses on the specific learning outcomes, i.e., how much the knowledge of students was improved as a result of doing this assignment. The result of this category is shown in Figure 6.8b. All students were asked to rate E2C with respect to each evaluation metric in the scale of 10.

The user experience part studies the user-friendliness of the E2C interface and how it makes technical concepts intuitive. Installing E2C is the first experience

[b]The complete survey can be retrieved from https://drive.google.com/file/d/1iW3pHFb7UicnmlUf6xIA$_SZIXf7PoD4/view?usp = sharinghere$.

of such nature. Figure 6.8a shows that students on average evaluated that the installation part is an easy and straightforward procedure with the score of 8.3, that is applicable for any operating system. The intuitive Graphical User Interface (GUI) of E2C is another metric of the user experience. The overall average score of 8.35 for this metric shows that the students has had no difficulty in dealing with the E2C through its GUI. As per gender assessment, female students assessed the GUI intuitive and easy to use with the average score of 9.3 while male students rated it as 8. Moreover, the average score of 8.3 (female average score:9.3, male average score: 7.9) for ease-of-use metric demonstrate that students assess the overall technical part of E2C is intuitive and easy to understand. However, the students assessed the report section with the average score of 5.7 (female average score:4.8, male average score: 5.9). Although the reports are comprehensive, we realized that the structure of the GUI for the report section is not intuitive, therefore, the students could not find their required reports easily. To address this issue, we are rearranging the report section in the GUI and make different reports and their fields more informative. In case of developing a custom scheduling in E2C, the graduate students responded that E2C was useful, with the average score of 8.3 (female average score:9.2, male average score: 7.4), in implementing and evaluating their custom scheduling policy. In general, the students evaluated their willingness for recommending E2C to others with the average score of 8.3 (female average score: 9.7, male average score: 7.8), as shown in Figure 6.8a.

Figure 6.8b summarizes the students' responses in terms of their learning

outcomes. The results show that they found E2C helpful in understanding the impact of scheduling methods in heterogeneous and homogeneous systems with the median score of 8.7 (female average score:9.8, male average score: 8.2) and 8 (female average score:9.5, male average score: 8.4), respectively. In addition, as explained in Section 6.4, they utilized three workload traces with varying arrival intensities to learn about the impact of arrival rate on the system performance in terms of on-time completion rate. As shown in Figure 6.8b, they responded that E2C could help them in understanding the impact of arrival rate on the system performance with the average score of 8.6 (female average score:9.7, male average score: 8.2). Overall, based on the survey results, shown in Figure 6.8b, students assessed E2C is useful in developing their knowledge in the distributed systems course with the median score of 8.8 (female average score:9.5, male average score: 8.6). More specifically, as shown in the results, female students assessed E2C as a an easy-to-use and useful learning tool with higher median score than male students. In other words, the gender-based results show that E2C is more effective for female students.

We asked students similar scheduling questions in the form of two quizzes, taken before and after using E2C as a course assignment. The quizzes asked the students to map three arriving tasks to four heterogeneous machines via the following scheduling methods: MEET, MECT, MM, and MSD. The average score of students has improved from 7.6 (out 12 points) in the first quiz to 8.94 in the second quiz. The results imply that E2C could improve the students' learning of scheduling methods in heterogeneous computing systems by 17.6%.

At the end of the survey study, we asked them to write us their feelings and suggestions that they would like to see in the next version of the E2C simulator. Here, is the main suggestions we received from them: "The simulator clarified the working of different scheduling methods well with its visual animation." "The application was intuitive when it comes to the context of this course and it was relatively easy to use." "I must commend the great work done by the everyone at the HPCC lab that contributed to the E2C simulator. This is a wonderful software." As for the suggestions, students reported several bugs that we already fixed. Some others had suggestions to make the GUI more intuitive, *e.g.,* by changing the mouse pointer when it is hovered on various components; also there were suggestions to enable drag and drop feature to the simulation scenario.

