Dynamicity-Aware Edge Computing: Predicting Worker Availability and Utilizing Reputation Scores

by

Maria Kantardjian

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
August 2023

Dedication

to my sister, Hera

Abstract

Democratizing the edge by harnessing the underutilized computational resources of end devices, also referred to as Extreme Edge Devices (EEDs), can foster a broad spectrum of data-intensive and/or delay-sensitive applications. However, EEDs are user-owned devices characterized by a highly dynamic nature. In this thesis, we propose a framework that accounts for such dynamicity and mitigates the associated risks. In particular, we address the risk of intermittent availability of EEDs and the risk of continuous changes in their available capabilities that can lead to incongruities between their perceived and actual performance, which can profoundly impact the Quality of Service (QoS). To resolve the intermittent availability issue, we propose the Dynamic Worker Availability Prediction (DWAP) scheme. DWAP predicts the availability of EEDs (i.e., workers) and adapts to the highly dynamic nature of the computing environment at the extreme edge. DWAP employs the Continuous-Time Markov Chain (CTMC) model to forecast the availability of workers in the upcoming time-step. It does so while continuously fine-tuning the model parameters to incorporate newly available data. Towards that end, we use a dataset that consists of real-world Google cluster workload traces. In addition, we propose the DWAP-Reputation Enhanced (DWAP-RE) scheme to account for the reliability issues triggered by possible discrepancies between the perceived and actual capabilities of workers. DWAP-RE is the

first scheme that uses a comprehensive reputation scoring system to assess the reliability of workers based on past performance. It then makes reliability-aware resource allocation decisions by incorporating the workers' reputation scores into the decision-making process. Extensive evaluations show that DWAP significantly outperforms a representative of state-of-the-art prediction schemes by up to 74% and 59% in terms of the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), respectively. Additionally, DWAP yields 97% and 48% reduction in task drop rate compared to prominent availability-oblivious and availability-aware resource allocation schemes, respectively. Moreover, DWAP-RE outperforms prominent reliability-oblivious and reliability-aware resource allocation schemes by up to 43% and 16%, and 42% and 22% in terms of execution time and satisfaction ratio, respectively. Furthermore, it outperforms prominent reliability-oblivious resource allocation schemes by 97% in terms of task drop rate.

Co-Authorship

Journal Articles

- Maria Kantardjian, Sara A. Elsayed, and Hossam S. Hassanein, "Dynamicity-Aware Extreme Edge Computing," 2023, (Journal paper in preparation)

Conference Publications

- Maria Kantardjian, Sara A. Elsayed, and Hossam S. Hassanein, "Dynamic Worker Availability Prediction at the Extreme Edge," 2023 IEEE Global Communications Conference (GLOBECOM), Kuala Lumpur, Malaysia, 2023, (Accepted)

Acknowledgments

First and foremost, I would like to extend my heartfelt and deepest appreciation to Prof. Hossam Hassanien. Joining Queen's and TRL not only marked a significant academic milestone, but a transformative period in my life. His continuous support, both academically and personally, has been a cornerstone throughout the course of my MSc journey.

I would also like to thank Dr. Sara Elsayed wholeheartedly. From the beginning to the end of this journey, amidst all challenges and triumphs, she has been an unwavering source of support. I am deeply grateful for her commitment, guidance, and the expertise she brought to this project.

I thank my family for their support and encouragement. To my sister, Hera, words fall short to express my gratitude. You have always been my anchor of strength and the person I look up to the most. This journey would have been infinitely more challenging without your enduring support, love, and belief in me. I am eternally indebted for all you have done and continue to do for me. I would not be who I am without you.

I am incredibly grateful for the new friends I have made, as well as those cherished ones who have walked beside me every step of the way. Thank you for making Kingston feel like home, and for profoundly supporting me during my most challenging

times.

Finally, to my colleagues and friends at Distributive: working alongside each of you has been an enriching experience. I am beyond grateful to each and every one of you.

Statement of Originality

I hereby certify that all the work presented in this thesis is original, and all notions and/or techniques attributed to others have been fully acknowledged in accordance with the proper referencing practices.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AURA	Availability-Unaware Resource Allocation
CC	Cloud Computing
CTMC	Continous-Time Markov Chains
DER	Deep Evidential Regression
DWAP-RE	DWAP-Reputation Enhanced
EC	Edge Computing
EED	Extreme Edge Devices
EMA	Exponential Moving Average
EMARA	Exponential Moving Average Resource Allocation
ETSI	European Telecommunications Standards Institute
FC	Fog Computing
ILP	Integer Linear Program
IoT	Internet of Things
LSTM	Long-Short-Term Memory
MAE	Mean Absolute Error

MEC	Multi-access Edge Computing
ML	Machine Learning
MTBF	Mean Time Between Failure
MTTR	Mean Time to Repair
OoD	Out-of-Distribution
QoS	Quality of Service
RMSE	Root Mean Squared Error
SQL	Structured Query Language
UQ	Uncertainty Quantification
WAN	Wide Area Network
WSM	Weighted Sum Method

List of Symbols

t_i	Task i
w_j	Worker j
x_{ij}	Decision variable for allocating task t_i to worker w_j
$c_j^{\max}$	Maximum CPU cycle frequency of worker w_j
β	Maximum number of tasks workers can run concurrently
c_{ij}	CPU cycle frequency dedicated by worker w_j to execute task t_i
γ_{ij}	Execution time of task t_i on worker w_j
α_j	Estimated availability duration of worker w_j
q_i	Workload intensity of task t_i
λ	Failure rate of a worker
μ	Repair rate of a worker
Q	State transition rate matrix for the markov chain
$P_{i,j}(t)$	Transition probability from state i to state j within time t

k_r	Criterion r
d_i	Deadline of task t_i
v_{rj}	Value of worker j for criterion r
v'_{rj}	Normalized value of worker j for criterion r
s_j	Reputation score for worker j
$\hat{c}_j^{\max}$	Perceived maximum CPU cycle frequency of worker w_j
$\hat{c}_{ij}$	Perceived CPU cycle frequency dedicated by worker w_j to execute task t_i
$\hat{\gamma}_{ij}$	Perceived execution time of task t_i on worker w_j

Chapter 1

Introduction

1.1 Overview and Motivation

The widespread adoption of the Internet of Things (IoT) is projected to cause a significant escalation in the total number of connected devices. A Gartner report estimates that there will be 20 billion IoT devices by 2025 [1]. Furthermore, according to the International Data Corporation, these devices will be a primary source of the 175 ZB of global data generated by 2025 [2]. This substantial increase is anticipated to trigger intensive demands on computational resources to meet the rigorous Quality of Service (QoS) requirements of delay-critical and/or data-intensive IoT applications. These applications include smart cities, Tactile Internet, autonomous vehicles, and virtual and augmented reality [3]. Satisfying the requirements of such applications cannot be achieved using Cloud Computing (CC). This is since CC involves transmitting large amounts of data to remote data centers, which can increase latency and traffic congestion at backhaul links [4].

Edge Computing (EC) has emerged as an auspicious paradigm that can mitigate the aforementioned issues. In EC, data processing is performed at the network edge,

closer to where the data is generated, drastically reducing communication latency. However, to effectively manage the offloaded computational tasks, most existing EC platforms rely on edge servers that are equipped with robust computational capabilities. These specialized servers are typically owned and managed exclusively by cloud service providers and/or network operators [5]. Such monopoly can be evaded by leveraging the underutilized computational resources of Extreme Edge Devices (EEDs), such as smartphones, connected vehicles, and PCs. This equips more players to develop and manage their own edge cloud, creating a new tech market for businesses, enterprises, and even municipalities, enabling them to act as edge service providers themselves and/or monetize their computing resources [3, 4, 6]. Furthermore, enabling parallel processing at multiple EEDs can amplify the computational power and bring the computing service in much closer proximity to end users, significantly curtailing the delay. Thus, the transformative initiative of using EED-enabled computing to democratize the edge and establish an edge computing market that is accessible and rewarding to all can play a pivotal role in realizing smart cities and IoT solutions [7, 8].

The positive impact of EED-enabled computing can be hindered by the fact that EEDs are heterogeneous and user-owned devices, which subjects them to a high dynamicity. This can lead to intermittent availability and reliability issues, triggering a significant increase in drop rate and latency. Such intermittent availability and reliability issues result from various factors, such as battery exhaustion, network disruption, and dynamic user access behavior. The latter pertains to the notion that at any given time, users may run computationally intensive applications on their devices, such as streaming videos or playing videogames. In such instances, an EED

may become unavailable and refrain from performing an offloaded task to preserve its own resources for its own convenience. Even if an EED continues executing the offloaded task, the dynamic behavior it displays can lead to discrepancies in the capabilities of the devices as perceived by the scheduler. Such discrepancies can also occur due to dynamic computational load triggered by offloaded tasks. As a result of these discrepancies, the scheduler may rely on overestimated capabilities of the EEDs, potentially assigning more intensive tasks than the devices can proficiently manage, which can profoundly impact the QoS and overall system performance. Consequently, it is imperative to account for the reliability issues involving such discrepancies, as well as the intermittent availability of EEDs, and consider their impact on resource allocation decisions.

The intermittent availability of EEDs (i.e., workers) tends to be overlooked in the literature. Some schemes rely on task replication [7] or task migration [9] to ensure service continuity and mitigate the adverse impacts of intermittent availability and unreliability of workers in EED-enhanced EC. However, task replication can lead to poor resource utilization and increased waste of resources [10], whereas task migration tends to address workers' unavailability after a failure already manifests, and can trigger increased latency and energy consumption [8]. There are also a limited number of studies that address the intermittent availability issue by employing probabilistic techniques [11, 12]. These techniques use static probabilistic methods to predict resource availability by fitting traces of workers to a statistical distribution and calculating the corresponding statistical parameters for estimating future availability [11, 12]. However, such static availability estimation schemes fail to account for the high dynamicity of EEDs, which can impact the prediction accuracy.