6.6 Summary

E2C provides a free (open-source) learning tool for students enrolled in courses like Distributed Systems, Operating Systems, and Computer Networks as well as researchers by delivering an intuitive way to simulate heterogeneous and homogeneous systems. It particularly helps the students to gain insight into the performance of different scheduling methods upon various heterogeneous systems and under various workload intensities without the need to use and expend for real infrastructure. As such E2C is a step towards reducing the widening educational gap nationally, and even at the global scale. The users of this system can employ several existing scheduling methods built into the simulator, but also have the ability to develop and test their own custom method. As we experienced it in our

Distributed and Cloud Computing class, it is an effective accompaniment that can remarkably improve the knowledge of students in the area of heterogeneous computing and scheduling. E2C comes with user friendly GUI for quick usage by beginners, but is also configurable enough to meet the needs of researchers and practitioners in the field. Based on the feedback we received from our students, we plan to extend E2C with several other features, including various communication paradigms and the ability to drag and drop components into the simulator.

Chapter 7: Conclusion and Future Research Directions

This chapter summarizes the research and major findings of this dissertation. Additionally, research topics that have surfaced during this research but have not been covered in this dissertation are brought up and discussed. These potential pathways for the future can be investigated further by other researchers working in this field.

7.1 Discussion

In this dissertation, our main objective was to take a holistic approach to heterogeneity and harness it as a result of multiple cooperation between middleware solutions. Our approach to harness heterogeneity can be categorized into three sections: (i) Maintain the robustness of the heterogeneous computing systems against uncertainties in tasks' execution time and arrival time, (ii) Devise fair energy- and latency-aware scheduling in heterogeneous computing systems, and (iii) quantify the impact of the heterogeneity on the system performance (*i.e.,* QoS).

In Chapter 3, we studied the robustness of distributed computing systems against uncertainties in tasks' execution time and arrival time. In this chapter, we considered task execution time as a random variable and used probabilistic analysis to develop an autonomous proactive task-dropping mechanism to attain our robustness goal. Experimental results demonstrated that the autonomous proactive dropping mechanism could improve the system robustness by up to 20%.

In Chapter 4, we explored fairness in the scheduling of latency-sensitive and concurrent Machine Learning (ML) applications on battery-powered heterogeneous

Edge systems. To that end, we investigated edge-friendly (lightweight) multi-objective mapping heuristics that did not become biased toward a particular application type to achieve the objectives; instead, the heuristics considered "fairness" across the concurrent ML applications in their mapping decisions. Moreover, we studied and analyzed resource allocation solutions that can increase the on-time task completion rate while considering the energy constraint. Performance evaluations demonstrated that the proposed heuristic outperforms widely used heuristics in heterogeneous systems in terms of latency and energy objectives.

In Chapter 5, we studied developing a "performance-driven heterogeneity measure" that can characterize the impact of the heterogeneity level of a system on its performance behavior (a.k.a. QoS) in terms of makespan. Performance evaluations across various simulated and real-world heterogeneous systems demonstrated that our proposed mathematical model can accurately characterize the performance behavior of these systems. Particularly, the results show that our proposed heterogeneity measure is able to predict the true makespan of heterogeneous systems without online evaluations with an average accuracy of 84%. This heterogeneity measure is instrumental for solution architects to configure their systems proactively to be sufficiently heterogeneous to meet their desired performance objectives. In particular, for a large heterogeneous configuration space, such as those offered by public clouds, HEET can be instrumental in configuring a system (instead of trial-and-error) with respect to the desired throughput and

without examining the workload.

7.2 Future Research Direction

Based on our findings during the exploration of natively-heterogeneous design of distributed computing systems, there are several points where the work could be expanded upon that were not covered in this dissertation.