The high dynamicity of EEDs not only makes predicting the availability of workers quite challenging, but also makes it crucial to account for the reliability issues stemming from potential discrepancies between their perceived and actual capabilities. Such discrepancies have not been taken into consideration in the literature. Some schemes account for the reliability of workers by relying on the ratio of tasks a worker has successfully completed [13–16]. Few schemes further focus on the number of high-priority tasks completed [11, 17]. However, these schemes overlook the fact that it is not merely the successful execution of tasks that is essential, but their timely execution as well, particularly in time-critical tasks.

1.2 Objectives and Contributions

Our objectives can be summarized as follows:

1. Estimating the availability of workers in a way that accurately captures the high dynamicity of EEDs.

2. Studying the impact of worker intermittent availability-awareness on resource allocation decisions.

3. Accounting for the reliability issues associated with possible discrepancies that may occur due to the dynamic nature of EEDs, particularly between the computational capabilities that the scheduler perceives the workers to have and their actual capabilities.

4. Making reliability-aware resource allocation decisions that strive to enhance the QoS by improving the task execution time, task drop rate, and satisfaction ratio (i.e., ratio of tasks executed within their deadline).

To achieve the objectives above, we introduce:

1. *Dynamic Worker Availability Prediction (DWAP)*: DWAP addresses Objectives 1 and 2. DWAP predicts the availability of EEDs by employing a dynamic approach that incorporates newly available data to ensure that the forecasting is based on the most current and relevant information, making the system adaptive and responsive to sudden changes. In particular, DWAP incorporates the Continuous-Time Markov Chain (CTMC) model to predict the availability of workers and adapt to the highly dynamic changes in the environment. CTMC is a mathematical model that describes stochastic processes based on transition rates, where the system transitions between states over time [18]. In DWAP, we calculate the transition rates using the Mean Time Between Failure (MTBF) and the Mean Time to Repair (MTTR), which represent the worker's expected lifespan before experiencing a failure and the repair time after failure, respectively. To accurately capture the dynamic changes in the environment, we re-calculate MTBF and MTTR as new data becomes available using a dataset from Google's real-world application usage traces [19]. We then incorporate the rendered availability predictions into the resource allocation decision to study the impact of accounting for the availability of workers. To the best of our knowledge, DWAP is the first scheme that accounts for the dynamic availability of workers by fostering dynamic availability prediction to allow for better resource allocation decisions that can reduce the task drop rate.

2. *DWAP-Reputation Enhanced (DWAP-RE)*: DWAP-RE addresses Objectives 3 and 4. DWAP-RE accounts for the reliability issues of workers and considers the discrepancies between their perceived and actual computational capabilities by

incorporating a comprehensive reputation scoring system. DWAP-RE assesses and scores the reputation of workers based on their past performance in order to make informed resource allocation decisions that account for their reliability. It operates on the premise that workers with bad or inconsistent performance will inevitably experience reduced credibility, which is bound to impact their future task assignments. In DWAP-RE, the reputation score of each worker depends on four key criteria; the average execution time of the tasks it executed in the previous time steps, the average workload of these tasks, the satisfaction ratio, and the task success rate. By incorporating these criteria, the reputation score offers a comprehensive reflection of the worker's reliability and true performance potentials. Following each time step, the performance measurements of each criterion for each worker are updated, ensuring they reflect the most recent data available. To the best of our knowledge, DWAP-RE is the first scheme that takes into account the potential differences between the perceived and actual capabilities of workers, and uses an extensive reputation scoring system to allow for better resource allocation decisions that improve the task execution time, task drop rate, and satisfaction ratio.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 provides a literature review of edge computing paradigms, as well as different approaches that address the intermittent availability of workers. Additionally, it explores various schemes that consider the reliability of workers. Chapter 3 presents DWAP, the underlying system model, prediction methodology, problem formulation, as well as the performance

evaluation compared to two other prominent schemes. Chapter 4 introduces DWAP-RE and the underlying reputation scoring mechanism, and evaluates its performance compared to DWAP and a baseline reputation-based scheme. Finally, Chapter 5 concludes our work and outlines future research directions.

Chapter 2

Background and Literature Review

In this chapter, we discuss Edge Computing (EC), emphasizing the favorable attributes of EC in comparison to Cloud Computing (CC). We present several prominent infrastructure-based EC paradigms, including Cloudlets, Fog Computing (FC), and Multi-access Edge Computing (MEC). Subsequently, we introduce the concept of edge democratization, which is achieved by leveraging the underutilized computational resources of Extreme Edge Devices (EEDs), elucidating the advantages of EED-enabled EC over infrastructure-based EC. Furthermore, we highlight the key challenges associated with EED-enabled EC, namely intermittent availability and unreliability. Additionally, we delve into existing strategies from the literature that are designed to mitigate these obstacles.

2.1 Edge Computing

The National Institute of Standards and Technology (NIST) defines CC as a computing paradigm whose aim is to facilitate the widespread, ubiquitous and on-demand network access to shared computing resources [20]. CC has significantly expanded

computing, storage, and network capabilities through its large pools of easily accessible virtualized resources that can be dynamically reconfigured to handle varying workloads [21]. However, with the proliferation of IoT devices and the volume of data they generate, it becomes inefficient and impractical to transfer all the data from these devices to remote data centers fostered by CC. Furthermore, the emergence of time-critical and latency-sensitive applications raises concerns about the cloud's ability to meet ultra-low latency requirements and provide location-aware services [22].

In response to the limitations and challenges imposed by CC, a paradigm shift has emerged to address the unique requirements of the modern computing landscape. Edge Computing (EC) has risen as a complementary approach, strategically placing computation and data processing closer to the data source, thereby reducing the inefficiencies associated with data transfer and latency [21]. Unlike CC, EC leverages distributed edge nodes, such as edge servers and local infrastructure, to process data in proximity to where it is generated. This proximity enhances real-time decision-making capabilities, offers significant bandwidth savings, and alleviates the strain on network resources, especially in scenarios with massive data volumes and stringent latency demands [23]. By bridging the gap between data generation and processing, EC introduces a more responsive and agile ecosystem that aligns seamlessly with the demands of IoT applications and time-sensitive services.

In the next sections, we delve deeper into the realm of Edge Computing by exploring various infrastructure-based EC paradigms that have gained prominence in recent years, as well as EED-Enabled EC.

2.1.1 Infrastructure-based Edge Computing

Fog Computing (FC), Cloudlets, and Multi-access Edge Computing (MEC) are prominent infrastructure-based EC paradigms, each offering distinct architectural and operational advantages in harnessing the potential of edge resources. FC was introduced by Cisco [24] as a solution to address the challenges posed by high-bandwidth, low-latency, and privacy-sensitive applications. FC involves placing fog nodes in close proximity to IoT devices, leading to significantly reduced latency compared to traditional cloud-based approaches [21]. Compared to centralized cloud data centers, fog nodes are distributed in less centralized locations, allowing for widespread geographic availability. Additionally, security in fog computing is localized at the edge or specific locations of fog nodes, as opposed to centralized security mechanisms of dedicated buildings in cloud data centers [21]. Moreover, the decentralized nature of FC allows devices to serve as fog computing nodes themselves or utilize fog resources as clients. Another key distinction between CC and FC is the scale of hardware components. In contrast to CC's reliance on large data centers, FC leverages a diverse range of smaller devices, such as servers, routers, switches, gateways, and access points [25]. Thus, unlike CC, where resources are concentrated in large data centers, fog nodes are widely distributed in large numbers across different geographical locations. This distribution enables FC to provide lower power consumption compared to CC, but at the trade-off of lower availability [25]. The reduced hardware size in fog computing allows it to be located closer to end-users, and fog-based services can be accessed from the edge of the network to the core, without continuous reliance on Internet connectivity. In contrast, CC requires constant connection during the service delivery [21].

Another infrastructure-based EC paradigm was introduced by the European Telecommunications Standards Institute (ETSI), known as Mobile Edge Computing, providing mobile users with the ability to access computing services directly from base stations [23]. ETSI defined Mobile Edge Computing as a platform that provides CC capabilities within Radio Access Network (RAN) in 4G and 5G, ensuring close proximity to mobile subscribers [26]. The Mobile Edge Computing paradigm underwent expansion to encompass a broader range of network types and applications that extend beyond the confines of mobile networks. This evolution has led to its current nomenclature, known as the Multi-access Edge Computing (MEC) paradigm [21]. Examples of MEC applications include connected vehicles, health monitoring, and augmented reality. The primary objective of MEC is to extend EC capabilities by providing compute resources near resource and energy constrained mobile devices [26]. As a result, RAN operators can integrate EC features into existing base stations. MEC operates at the edge of the Internet and can operate effectively with limited Internet connectivity. MEC establishes connectivity through a Wide Area Network (WAN), WiFi, or cellular connection [21]. Moreover, MEC allows EC to be accessible to a wide range of mobile devices with reduced latency and more efficient mobile core networks [27]. MEC also facilitates the deployment of time-critical and delay-sensitive applications over the mobile network [28].

A third example of an infrastructure-based EC paradigm is Cloudlets, which was proposed by Satyanarayan et al. [29]. Cloudlets are miniature versions of cloud data centers typically located just one network hop away from mobile devices. They are resource-rich nodes with Internet connectivity. The concept revolves around transferring computational tasks from mobile devices to these virtual machine-based cloudlets

at the edge of the network [30], bringing the remote cloud services closer to end users. The concept of cloudlets was introduced as an intermediary layer within a 3-tier continuum; the end devices, an edge cloud platform (i.e., cloudlet), and a centralized datacenter (i.e., cloud). The edge cloud node could reside in community places such as shopping malls and highly populated areas such as train stations [27]. Given that cloudlets are small, local clouds in close proximity to mobile devices, they could be operated by cloud service providers aiming to bring their services closer to mobile users. Network infrastructure owners, such as AT&T and Nokia could equip cloudlets with virtualization capabilities and place them in close proximity to mobile devices, while occupying less hardware space than the data centers associated with traditional CC [31]. This results in lower latency and energy consumption than CC. Cloudlet computing is designed to service devices within a localized area and requires infrastructure equipped with virtualization capabilities, specifically the ability to operate virtual machines [21].

2.1.2 EED-enabled Edge Computing

Recently, there has been a growing body of literature that focuses on shifting from infrastructure-based EC to leverage the computational resources of end devices, including user-owned smartphones, laptops, tablets, smart appliances, connected vehicles, as well as computing devices of various public, private, and educational institutions (e.g., servers, workstations, and computer labs during low/non-working hours). The concept of EED-enabled EC capitalizes on the underutilized and idle computational resources of these devices to carry out computational tasks [3, 4, 8, 32]. This shift is primarily driven by the current state of EC platforms and models which are

typically controlled by cloud service providers and/or network operators [32]. Challenging this monopoly by repurposing the abundant yet underutilized computational resources of EEDs can lead to a democratized edge, creating a new market where more entities manage their own edge cloud. Additionally, parallel processing of tasks on computing groups of EEDs can lead to a substantial increase in the harvested computational power, and can also bring the computing service significantly closer to end-users, further reducing delay [3]. This approach offers the advantage of optimizing performance without incurring the extra cost of acquiring or upgrading infrastructure, which can be capital-intensive. Nonetheless, EED-enabled EC presents a new set of challenges. Due to their inherent dynamicity and intermittent availability, EEDs may not offer the same level of reliability as infrastructure-based EC [3, 7].

Other existing frameworks that share a conceptual foundation with EED-enabled EC include Volunteer Computing [33] and Mobile Crowdsensing [34]. All three frameworks leverage individual devices to attain a collective goal, marking a cooperative approach to problem-solving. However, unlike in EED-enabled EC, Volunteer Computing often leans towards scientific endeavors, harnessing unused computational resources of volunteers' devices for research projects, typically without offering incentives or rewards for the contributors [9, 33, 35]. On the other hand, Mobile Crowdsensing transforms smartphones and other smart devices into dynamic sensors that actively gather real-time data from their surroundings. This data is then transmitted to the cloud for detailed analysis, management, and storage. In contrast to EED-enabled EC, which often operates passively in the background, mobile crowdsensing is a more active and engaged process, relying on the constant input from a network of devices and their sensors [34, 36].

2.2 Intermittent Availability Management

Several methods have been proposed to mitigate the adverse impacts of intermittent availability of workers in EED-enhanced EC. These methods can be broadly classified into two categories: *Reactive* and *Proactive*. Reactive approaches refer to measures taken after a failure occurs in order to minimize the negative impact on offloaded tasks. Proactive approaches, on the other hand, refer to measures taken to reduce the likelihood of failure before its occurrence.

2.2.1 Reactive Approaches

Common reactive methods for recovering from failures include task checkpointing [10], task replication [7], and task migration [9]. Checkpointing is used to regularly save the task's state [10]. In the event of failure, the task resumes from the last saved state. However, determining the optimal frequency of checkpointing is a challenging issue in such schemes. Creating checkpoints too frequently can lead to a waste of time and resources, whereas creating infrequent checkpoints can lead to significant data loss [10]. Typically, checkpoint creation in distributed systems involves one of three methods [37]: coordinated checkpointing [38, 39], independent checkpointing [40] and communication-induced checkpointing [41]. In coordinated checkpointing, all processes in the system coordinate to create a consistent global snapshot of the system state at the same time. While this consistent set of checkpoints provides a reliable recovery point in case of system failures, the synchronization among all processes introduces additional overhead [38, 39]. On the other hand, in independent checkpointing, each process saves its state independently of others. This is more efficient in terms of resource utilization and could reduce the time required for checkpoint

creation, since processes do not need to be synchronized [40]. However, it may lead to inconsistencies between different checkpoints. This inconsistency could make the restoration more difficult as it may be unclear which state the system should revert to [40]. In communication-induced checkpointing, checkpoints are created based on communication between processes, rather than following a fixed schedule as in coordinated checkpointing, or independently as in independent checkpointing [41]. This approach reduces synchronization overhead and mitigates potential state inconsistencies [41].

An alternative approach to checkpointing that tackles the problem of worker intermittent availability is task replication. In [7], Amer et al. allocate multiple replicas of each task to multiple workers in order to increase the chance of continuing the task without interruption in the event of failure at one worker. However, task replication can lead to poor resource utilization and increased waste of resources [10]. Determining the optimal number of replicas is a difficult challenge. It is based on the resource usage patterns, criticality of the task in terms of timely completion, and the workload on the system [42]. Most existing research works propose the use of a single task replica or determine a predefined fixed number of replicas [10, 42]. In [43], Sun et al. propose a task replication scheme to minimize the average task offloading delay. A learning-based task replication algorithm based on a combinatorial multi-armed bandit is used. However, the number of replicas is fixed. This causes a significant degradation in performance since multiple replicas lead to poor resource utilization and higher overall energy consumption [42]. In [42], McGough et al. propose a Reinforcement Learning approach to learn the best number of replicas, which is based on the time of day and the criticality of the task. This approach is less likely to suffer

from poor performance due to sub-optimal replica count selection in comparison to approaches that use a fixed number. However, the risk of increased wasted resources still manifests.

Rather than replicating the tasks, some schemes resort to task migration to alleviate the impact of worker intermittent availability. When a worker experiences a failure, its allocated tasks could be migrated to another device to resume the service [33]. In [9], Saleh et al. use service migration in peer-to-peer networks to transfer the remaining portion of the task from an unreliable worker to a different worker within the network. Two task migration approaches are proposed [9]. The first approach migrates tasks to the first available worker, while the second approach waits for all sub-peers to finish their tasks before initiating migrations. The second approach is shown to be more reliable than the first approach. However, task migration can introduce additional overhead, latency, and energy consumption due to the transmission of the task from one worker to another.

In general, reactive approaches can often result in wasted resources, lost time, and decreased satisfaction for end-users due to increased delays [8]. Proactive approaches tend to render better performance results than reactive approaches [3].

2.2.2 Proactive Approaches

Several proactive techniques have been proposed to enable better decision-making concerning resource allocation to improve QoS for end users. These methods are often based on making predictions regarding resource availability. Some use statistical analysis [12, 44] while others use machine learning (ML) [35] to make these predictions. Some other schemes also explore stochastic modeling techniques [18].

In [12] and [44], Javadi et al. and Nurmi et al. propose availability prediction methods that involve fitting availability traces to a statistical distibution, such as Gamma or Weibull. Subsequently, the derived parameters from these fitted models are utilized to predict future resource availability. However, these methods are static and may not account for the dynamic characteristics of the environment, such as changes in the resource usage behavior of users. This results in a high number of incorrect predictions, which may negatively impact resource allocation decisions.

In [35], McGough et al. utilize ML algorithms to forecast the idle periods of workers and allocate the tasks accordingly. They train the algorithms using several months' worth of traces and predict the idle times for the following month. However, a notable limitation of this approach is its lack of adaptability to dynamic environments. Making predictions a month in advance could result in inaccurate predictions due to ongoing changes in the environment. In [45], Ramachandran et al. propose using unsupervised ML to perform clustering analysis. They leverage historical data of the worker's resources to identify patterns that aid in predicting future availability, employing Hierarchical clustering algorithms. ML techniques have also been used to forecast resource usage. In [46], Violos et. al use a Long Short-Term Memory and Convolutional Neural Network to predict the resource usage of workers. In the prediction methodology, a combination of Bayesian and particle swarm optimization is used for hyper-parameter search. The proposed method outperforms other ML models, such as support vector regression, multiple linear regression, and XGBoost. While ML techniques are robust predictive tools, they are computationally intensive and suffer from long training times [3]. Additionally, as highlighted in [47], their adaptability to rapidly and consistently shifting patterns is limited, reducing their

efficacy in dynamic environments. These factors, combined with the fact that most proposed ML-based models rely on a static training process, their practical application in a dynamic setting can be limited. In [48], Deng et al. propose a predictive task allocation mechanism by using the Exponential Moving Average (EMA). EMA is a statistical method used in time series analysis. It calculates the weighted average of past availability observations, with more weight given to recent observations. However, EMA may fail to capture sudden shifts. In [3], Kain et. al propose the Resource Usage Multi-step Prediction (RUMP) scheme, which enables multi-step ahead prediction of the resource usage of workers. RUMP uses the Hierarchical Dirichlet Process-Hidden Semi Markov Model(HDP-HSMM). However, RUMP does not predict the workers' availability, it predicts their resource usage.

In [18], Umer et al. develop a Markov chain-based model to predict worker states up to 15 days ahead. However, this scheme also fails to account for the dynamicity of EED-enabled environments. This is since it relies on a static training process, which involves training the model on a fixed dataset and making predictions for days in advance. Additionally, the set of workers considered are fairly reliable, without much dynamic activity. As a result, it remains unclear how well the model would perform in environments with more frequent and significant changes in worker behavior.

Based on the aforementioned discussion, it can be noted that existing proactive approaches are not entirely applicable to dynamic environments. They either fail to predict the availability of workers or make predictions that rely on static training models that analyze historical data and provide estimations over days or even months. These models may provide estimates spanning days or even months. However, in a dynamic environment where conditions can change rapidly, these long-term

estimations can become unreliable.

2.3 Reliability-Aware Techniques

Numerous schemes have been proposed to enhance the reliability of dynamic EC environments. Some of the proposed approaches consider the resources and capabilities of workers, acknowledging these elements as fundamental indicators of their dependability and reliability [49]. Other methods analyze the historical performance data of workers on previous task executions [11, 14, 17]. These performance-based approaches focus on past outcomes as a reliable indicator for future task executions.