7.2.1 Cost- and QoS-aware Heterogeneous System Configuration

We can utilize the proposed measure of heterogeneity to compare the QoS of different heterogeneous computing systems. Thus, a scaling manager can be devised to use HEET score for recommending new system configuration such that the cost is minimized with a user-defined QoS constraint. In practice, the scaling manager uses HEET score to make recommendations for adjustments to the heterogeneous computing system. These adjustments may involve altering the configuration by replacement, addition or removal of specific machines from the user-defined pool of available machine types.

7.2.2 Probabilistic Heterogeneity Measure

Note that HEET score calculation is based on the expected values of the execution time of tasks on various machines. The presence of variance within the execution time distribution can introduce uncertainty into the HEET score, potentially undermining its ability to accurately predict the QoS in a heterogeneous computing system. To mitigate this issue, an enhancement can be introduced by integrating the stochastic nature of execution times into the HEET score calculation.

In this context, we can employ a probabilistic approach wherein the elements

of the EET matrix are replaced with the Probability Mass Function (PMF) of execution times associated with a particular task type on a given machine type. Consequently, the HEET score reflects the execution time PMF of the hypothetical equivalent task type on the equivalent machine type. This adjustment not only considers the mean execution time but also encapsulates the inherent variability, resulting in a more robust heterogeneity measure.

7.2.3 Energy-aware Heterogeneity Measure

The dimensions of system heterogeneity extend beyond task and machine diversities. Another critical facet of heterogeneity pertains to variations in energy consumption among different machine types. Hence, the scope of heterogeneity dimensions can be broadened to encompass three key elements by integrating machine power heterogeneity into HEET score calculations.

By adopting this three-dimensional approach, the HEET score transforms into a two-dimensional (2D) metric. It now encapsulates not only the homogeneous equivalent execution time of a hypothetical equivalent task type on the equivalent machine type but also incorporates the corresponding expected energy consumption. This enhanced HEET score thus furnishes a comprehensive representation of system behavior. Subsequently, the 2D HEET measure can be harnessed to develop system configurations that are cost- and energy-aware, all while adhering to QoS constraints set by the user. In essence, this multi-dimensional approach offers a more holistic approach in the natively-heterogeneous design of distributed computing systems.

Bibliography

[1] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, S. Ali *et al.*, "Representing task and machine heterogeneities for heterogeneous computing systems," *Journal of Applied Science and Engineering*, vol. 3, no. 3, pp. 195–207, 2000.

[2] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *DAC Design Automation Conference 2012*, 2012, pp. 1131–1136.

[3] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th annual international symposium on Computer architecture*, 2011, pp. 365–376.

[4] "Amazon sagemaker," https://aws.amazon.com/sagemaker/.

[5] "Heterogeneous clusters for amazon sagemaker model training," https://aws.amazon.com/about-aws/whats-new/2022/07/announcing-heterogeneous-clusters-amazon-sagemaker-model-training/.

[6] "Glass enterprise editiotn 2," https://www.google.com/glass/tech-specs/.

[7] "Qualcomm reveals the world's first dedicated xr platform," https://www.qualcomm.com/news/releases/2018/05/qualcomm-reveals-worlds-first-dedicated-xr-platform.

[8] S. G. Cardwell, C. Vineyard, W. Severa, F. S. Chance, F. Rothganger, F. Wang, S. Musuvathy, C. Teeter, and J. B. Aimone, "Truly heterogeneous hpc: Co-design to achieve what science needs from hpc," in *Smoky Mountains Computational Sciences and Engineering Conference*, 2020, pp. 349–365.

[9] "Top500," https://www.top500.org/lists/top500/2022/06/.

[10] 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.

[11] 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.

[12] M. A. Salehi, J. Smith, A. A. Maciejewski, H. J. Siegel, E. K. Chong, J. Apodaca, L. D. Briceno, 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*, vol. 97, pp. 96–111, 2016.