The majority of reliability-aware schemes that adopt a performance-based approach tend to rely solely on the task success rate (i.e., ratio of the number of successfully completed tasks to the total number of tasks) as a measure of the worker's performance reliability [14]. However, the worker's resource requirements and performance sensitivity across various applications call for a more intricate approach [11, 17]. This approach should consider not only the success rate of tasks but also their specific attributes and the actual workload of workers.

In [17], Alsenani et. al propose a scheme for assessing worker reliability based on their performance in past task executions. The parameters considered for the reliability score include the number of tasks allocated to the worker, the number of tasks it fails to complete, and its CPU and memory capabilities. These parameters are then fed into an Artificial Neural Network, which predicts potential failures and resource utilization. The worker's reliability score is determined from the network's output, namely the failure rate and resource utilization ratio. However, a significant oversight in this work is that it fails to consider the size of the tasks allocated

to the workers. Workers handling larger tasks successfully should be deemed more reliable. In addition, it neglects the time-sensitive nature of tasks, which is an essential aspect to consider when assessing reliability. It is not merely completing the tasks that is important, but also ensuring that they are completed within acceptable time constraints. Moreover, the dynamic fluctuations in the worker's capabilities are completely overlooked.

In [11], another reliability-aware scheme is proposed. This scheme assesses the reliability of workers based on the likelihood of a worker successfully executing a task, utilizing a Beta reputation system. This system applies the standard Beta distribution, a two-parameter distribution commonly used across numerous reliability engineering applications. To determine the worker's reliability, the Beta reputation system takes into account two factors: the number of tasks a worker has successfully completed, denoted α, and the number of tasks the worker has failed to complete, denoted β. However, in the proposed scheme, α and β are not merely the counts of successes and failures in terms of task executions. Instead, α is calculated using a weighted sum that takes into consideration the varying priority levels of successful tasks. Similarly, β is calculated using a weighted sum that takes into account the varying priority levels for unsuccessful tasks. The final worker reliability is calculated as the expected value of a random variable following the Beta distribution. The resulting value is in the range [0,1], where a score of 0.5 is considered a neutral rating. Although this scheme considers the priority of executed tasks as an element influencing the worker's reliability score, it overlooks the crucial aspect of time-criticality. Moreover, the proposed method is static and therefore not well-suited for dynamic environments.

Another reliability-aware approach involves assigning a higher reliability score to workers who contribute more resources, thereby equating reliability with effort [33]. In [49], Vega et. al introduce an effort-based scheme for resource sharing in collaborative applications. In this scheme, the reliability of the worker is evaluated based on the worker's resource contributions, rather than the absolute value of their resources. Specifically, it considers the proportion of a worker's resource contribution relative to its total resource capacity. This effort-based approach aims to create fairness by favoring workers with limited resource capacity that still contribute a significant portion of their resources.

Based on the aforementioned discussion, it can be noted that existing reliability assessment methods are inadequate for highly dynamic EC environments. In particular, these schemes fail to account for the dynamic fluctuations in workers' capabilities and the ensuing discrepancies between their perceived and actual performance. While existing approaches concentrate on metrics like task success rate, the proportion of the worker's resources contributed to tasks, CPU and memory capabilities, or the execution of high-priority tasks, they often fall short in capturing the uncertainties surrounding the workers' actual capabilities and how these uncertainties influence task execution speed and efficiency. Furthermore, these methods neglect to consider the impact of task characteristics, such as workload intensity, on the worker's performance, and they fail to account for the time-critical nature of certain tasks. This deficiency hampers their ability to accurately assess reliability in scenarios where tasks vary in intensity and urgency.

In contrast to existing schemes, we propose a proactive approach that is specifically tailored to dynamic EED-enabled environments and enables efficient resource

allocation in real time. We foster dynamic prediction of worker availability for optimized resource allocation. Furthermore, we factor in the uncertainty in worker capabilities and propose a comprehensive reliability assessment scheme that allows making better resource allocation decisions.

Chapter 3

Dynamic Worker Availability Prediction (DWAP)

In this chapter, we introduce DWAP and provide an overview of the underlying system model, problem formulation, and performance evaluation.

3.1 System Model and Overview

In DWAP, the system consists of three major components; workers, requesters, and an orchestrator. User-owned devices (i.e., workers) contribute their resources to the system in exchange for some incentives provided by the orchestrator. Requesters send the computing tasks that need to be offloaded for execution to the orchestrator. The latter plays a central role in the system by allocating the incoming tasks to participating workers based on their capabilities and availability within certain constraints and limitations. The orchestrator predicts the availability of workers using the Continuous-time Markov Chain (CTMC) model. It then makes availability-aware resource allocation decisions using such worker availability predictions. During resource allocation, the orchestrator endeavors to minimize the execution time by employing a refined optimization-based approach, akin to strategies adopted in various schemes [4, 8]. Fig. 3.1 shows a high-level overview of the system architecture of DWAP and

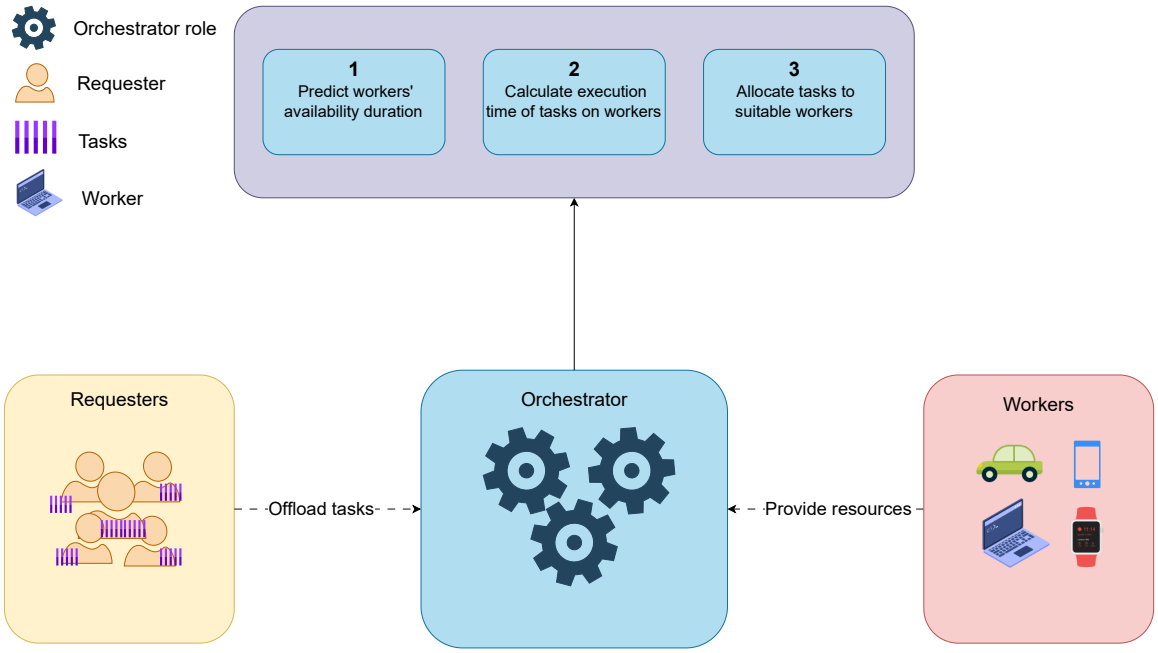

Figure 3.1: DWAP's System Architecture

the interactions between the requesters, orchestrator and workers.

Consider a set of m workers $W = \{w_1, w_2, \ldots, w_m\}$ and a set of n tasks $T = \{t_1, t_2, \ldots, t_n\}$. Each task $t_i \in T$ is associated with a certain computation workload q_i (in CPU cycles). Each worker $w_j \in W$ has a maximum CPU cycle frequency c_j^{max} (in CPU cycles/sec). The maximum number of tasks that can be executed in parallel on each worker w_j is denoted β. One of the key responsibilities of the orchestrator is to make resource allocation decisions that ensure assigning tasks to workers that can successfully complete them. Note that the successful completion of tasks is highly dependent on the availability of workers, which is predicted by the orchestrator to be subsequently used in the resource allocation process. In the latter, the orchestrator minimizes the total execution time of tasks while ensuring their successful completion by ensuring that the execution time of each task does not

exceed the availability duration of the worker to which it is allocated. For each task t_i allocated to worker w_j, the execution time depends on both the computation workload q_i and the CPU cycle frequency, denoted c_{ij} (in CPU cycles/sec), dedicated by w_j to execute t_i. To determine the CPU cycle frequency c_{ij} that worker w_j dedicates to task t_i, the maximum CPU cycle frequency c_j^{max} of each worker is divided equally among the maximum number of tasks that it can accommodate, as given by Eq. 3.1. Accordingly, the execution time required to perform task t_i on worker w_j is denoted γ_{ij} and is given by Eq. 3.2 [4, 7, 8].

$$c_{ij} = \frac{c_j^{max}}{\beta} \tag{3.1}$$

$$\gamma_{ij} = \frac{q_i}{c_{ij}} \tag{3.2}$$

Each worker w_j is estimated to be available for a certain duration α_j. Consequently, the orchestrator only assigns tasks to workers whose estimated availability is sufficient to accommodate the entire duration of task execution. To estimate the availability of any given worker, historical data on the worker's availability behavior is used and fitted into a Continuous-time Markov Chain (CTMC) model. More details on how this estimation is performed is discussed in the following section.

3.2 Prediction Methodology

In this section, we present the dataset utilized, as well as the steps taken in preprocessing it. In addition, we present the prediction process by providing a detailed description of the CTMC model used.

3.2.1 Dataset and Preprocessing

We utilize a large usage trace of real-world applications made available by Google [19, 50]. The data set is available in Google BigQuery, which is a storage and analysis tool that utilizes a variant of Structured Query Language (SQL). The dataset encompasses 8 Google computing clusters, also referred to as cells, collected in May 2019. Each cell comprises roughly 12k machines (i.e., workers) and provides detailed information on various aspects, including the characteristics of individual machines, events occurring on those machines, the jobs and tasks they perform, and the utilization of resources. Each job comprises multiple tasks, each of which can be in one of six possible states at any given time, namely Submitted, Queued, Scheduled, Evicted, Failed, or Finished. Machines can also be in one of three possible states indicating their operational status, namely Add (the machine is added to the cluster), Remove (the machine is removed from the cluster), or Update (the machine's available resources are updated). Additionally, the machines in the cells exhibit a significant level of heterogeneity. Notably, among all the cells, cell g is the most heterogeneous [51], comprising 13 different types of machines with varying CPU and memory capabilities. Thus, we use data obtained solely from cell g.

We sample the 100 most volatile and unreliable machines out of the total 12k in the cluster, while ensuring representation of all 13 CPU and memory capability combinations and proportionality to the initial population. Furthermore, the dataset provides an element of randomness; there is no distinct pattern dictating when machines are added or removed. Machines could experience hardware or software failures and tasks could get halted due to user intervention [52]. This aspect mirrors the unpredictable nature of EED-enabled EC environment, where devices may join or leave

at any time. This randomness, combined with the careful selection of the 100 most volatile machines enhances the suitability of this dataset for our application.

We construct queries to extract the necessary data. After extracting the availability traces of the sampled machines, we preprocess the events by categorizing the Add and Update events as "Available" and the Remove events as "Unavailable", since our system is only concerned with the two states of availability and unavailability. Moreover, we preprocess the time stamps into 15-minute intervals. More details on the dataset and preprocessing steps is provided in Appendix B.

3.2.2 CTMC Model

Markov processes, specifically CTMC, enable modeling the system as a set of states and transitions between states that occur probabilistically over continuous time intervals, making it possible to analyze the probabilities of different system states at various points in time [53]. The transitions between states are specified through rates at which workers transfer from one state to another, either due to failure (i.e. leaving the system) or due to repair (i.e. rejoining the system). The state diagram of worker transitions is illustrated in Fig. 3.2.

Let λ denote the worker failure rate. We express the failure rate in terms of the Mean Time Between Failure (MTBF), as given by Eq. 3.3. MTBF is a metric that measures the expected time between component failures [54]. In our setting, MTBF refers to the expected duration between workers leaving.

$$\lambda = \frac{1}{MTBF} \tag{3.3}$$

Similarly, let μ denote the worker repair rate. We express the repair rate in terms

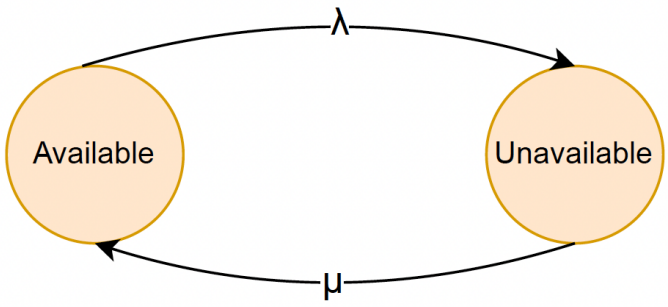

Figure 3.2: State diagram of worker transitions

of the Mean Time to Repair (MTTR), as given by Eq. 3.4. MTTR is a metric that measures the expected time to repair a component after it fails [54]. In our setting, MTTR reflects the expected time for a worker to rejoin the system.

$$\mu = \frac{1}{MTTR} \tag{3.4}$$

The state transition rate matrix for the described Markov chain is given by Q, as follows:

$$Q = \begin{bmatrix} -\lambda & \lambda \\ \mu & -\mu \end{bmatrix}, \quad \text{where } \lambda, \mu > 0$$

Let P(t) denote the transition probability matrix, where $P_{i,j}(t)$ is the transition probability from state i to state j within time t. The state transition probability equations are calculated by Eq. 3.5.

$$\frac{d}{dt}P(t) = P(t).Q \tag{3.5}$$

By considering "Available" as state 1 and "Unavailable" as state 2, and assuming that $P_{1,1}(0) = 1$, solving the differential equations as shown in Appendix A yields the transition probabilities given by Eq. 3.6, Eq. 3.7, Eq. 3.8, and Eq. 3.9.

$$P_{1,1}(t) = \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t} + \frac{\mu}{\lambda+\mu} \tag{3.6}$$

$$P_{1,2}(t) = -\frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t} + \frac{\lambda}{\lambda+\mu} \tag{3.7}$$

$$P_{2,2}(t) = \frac{\mu}{\lambda+\mu}e^{-(\lambda+\mu)t} + \frac{\lambda}{\lambda+\mu} \tag{3.8}$$

$$P_{2,1}(t) = -\frac{\mu}{\lambda+\mu}e^{-(\lambda+\mu)t} + \frac{\mu}{\lambda+\mu} \tag{3.9}$$

Using Eq. 3.6, Eq. 3.7, Eq. 3.8, and Eq. 3.9, a worker's state is predicted for a specified number of time steps ahead. A worker's availability duration, α_j, is calculated by aggregating the duration of consecutive time intervals when it is predicted to be available within these time steps. For training the prediction model, we first split the data into training and testing sets. We use this initial split to compute the failure and repair rates of each worker up until the current time step. To further refine the training and enhance predictions on worker availability, we employ the Expanding Window Cross-Validation technique [55]. This involves gradually increasing the size of the training data by incorporating more historical data through an expanding window and making predictions on a fixed window of test data. Once this window passes, the model parameters (failure and repair rates) are updated using the latest training set, which now encompasses the data from the most recent window. This iterative process of updating the model and forecasting continues for each subsequent

window of test data.

3.3 Problem Formulation

Upon estimating the availability duration of workers as described in Section 3.2, i.e., α_j, the orchestrator uses such estimations to make availability-aware resource allocation decisions. During resource allocation, the objective is to minimize the total execution time of all tasks. The execution time of tasks is expressed in terms of the computation workload of the task and the CPU cycle frequency dedicated by the worker to the task, as defined by Eq. 3.2. Consideration of task deadlines will be discussed in detail in Chapter 4. We formulate the resource allocation problem as an Integer Linear Program (ILP) problem, where the binary decision variable x_{ij} is set to 1 if task t_i is allocated to worker w_j , and 0 otherwise, as given by Eq. 4.6.

$$x_{ij} = \begin{cases} 1 & \text{if task } t_i \text{ is allocated to worker } w_j \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

The problem formulation is given by Eq. 3.11.

$$\min_{x_{ij} \in \{0,1\}} \sum_{j \in W} \sum_{i \in T} x_{ij} \gamma_{ij} \tag{3.11a}$$

subject to:

$$\sum_{j \in W} x_{ij} = 1 \qquad \forall\, i \in T \tag{3.11b}$$

$$\sum_{i \in T} x_{ij} c_{ij} \leq c_j^{max} \quad \forall\, j \in W \tag{3.11c}$$

$$x_{ij} \gamma_{ij} \leq \alpha_j \qquad \forall\, i \in T, \forall\, j \in W \tag{3.11d}$$

The objective given by Eq. (3.11a) is subject to the constraints (3.11b)-(3.11d). Constraint (3.11b) ensures that each task is assigned to one worker. This guarantees that each task is assigned only once. Constraint (3.11c) guarantees that the total CPU cycle frequency used to execute all the tasks assigned to each worker does not exceed its maximum capacity. Constraint (3.11d) ensures that the execution time of each task assigned to each worker does not exceed the estimated availability duration α_j of the worker. This is critical to ensure that a worker is not assigned a task that takes longer to complete than the time it is available to work on.

3.4 Performance Evaluation

In this section, we evaluate the performance of DWAP compared to a representative of state-of-the-art availability-oblivious resource allocation schemes [4, 8], referred to as the Availability-unaware Resource Allocation (AURA) scheme. Note that AURA adopts the same optimization problem used in DWAP but without considering the worker availability constraint. In order to assess the impact of the prediction model used in DWAP, we compare it to the prominent Exponential Moving Average (EMA) prediction model [48]. Note that the Exponential Moving Average (EMA) operates

by calculating the weighted average of past availability observations, with a bias towards more recent data points, guided by a smoothing factor that falls between 0 and 1. This factor dictates the weight allocation, thereby determining the emphasis placed on recent observations compared to older ones. We apply EMA to estimate the availability duration of workers and then apply the same availability-aware resource allocation approach used in DWAP. We refer to such a combination as the Exponential Moving Average Resource Allocation (EMARA) scheme. We use the following performance metrics:

1. **Average execution time**, which is the average time taken to execute the tasks successfully across all time steps and all workers.

2. **Task drop rate**, which is the ratio of the total number of tasks that fail to be executed to the total number of tasks.

3. **Mean Absolute Error (MAE)**, which is the average absolute difference between the predicted and actual values of worker availability.

4. **Root Mean Squared Error (RMSE)**, which is the square root of the average of the squared differences between the predicted and actual values of worker availability.

3.4.1 Simulation Setup

For the prediction model, the initial training set comprises of the first 80% of the data and the testing set comprises the remaining 20%. The window size in the Expanding Window Cross Validation is set to 30 minutes.

DWAP, EMARA, and AURA are all implemented using Python, which is integrated with Gurobi [56] to generate the optimal solution. The total number of tasks at every time step is set to 200 and the number of time steps is set to 150. The computation workload of tasks is uniformly distributed in the range of [50,000, 70,000]. Unless otherwise specified, the maximum number of tasks that workers can execute in parallel is set to 4. The maximum CPU cycle frequency of each worker is extracted from the dataset. Note that in the dataset, the values provided are normalized relative to the most powerful worker. Thus, we set the maximum possible value to 600 CPU cycles/sec and scale the values accordingly. Finally, the smoothing factor in EMARA, which determines the weight given to past observations relative to new observations, is set to 0.1.