[13] espressif, "ESP32," Accessed April 10th, 2023. [Online]. Available: https://www.espressif.com/en/products/socs/esp32

[14] himax, "Himax WE-I," Accessed April 10th, 2023. [Online]. Available: https://www.himax.com.tw/products/intelligent-sensing/always-on-smart-sensing/

[15] M. B. Taylor, L. Vega, M. Khazraee, I. Magaki, S. Davidson, and D. Richmond, "Asic clouds: Specializing the datacenter for planet-scale applications," *Communications of the ACM*, vol. 63, no. 7, pp. 103–109, 2020.

[16] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser *et al.*, "The future of fpga acceleration in datacenters and the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 3, pp. 1–42, 2022.

[17] A. Mokhtari, M. A. Hossen, P. Jamshidi, and M. A. Salehi, "Felare: Fair scheduling of machine learning tasks on heterogeneous edge systems," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, 2022, pp. 459–468.

[18] S. Zobaed, A. Mokhtari, J. P. Champati, M. Kourouma, and M. A. Salehi, "Edge-multiai: Multi-tenancy of latency-sensitive deep learning applications on edge," in *in Proceedings of 15th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '22)*, 2022.

[19] C. Denninnart, J. Gentry, A. Mokhtari, and M. A. Salehi, "Efficient task pruning mechanism to improve robustness of heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 142, pp. 46–61, 2020.

[20] J. Gentry, C. Denninnart, and M. A. Salehi, "Robust dynamic resource allocation via probabilistic task pruning in heterogeneous computing systems," in *Proceedings of IEEE international parallel and distributed processing symposium (IPDPS)*, 2019, pp. 375–384.

[21] B. Pérez, E. Stafford, J. L. Bosque, and R. Beivide, "Sigmoid: an auto-tuned load balancing algorithm for heterogeneous systems," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 30–42, 2021.

[22] Y. N. Khalid, M. Aleem, U. Ahmed, M. A. Islam, and M. A. Iqbal, "Troodon: A machine-learning based load-balancing application scheduler for cpu–gpu system," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 79–94, 2019.

[23] R. Nozal, J. L. Bosque, and R. Beivide, "Enginecl: usability and performance in heterogeneous computing," *Future Generation Computer Systems*, vol. 107, pp. 522–537, 2020.

[24] K. Gardner, J. Abdul Jaleel, A. Wickeham, and S. Doroudi, "Scalable load balancing in the presence of heterogeneous servers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 48, no. 3, pp. 37–38, 2021.

[25] J. Vandebon, J. G. Coutinho, W. Luk, E. Nurvitadhi, and M. Naik, "Enhanced heterogeneous cloud: Transparent acceleration and elasticity," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 162–170.

[26] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 2, pp. 1–24, 2020.

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

[28] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi *et al.*, "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment," *Journal of parallel and distributed computing*, vol. 67, no. 2, pp. 154–169, 2007.

[29] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 481–498.

[30] A. Mokhtari, C. Denninnart, and M. A. Salehi, "Autonomous task dropping mechanism to achieve robustness in heterogeneous computing systems," in *29th Heterogeneity in Computing Workshop (HCW 2020), in the Proceedings of the IPDPS 2020 Workshops PhD Forum (IPDPSW)*, 2020, pp. 17–26.

[31] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '18, 2018, pp. 121–134.

[32] A. Marahatta, S. Pirbhulal, F. Zhang, R. M. Parizi, K. R. Choo, and Z. Liu, "Classification-based and energy-efficient dynamic task scheduling scheme for

virtualized cloud data center," *IEEE Transactions on Cloud Computing*, May 2019.

[33] K.-B. Chen and H.-Y. Chang, "Complexity of cloud-based transcoding platform for scalable and effective video streaming services," *Multimedia Tools and Applications*, vol. 76, no. 19, pp. 19 557–19 574, Oct. 2017.