3.4.2 Results and Analysis

In our experiments, we evaluate the performance of DWAP, EMARA, and AURA over varying β (i.e., maximum number of parallel tasks) and average task computation workload q. In addition, we evaluate the performance of the prediction models used in DWAP and EMARA over varying time intervals. All experiments are repeated 5 times and the results are presented at a confidence level of 95%. Since the confidence intervals are negligible, they are not explicitly depicted in the figures for clarity of presentation.

3.4.2.1 Prediction Results

Fig. 3.3 depicts the actual and the predicted number of available workers in DWAP and EMARA at each time interval. It can be observed that the predicted values in

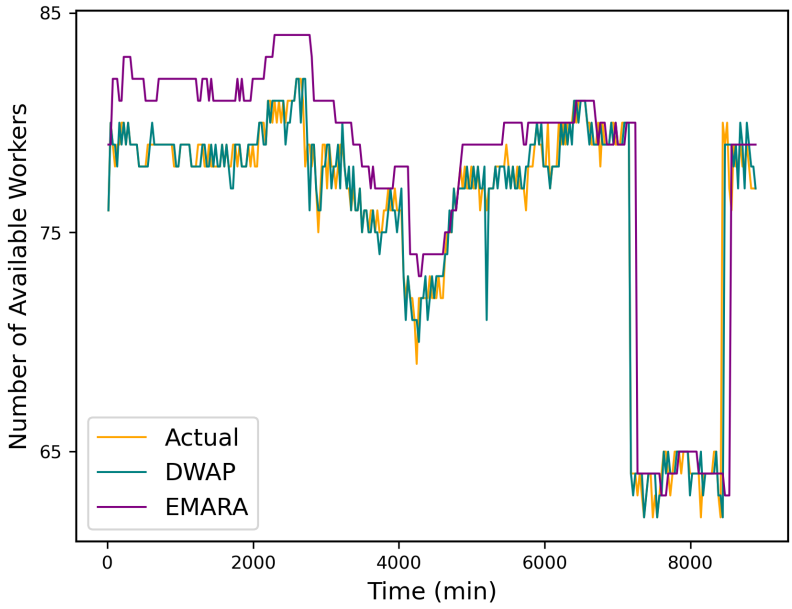

Figure 3.3: Actual and predicted number of available workers in DWAP and EMARA at each time interval

DWAP closely follow the actual data, which shows the ability of DWAP to effectively capture the overall trend of worker availability. In contrast, predictions made using EMARA substantially deviate from the actual values, indicating that it is significantly less accurate than DWAP in predicting worker availability.

Notably, DWAP is also able to capture short-term fluctuations in the data, which EMARA is unable to do. This demonstrates that DWAP is not only more accurate, but also more adaptable to changes. In order to provide a more comprehensive evaluation of the models' performance, we assess their MAE and RMSE. As depicted in Table 3.1, DWAP outperforms EMARA in terms of both metrics, achieving an improvement of 74% and 59% in terms of MAE and RMSE, respectively. This can be attributed to DWAP's utilization of state transitions, consideration of time intervals between events, and dynamic parameter adjustments, which effectively capture

Table 3.1: MAE and RMSE of DWAP and EMARA

Scheme	MAE	RMSE
DWAP	0.58	1.36
EMARA	2.31	3.39

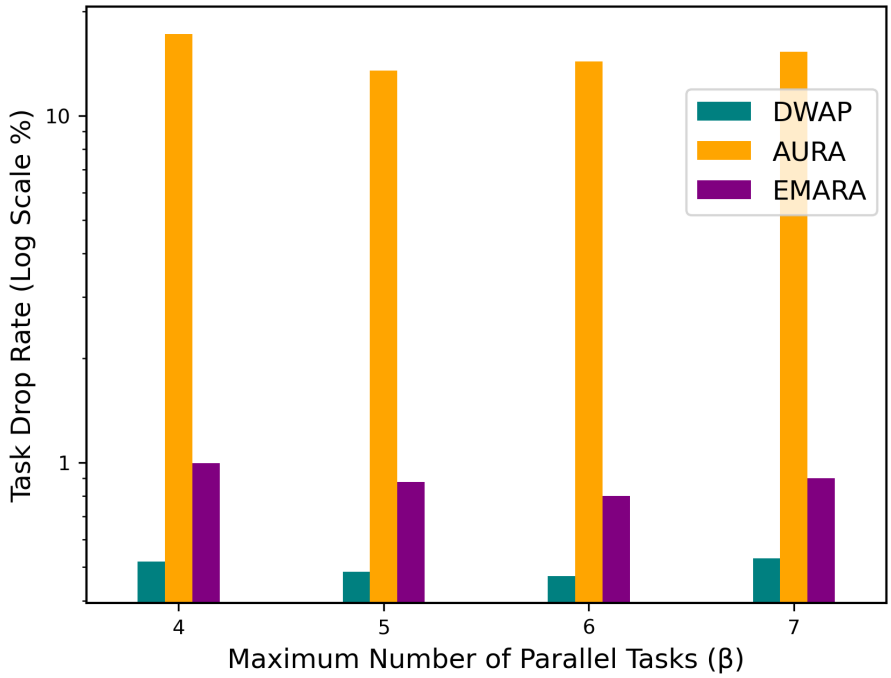

Figure 3.4: Task drop rate of DWAP, AURA, and EMARA over varying number of parallel tasks (β)

data dynamics and temporal dependencies. EMARA, on the other hand, employs a weighted average of previous data points, which may fall short in capturing complex patterns or sudden shifts.

3.4.2.2 The Impact of the Maximum Number of Parallel Tasks

Fig. 3.4 shows the task drop rate of DWAP, AURA, and EMARA over varying the maximum number of parallel tasks β. The results show that the task drop rate generally decreases as β increases in all of the schemes. This is since as β increases,

more tasks are assigned to the same worker, increasing the benefit gained due to the worker's availability for the entire duration of task execution, thus reducing the task drop rate. However, it can be observed that at some point, this benefit can be counteracted by the risk of assigning too many tasks to workers with shorter availability duration than the task execution time. Note that DWAP significantly outperforms AURA, with an improvement of up to 97%. This is because in contrast to AURA, DWAP accounts for worker availability and only assigns tasks to workers that are predicted to be available for the entire duration of task execution, thus reducing the risk of tasks being dropped due to intermittent availability of workers. In addition, DWAP significantly outperforms EMARA, yielding an improvement of up to 48%. This is since DWAP provides more accurate estimations of workers' availability than EMARA, thus reducing the risk of assigning tasks to workers that are estimated to be available longer than they actually are.

Fig. 3.5 depicts the performance of DWAP, AURA, and EMARA in terms of the average execution time over varying β. The results show that as the value of β increases, the average execution time increases in all the schemes. This is because as β increases, the computational capability dedicated to each individual task decreases, resulting in a prolonged execution time. Note that the three schemes yield fairly comparable results, with DWAP and EMARA closely approaching the performance of AURA, wherein the latter demonstrates a marginal performance gain of up to 4% over DWAP and EMARA. This difference can be attributed to the fact that AURA focuses solely on minimizing the execution time without considering worker availability. In contrast, DWAP and EMARA take both factors into consideration, which results in a more constrained selection of available resources. Consequently,

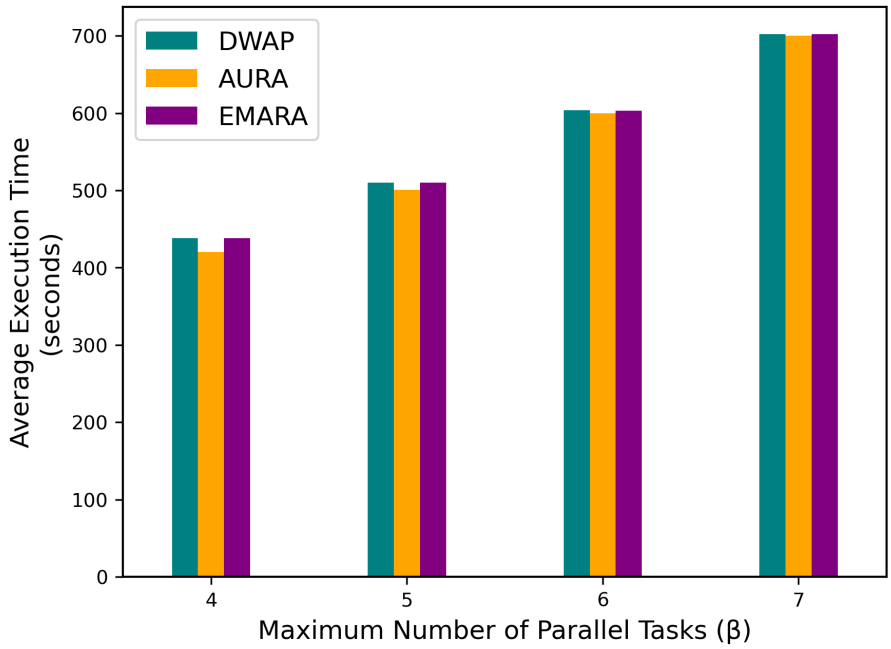

Figure 3.5: Average execution time of DWAP, AURA, and EMARA over varying number of parallel tasks (β)

DWAP demonstrates significant improvements in terms of task drop rate compared to AURA, while still managing to closely approach AURA's execution time performance, resulting in a remarkably small gap of up to 4%.