[34] S. Lin, X. Zhang, Q. Yu, H. Qi, and S. Ma, "Parallelizing video transcoding with load balancing on cloud computing," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, ser. ISCAS '13, May 2013, pp. 2864–2867.

[35] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E.-g. Talbi, "Towards understanding uncertainty in cloud computing resource provisioning," *Procedia Computer Science*, vol. 51, no. C, pp. 1772–1781, Sep. 2015.

[36] 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.

[37] A. G. Kumbhare, Y. Simmhan, M. Frincu, and V. K. Prasanna, "Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 105–118, Jan. 2015.

[38] M. A. Oxley, S. Pasricha, A. A. Maciejewski, H. J. Siegel, J. Apodaca, D. Young, L. Briceno, J. Smith, S. Bahirat, B. Khemka *et al.*, "Makespan and energy robust stochastic static resource allocation of a bag-of-tasks to a heterogeneous computing system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2791–2805, 2014.

[39] E. Araujo Macedo, A. C. Magalhaes Alves De Melo, G. H. Pfitscher, and A. Boukerche, "Multiple biological sequence alignment in heterogeneous multicore clusters with user-selectable task allocation policies," *Journal of Supercomputing*, vol. 63, no. 3, pp. 740–756, Mar. 2013.

[40] G. Aupy, A. Gainaru, V. Honoré, P. Raghavan, Y. Robert, and H. Sun, "Reservation strategies for stochastic jobs," in *Proceedings of the 33rd IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '19, May 2019.

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

[42] 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.

[43] C. Denninnart, J. Gentry, and M. Amini Salehi, "Improving robustness of heterogeneous serverless computing systems via probabilistic task pruning," in *Proceedings of the 28th Heterogeneity in Computing Workshop*, ser. HCW '19, May 2019.

[44] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[45] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 82–95.

[46] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–37, 2020.

[47] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[48] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proceedings of the 35th International Conference on Computer-Aided Design*, 2016, pp. 1–7.

[49] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.

[50] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[51] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique, "Px-cgra: Polymorphic approximate coarse-grained reconfigurable

architecture," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 413–418.

[52] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," *arXiv preprint arXiv:2004.09602*, 2020.

[53] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[54] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 675–686, 2021.

[55] C. Liu, K. Li, J. Liang, and K. Li, "Cooper-sched: A cooperative scheduling framework for mobile edge computing with expected deadline guarantee," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[56] S. Bitam, S. Zeadally, and A. Mellouk, "Fog computing job scheduling optimization based on bees swarm," *Enterprise Information Systems*, vol. 12, no. 4, pp. 373–397, 2018.

[57] R. Yadav, W. Zhang, O. Kaiwartya, P. R. Singh, I. A. Elgendy, and Y.-C. Tian, "Adaptive energy-aware algorithms for minimizing energy consumption and SLA violation in cloud computing," *IEEE Access*, vol. 6, pp. 55 923–55 936, 2018.

[58] Y. Zhang, J. He, and S. Guo, "Energy-efficient dynamic task offloading for energy harvesting mobile cloud computing," in *IEEE international conference on networking, architecture and storage (NAS)*. IEEE, 2018, pp. 1–4.

[59] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, and J. Zeng, "Q-learning based dynamic task scheduling for energy-efficient cloud computing," *Future Generation Computer Systems*, vol. 108, pp. 361–371, 2020.

[60] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft," *Future Generation Computer Systems*, vol. 93, pp. 278–289, 2019.

[61] M. Kumar and S. C. Sharma, "Pso-based novel resource scheduling technique to improve qos parameters in cloud computing," *Neural Computing and Applications*, vol. 32, no. 16, pp. 12 103–12 126, 2020.

[62] S. Ghanavati, J. H. Abawajy, and D. Izadi, "An energy aware task scheduling model using ant-mating optimization in fog computing environment," *IEEE Transactions on Services Computing*, 2020.