3.4.2.3 The Impact of the Average Task Computation Workload

Fig. 3.6 depicts the drop rate of DWAP, AURA, and EMARA over varying average computation workload (q). It can be observed that increasing the computation workload of tasks does not significantly impact the resource allocation decision process. Instead, increasing q only makes the tasks more computationally intensive, which can prolong the average execution time, but does not affect the resource allocation decision. This is since the latter is designed to optimize task distribution based on

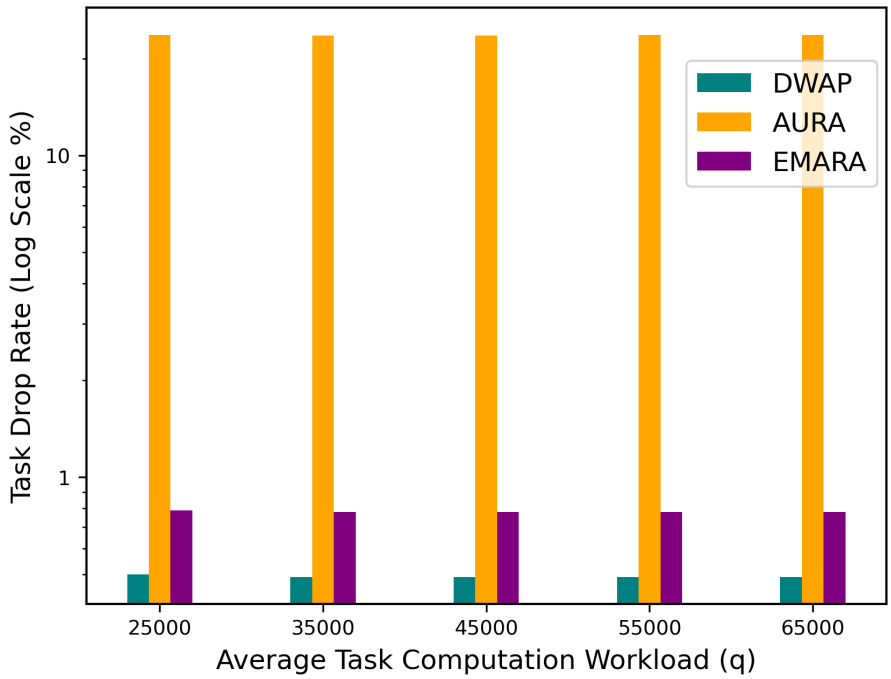

Figure 3.6: Task drop rate of DWAP, AURA, and EMARA over varying task computation workload (q)

the available worker pool and their characteristics, with the goal of minimizing execution time and enhancing overall system efficiency. Since the criteria governing worker selection remain unchanged, the resource allocation decision process remains largely unaffected by variations in the average task computation workload. Thus, as q increases, the task drop rate of DWAP, AURA, and EMARA remains the same.

Chapter 4

DWAP-Reputation Enhanced (DWAP-RE)

In this chapter, we introduce DWAP-RE. While DWAP primarily focuses on predicting the availability duration of workers, DWAP-RE takes a step further by also considering their reputation scores. DWAP, similar to existing schemes, overlooks the potential discrepancies between the workers' capabilities perceived by the orchestrator and their actual capabilities. These discrepancies often emerge due to rapid changes in the dynamic environment. As such, the real-time capabilities of a worker can be far less than the orchestrator's initial assessment, largely due to the device owner utilizing more resources for personal use. This dynamic access behavior introduces a layer of unpredictability that can lead to inefficiencies in resource allocation, which can significantly prolong task execution. Moreover, DWAP primarily emphasizes the impact of accounting for the intermittent availability of workers on the reduction of task drop rate, but overlooks the reliability issues associated with the aforementioned discrepancies on satisfying the deadline of tasks. DWAP-RE effectively tackles these two challenges. In DWAP-RE, the orchestrator exploits the past performance of workers to build a reputation scoring system that evaluates their actual performance potential based on various factors. These include their time-efficiency in executing tasks

of varying workloads, the proportion of successfully completed tasks, and the ratio of executed tasks that satisfy their respective deadlines, providing a comprehensive measure of the workers' reputation. DWAP-RE relies on the fact that poor performance of a worker decreases that worker's credibility and reliability, subsequently influencing the allocation of future tasks.

4.1 System Model

DWAP-RE shares the same system model as DWAP (outlined in Section 3.1), with some additional features. These features include the fact that each task t_i is associated with a deadline d_i (in seconds). Note that if the orchestrator allocates a task to a worker and the deadline is not met, the task will not be dropped but the task requester will not be satisfied with the overall quality or level of service provided. Furthermore, as a consequence of the discrepancy induced by the dynamic user access behavior, the orchestrator only has access to a perception of the maximum CPU cycle frequency, denoted as $\hat{c}_j^{max}$ (in CPU cycles/sec), as opposed to the actual maximum frequency, c_j^{max}. This perceived frequency might be an overestimation. This obfuscation means that the orchestrator might have imprecise estimation of the actual execution time of tasks.

Imprecise estimations of the actual capabilities of workers lead to poor resource allocation decisions and cause the workers' performance to be unreliable. This is since overestimating the performance of workers can make the orchestrator assign them computationally intensive tasks that end up taking longer execution time than anticipated, significantly impacting the delay, satisfaction ratio, and drop rate. In order to account for these reliability issues in DWAP-RE, the orchestrator resorts

to scoring the reputation of each worker w_j, denoted s_j , based on the worker's past performance. The reputation score acts as a reliability measure that assesses the level of confidence in the worker's estimated performance. High reputation scores reflect a consistent ability to effectively execute tasks, which thereby instills a degree of trust in the worker's future performance. As such, these reputation scores become a crucial tool for the orchestrator to make reliability-aware resource allocation decisions, in the presence of inherent discrepancies. More details on the criteria included in the reputation score and how it is calculated are discussed in the following section.

The perceived CPU cycle frequency $\hat{c}_{ij}$ (in CPU cycles/sec), dedicated by worker w_j to execute t_i, is given by Eq. 4.1, where β is the maximum number of tasks that w_j can execute in parallel. Thus, the perceived execution time of executing task t_i on worker w_j is denoted $\hat{\gamma}_{ij}$ and is given by Eq. 4.2 [4, 7, 8], where q_i is the computation workload of the task (in CPU cycles).

$$\hat{c}_{ij} = \frac{\hat{c_j}^{max}}{\beta} \tag{4.1}$$

$$\hat{\gamma}_{ij} = \frac{q_i}{\hat{c}_{ij}} \tag{4.2}$$

4.2 Reputation Scoring

In this section, we present the criteria used to assess the reputation of each worker, as well as the mechanism adopted to calculate the reputation score.

4.2.1 Criteria for Worker Reputation Assessment

There are four criteria that determine the worker's overall reputation. They are outlined as follows:

1. *Task Success Rate*: This criterion represents the proportion of tasks successfully executed by a worker relative to the total tasks assigned. Note that the success of tasks on any worker relies on the worker's availability during the entire duration needed for the task to be executed. Thus, it is crucial to consider this criterion when determining the reputation of any given worker as it serves as an indicator of its reliability upon task assignment. Workers demonstrating a low task success rate are likely to receive fewer tasks in subsequent allocations, as a low task success rate contributes to low reputation.

2. *Satisfaction Ratio*: This criterion represents the ratio of the tasks that the worker has executed within their corresponding deadline to the total number of assigned tasks in previous allocations. It serves as an indicator of the worker's ability to consistently provide a certain QoS demanded by the requesters to maintain their satisfaction. Note that determining how fast the workers tend to execute their allocated tasks can also provide further insights, which necessitates the use of the average execution time criterion (explained below).

3. *Average Execution Time*: This criterion represents the average execution time of all the tasks that have been successfully executed by the worker in previous allocations. Its significance in the reputation assessment process stems from the fact that it can indicate the worker's consistent ability to sustain low latency. A low average execution time serves as a strong indicator of a reliable performance,

particularly when combined with the other relevant criteria.

4. *Average Task Computation Workload*: This criterion represents the average computation workload of all the tasks that have been successfully executed by the worker in previous allocations. It provides key insights into the worker's ability to handle tasks of various workloads, particularly computationally intensive tasks. This criterion can complement the other criteria. For example, together with the average execution time criterion, it can provide a more insightful indicator of how fast a worker executes tasks relevant to their computation workload. The pertinence of the average task computation workload criterion can be elaborated further using the following examples:

    - Consider two workers A and B. Both workers exhibit equally low average execution time. However, Worker A demonstrates a higher average task computation workload than Worker B. This indicates that Worker A possesses greater performance as it successfully handles larger and more demanding tasks within the same time frame as Worker B.
    - Consider two workers C and D. Both workers display a high task success rate, meaning that they complete most tasks assigned to them successfully. However, Worker C has a higher average task computation workload than Worker D. This means that while both workers are equally successful at completing tasks, Worker C is more reliable since it is able to handle tasks with larger workloads while having the same success rate as Worker D.
    - Consider two workers E and F. Both workers demonstrate a high satisfaction ratio, implying that they complete most tasks assigned to them

within the specified deadline. However, Worker E has a higher average task computation workload than Worker F. Typically, larger workloads correspond to more complex or demanding tasks. Hence, Worker E's ability to handle computationally demanding tasks while maintaining a high satisfaction ratio indicates greater reliability in managing demanding tasks within specified deadlines.

4.2.2 Reputation Score Calculation

The criteria outlined in the previous section are measured on different scales. Consequently, normalization is applied before calculating the overall reputation score. To do that, it is first important to note that the criteria can be classified into the following two categories [57]:

1. *Beneficial Criteria*: These represent the criteria for which a higher value is favorable. The satisfaction ratio, task success rate, and average task computation workload are classified under this category.

2. *Non-beneficial Criteria*: Conversely, these encompass the criteria for which a lower value is preferable. In the context of this study, the average execution time is included in this category.