[63] A. Tarafdar, M. Debnath, S. Khatua, and R. K. Das, "Energy and makespan aware scheduling of deadline sensitive tasks in the cloud environment," *Journal of Grid Computing*, vol. 19, no. 2, pp. 1–25, 2021.

[64] M. Hussain, L.-F. Wei, A. Lakhan, S. Wali, S. Ali, and A. Hussain, "Energy and performance-efficient task scheduling in heterogeneous virtualized cloud computing," *Sustainable Computing: Informatics and Systems*, vol. 30, p. 100517, 2021.

[65] S. K. Panda and P. K. Jana, "An energy-efficient task scheduling algorithm for heterogeneous cloud computing systems," *Journal of Cluster Computing*, vol. 22, no. 2, pp. 509–527, 2019.

[66] Y. Chen, Y. Zhang, Y. Wu, L. Qi, X. Chen, and X. Shen, "Joint task scheduling and energy management for heterogeneous mobile edge computing with hybrid energy supply," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8419–8429, 2020.

[67] Z. Quan, Z.-J. Wang, T. Ye, and S. Guo, "Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1165–1182, 2019.

[68] S. Ghafouri, A. A. Saleh-Bigdeli, and J. Doyle, "Consolidation of services in mobile edge clouds using a learning-based framework," in *Proceedings of IEEE World Congress on Services (SERVICES)*, 2020, pp. 116–121.

[69] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient iot task scheduling in fog computing systems: A semi-greedy approach," *Journal of network and computer applications*, vol. 201, p. 103333, 2022.

[70] S. M. Hussain and G. R. Begh, "Hybrid heuristic algorithm for cost-efficient qos aware task scheduling in fog–cloud environment," *Journal of Computational Science*, vol. 64, p. 101828, 2022.

[71] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1505–1533, 2015.

[72] B. Li, R. B. Roy, T. Patel, V. Gadepally, K. Gettings, and D. Tiwari, "Ribbon: Cost-effective and qos-aware deep learning model inference using a diverse pool of cloud computing instances," in *Proceedings of the International*

Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476168

[73] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Cocktail: A multidimensional optimization for model serving in cloud," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 1041–1057. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/gunasekaran

[74] B. Li, S. Samsi, V. Gadepally, and D. Tiwari, "Kairos: Building cost-efficient machine learning inference systems with heterogeneous cloud resources," *arXiv preprint arXiv:xxxx.xxxxx*, 2022, submitted on 12 Oct 2022 (v1), last revised 2 May 2023 (this version, v3).

[75] 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, Jan. 2017.

[76] 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.

[77] I. Grasso, S. Pellegrini, B. Cosenza, and T. Fahringer, "A uniform approach for programming distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 74, no. 12, pp. 3228–3239, Dec. 2014.

[78] Amazon, "Amazon Web Sevices (AWS) Instance Types," Accessed Oct. 1st, 2019. [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[79] X. Li, M. A. Salehi, Y. Joshi, M. Darwich, M. Bayoumi, and B. Landreneau, "Performance analysis and modeling of video transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Sep. 2018.

[80] T. Hansen, F. M. Ciorba, A. A. Maciejewski, H. J. Siegel, S. Srivastava, and I. Banicescu, "Heuristics for robust allocation of resources to parallel applications with uncertain execution times in heterogeneous systems with uncertain availability," in *Proceedings of the World Congress on Engineering*, vol. 1, Jul. 2014.

[81] M. A. Oxley, S. Pasricha, H. J. Siegel, and A. A. Maciejewski, "Energy and deadline constrained robust stochastic static resource allocation," in

Proceedings of the 10th International Conference on Parallel Processing and Applied Mathematics, ser. PPAM '13, Sep. 2013, pp. 761–771.

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

[83] 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*, Aug. 2019, pp. 929–936.

[84] J. Bi, H. Yuan, W. Tan, M. Zhou, Y. Fan, J. Zhang, and J. Li, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Transactions on Automation Science and Engineering (T-ASE)*, vol. 14, no. 2, pp. 1172–1184, Apr 2017.

[85] M. Hosseini, M. A. Salehi, and R. Gottumukkala, "Enabling Interactive Video Stream Prioritization for Public Safety Monitoring through Effective Batch Scheduling," in *Proceedings of the 19th IEEE International Conference on High Performance Computing and Communications*, ser. HPCC '17, Dec. 2017, pp. 474–481.

[86] 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.

[87] ——, "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, May 2016, pp. 600–603.

[88] Standard Performance Evaluation Corporation, Accessed Oct. 1st, 2019. [Online]. Available: https://www.spec.org/benchmarks.html

[89] Louisiana Optical Network Infrastructure, "LONI Resources QB2," Accessed Oct. 1st, 2019. [Online]. Available: http://hpc.loni.org/resources/hpc/system.php?system=QB2

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

[91] M. Bhatia and S. K. Sood, "A comprehensive health assessment framework to facilitate iot-assisted smart workouts: A predictive healthcare perspective," *Computers in Industry*, vol. 92, pp. 50–66, 2017.

[92] D. G. Samani and M. Amini Salehi, "Exploring the impact of virtualization on the usability of the deep learning applications," in *Proceedings of the 22th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, ser. CCGrid '22, May 2022.

[93] J. Wen, J. Liu, F. Xu, X. Duan, and J. Huang, "Face recognition system design based on esp32," in *2022 International Seminar on Computer Science and Engineering Technology (SCSET)*, 2022, pp. 114–116.

[94] "Designware arc em9d / em11d processors," https://www.synopsys.com/dw/ipdir.php?ds=arc-em9d-em11d, accessed: 2022-05-24.

[95] N. Gholipour, E. Arianyan, and R. Buyya, "Recent advances in energy efficient resource management techniques in cloud computing environments," *arXiv preprint arXiv:2107.06005*, 2021.

[96] M. Aazam, S. Zeadally, and E. F. Flushing, "Task offloading in edge computing for machine learning-based smart healthcare," *Computer Networks*, vol. 191, p. 108019, 2021.

[97] N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Optimized provisioning of edge computing resources with heterogeneous workload in iot networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 459–474, 2019.

[98] M. A. Rahman and M. S. Sadi, "Iot enabled automated object recognition for the visually impaired," *Computer Methods and Programs in Biomedicine Update*, vol. 1, p. 100015, 2021.

[99] C. Denninnart, J. Gentry, and M. Amini Salehi, "Improving robustness of heterogeneous serverless computing systems via probabilistic task pruning," in *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2019, pp. 6–15.

[100] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[101] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[102] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE signal processing letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[103] G. B. H. E. Learned-Miller, "Labeled faces in the wild: Updates and new reporting procedures," University of Massachusetts, Amherst, Tech. Rep. UM-CS-2014-003, May 2014.

[104] S. P. Praveen, K. T. Rao, and B. Janakiramaiah, "Effective allocation of resources and task scheduling in cloud environment using social group optimization," *Arabian Journal for Science and Engineering*, vol. 43, no. 8, pp. 4265–4272, 2018.

[105] Y. Shi, Z. Chen, W. Quan, and M. Wen, "A performance study of static task scheduling heuristics on cloud-scale acceleration architecture," in *Proceedings of the 5th International Conference on Computing and Data Engineering*, 2019, pp. 81–85.

[106] S. Allmeier and N. Gast, "Mean field and refined mean field approximations for heterogeneous systems: It works!" *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 1, pp. 1–43, 2022.

[107] L. Ying, "On the approximation error of mean-field models," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1, pp. 285–297, 2016.

[108] D. J. Lilja, *Measuring computer performance: a practitioner's guide*. Cambridge university press, 2005.

[109] L. K. John, "More on finding a single number to indicate overall performance of a benchmark suite," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 1, pp. 3–8, 2004.

[110] J. E. Smith, "Characterizing computer performance with a single number," *Communications of the ACM*, vol. 31, no. 10, pp. 1202–1206, 1988.

[111] Salesforce, "Real-world examples of machine learning," June 2020, accessed: August 2, 2023. [Online]. Available: https://www.salesforce.com/eu/blog/2020/06/real-world-examples-of-machine-learning.html

[112] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[113] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[114] G. Jocher, "YOLOv5 by Ultralytics," May 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[115] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[116] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, 2020.

[117] O. D. Team, "Onnx github repository," accessed: August 2, 2023. [Online]. Available: https://github.com/onnx/onnx

[118] Microsoft, "Convert a pytorch model to onnx for windows machine learning," 2023, accessed: August 2, 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/ai/windows-ml/tutorials/pytorch-convert-model

[119] Seldon, "What is multi-model serving and how does it transform your ml infrastructure?" 2023, accessed: August 2, 2023. [Online]. Available: https://www.seldon.io/what-is-multi-model-serving-and-how-does-it-transform-your-ml-infrastructure

[120] N. Corporation, "Triton inference server github repository," accessed: August 2, 2023. [Online]. Available: https://github.com/triton-inference-server/server

[121] "GRPC." [Online]. Available: https://grpc.io/

[122] M. Techlabs, "Comparison between grpc vs. rest," accessed: August 2, 2023. [Online]. Available: https://marutitech.com/rest-vs-grpc/#Comparison_Between_gRPC_vs_REST

[123] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[124] "prometheus." [Online]. Available: https://prometheus.io/

[125] A. W. Services, "Aws sdk for python," accessed: August 2, 2023. [Online]. Available: https://aws.amazon.com/sdk-for-python/

[126] C. Denninnart, J. Gentry, A. Mokhtari, and M. Amini Salehi, "Efficient task pruning mechanism to improve robustness of heterogeneous computing systems," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 142, pp. 46–61, 2020.

[127] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[128] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3493, 2018.

[129] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[130] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "icancloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, pp. 185–209, 2012.

[131] Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, and M. N. Alsaleh, "Teachcloud: a cloud computing educational toolkit," *International Journal of Cloud Computing 1*, vol. 2, no. 2-3, pp. 237–257, 2013.

[132] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of parallel and distributed computing*, vol. 59, no. 2, pp. 107–131, 1999.

[133] A. Mokhtari, M. A. Hossen, P. Jamshidi, and M. Amini Salehi, "FELARE: Fair Scheduling of Machine Learning Applications on Heterogeneous Edge Systems," in *Proceedings of the 15th IEEE International Conference on Cloud Computing*, ser. IEEE Cloud '22, 2022.

[134] S. Zobaed, A. Mokhtari, J. P. Champati, M. Kourouma, and M. Amini Salehi, "Edge-MultiAI: Multi-Tenancy of Latency-Sensitive Deep Learning Applications on Edge," in *Proceedings of 15th IEEE/ACM International Conference on Utility and Cloud Computing*, ser. UCC '22, Dec. 2022.

Biographical Sketch

Ali Mokhtari was born in Shiraz, Iran on July 30, 1988. In 2010, he successfully obtained a Bachelor of Science degree in mechanical engineering from Shiraz University. Building on this foundation, Ali pursued higher education and achieved a master's degree in aerospace engineering from Sharif University of Technology in 2016. In 2019, Ali further expanded his expertise by enrolling in the Ph.D. program in computer science at UL Lafayette. Throughout his doctoral studies, his research focus revolved around heterogeneous computing systems, showcasing his dedication to advancing the understanding of heterogeneity in distributed computing systems as cutting-edge technologies. Ali's academic journey reached a significant milestone in the Fall of 2023 when he proudly graduated with a Ph.D. in computer science from UL Lafayette, solidifying his position as a skilled and accomplished professional in the field.