Let $K = \{k_1, k_2, k_3, k_4\}$ be the set of the 4 aforementioned criteria, where the value of criterion k_r for each worker w_j is denoted v_{rj}. To normalize the criteria, the Min-Max method [58] is employed. For beneficial criteria, the normalization equation is given by Eq. 4.3, whereas for non-beneficial criteria, the normalization equation is given by Eq. 4.4. Note that the task success rate and satisfaction ratio are not normalized, since their values are already within the range of [0,1].

$$v'_{rj} = \frac{v_{rj} - \min_j(v_{rj})}{\max_j(v_{rj}) - \min_j(v_{rj})} \tag{4.3}$$

$$v'_{rj} = \frac{\max_j(v_{rj}) - v_{rj}}{\max_j(v_{rj}) - \min_j(v_{rj})} \tag{4.4}$$

Following the aforementioned normalization, the reputation score of worker w_j, denoted s_j, can then be calculated as given by Eq. 4.5.

$$s_j = \sum_{r=1}^{4} v'_{rj} \tag{4.5}$$

In calculating the reputation score, the utilization of an aggregation method naturally leads to a scenario where all criteria hold equal weight, fostering balanced performance across all metrics. This ensures fairness, preventing potential imbalances that can occur with unequal weighting. For instance, consider a scenario where one worker showcases a very high success rate but registers a low satisfaction ratio. In a system where the satisfaction ratio carries lower weight, this worker could be deemed superior to another worker that maintains a slightly lower success rate yet excels in satisfaction ratio. By sustaining an equal weight approach, we promote a more comprehensive and equitable evaluation, encouraging workers to maintain consistent performance in all criteria, rather than overemphasizing a single criterion to increase their score. To validate our methodological approach, we also explored using the Weighted Sum Method (WSM) to calculate the reputation score s_j. In this method, individual criteria are allocated specific weights, and the final reputation score is derived by summing up these weighted values. To determine the weights, the Analytic Hierarchy Process (AHP) [59] was used. Experimental evaluations confirmed

that assigning equal weights to all four criteria yields the most optimal results.

4.3 Problem Formulation

Upon assessing the workers' reputation scores, the orchestrator incorporates them to make reliability-aware resource allocation decisions. During resource allocation, we strive to minimize the total execution time of all tasks. However, the orchestrator makes resource allocation decisions based on its perceived estimation of the workers capabilities and performance potential. To account for possible discrepancies, the orchestrator relies on workers' reputation scores to evaluate their dependability and the level of confidence in their performance. This evaluation takes into account the workers' previous track record, contributing to a more reliable and efficient task execution process. A higher worker reputation correlates with a reduced disparity between its perceived and actual performance. The worker's reputation score is incorporated into the resource allocation process. The main objective of the latter is to minimize the total task execution time and the unreliability of the workers to which the tasks are allocated. In this context, the time required for task execution is divided by the worker's reputation score, addressing performance variations based on their historical reliability. This approach fosters a non-linear relationship, enabling a more precise depiction of the dynamics between execution time and reliability of workers. Moreover, by incorporating the reputation score in this manner, we underscore its substantial and multiplier effect on the process. Toward that end, we formulate the resource allocation problem as an ILP, where the binary decision variable x_{ij} is set to 1 if task t_i is allocated to worker w_j , and 0 otherwise, as given by Eq. 4.6.

$$x_{ij} = \begin{cases} 1 & \text{if task } t_i \text{ is allocated to worker } w_j \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The problem formulation is given by Eq. 4.7.

$$\min_{x_{ij} \in \{0,1\}} \sum_{j \in W} \sum_{i \in T} x_{ij} \frac{\hat{\gamma}_{ij}}{s_j} \tag{4.7a}$$

subject to:

$$\sum_{j \in W} x_{ij} = 1 \qquad \forall\, i \in T \tag{4.7b}$$

$$\sum_{i \in T} x_{ij} \hat{c}_{ij} \leq \hat{c}_j^{max} \quad \forall\, j \in W \tag{4.7c}$$

$$x_{ij} \frac{\hat{\gamma}_{ij}}{s_j} \leq \alpha_j \qquad \forall\, i \in T, \forall\, j \in W \tag{4.7d}$$

The objective given by Eq. (4.7a) is subject to the constraints (4.7b)-(4.7d). Constraint (4.7b) ensures that each task is assigned to one worker. Constraint (4.7c) guarantees that the total perceived CPU cycle frequency used to execute all the tasks assigned to each worker does not exceed its perceived maximum capacity. Constraint (4.7d) ensures that the perceived execution time of each task assigned to each worker does not exceed the estimated availability duration α_j of the worker.

4.4 Performance Evaluation

In this section, we evaluate the performance of DWAP-RE compared to DWAP and a representative of state-of-the-art reliability-aware schemes that rely solely on the task success rate to assess the reliability of workers [13, 14]. We refer to the latter

as the Conventional Evaluation for Reputation-based Resource Allocation (CERRA) scheme. Note that in contrast to DWAP and DWAP-RE, CERRA does not apply any availability prediction. Thus, to ensure a fair comparison and to solely and explicitly evaluate the impact of the reputation scoring system, we tweak CERRA by implementing the same availability prediction-based approach used in DWAP-RE. Note that DWAP, DWAP-RE, and CERRA are all evaluated within the context of discrepancies between the perceived and actual capabilities of workers. We use the following performance metrics:

1. **Average execution time**, which is the average time taken to execute tasks successfully.

2. **Task drop rate**, which is the ratio of the total number of tasks that fail to be executed to the total number of tasks.

3. **Satisfaction Ratio**, which is the ratio of the number of tasks that are successfully executed within the deadline to the total number of tasks.

4.4.1 Simulation Setup

DWAP, DWAP-RE, and CERRA are all implemented using Python, which is integrated with Gurobi [56] to generate the optimal solution. We set the number of time steps to 150 and the number of tasks in each time step to 200. Unless otherwise specified, the computation intensity of tasks is uniformly distributed in the range of [70,000, 90,000], the maximum number of tasks that workers can execute in parallel is set to 4, and the deadline of tasks is uniformly distributed in the range of [900, 1200]. We use the same set of 100 workers as in DWAP. The maximum CPU cycle

frequency of each worker is extracted from the Google dataset[19, 50]. Note that in the dataset, the values provided are normalized relative to the most powerful worker. Thus, we set the maximum possible value to 600 CPU cycles/sec and scale the values accordingly.

4.4.2 Results and Analysis

In our experiments, we evaluate the performance of DWAP, DWAP-RE, and CERRA over varying β (i.e., maximum number of parallel tasks), average task deadline d, and average computation workload q. All experiments are repeated 5 times. The results are presented at a confidence level of 95%. Note that the rendered confidence intervals are shown in Fig. 4.2. Since the confidence intervals are negligible, they are not explicitly depicted in the other figures for clarity of presentation.

4.4.2.1 The Impact of the Maximum Number of Parallel Tasks

Fig. 4.1 shows the effect of varying the maximum number of parallel tasks, β, on the average execution time of tasks for DWAP, DWAP-RE, and CERRA. It can be observed that the average execution time increases in all schemes as β increases. This can be attributed to the fact that as the number of tasks that are executed concurrently by the worker increases, the worker's computational capacity available for each individual task decreases. Consequently, the time required to complete each task increases, leading to an increase in the overall average execution time. Note that DWAP-RE yields the best results among all schemes. It outperforms DWAP by up to 43%. This is since unlike DWAP, DWAP-RE accounts for the reliability issues that ensue from the presence of disparities between the workers' capabilities perceived by

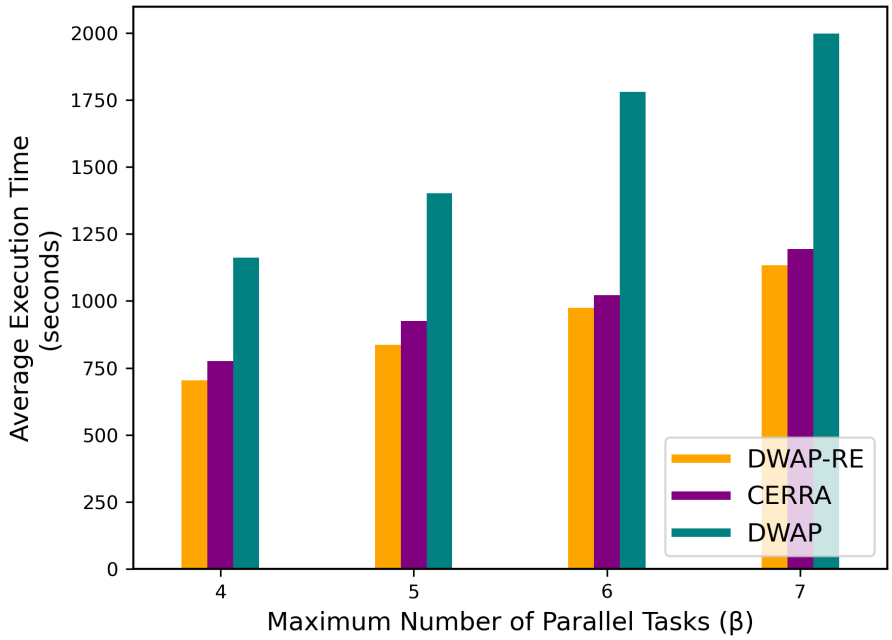

Figure 4.1: Average execution time of DWAP, DWAP-RE, and CERRA over varying number of parallel tasks (β)

the orchestrator and their actual capabilities. DWAP-RE assesses the reputation of workers based on their past performance and leverages the use of reputation scores to provide valuable insights on the proficiency and consistency of their performance. This reduces the risk of prolonging the execution time by assigning computationally intensive tasks to poor workers that are perceived to be more capable than they actually are. Moreover, DWAP-RE outperforms CERRA by 10%. This is because in contrast to CERRA, which relies solely on the task drop rate to address the reliability issues of workers, DWAP-RE implements a comprehensive reputation scoring system. This system assesses the quality and consistency of the performance of workers based on a number of metrics. Along with the task drop rate, these metrics include the average time that the worker takes to execute its allocated tasks, the average computation intensity of the tasks that it executes, and its ability to satisfy the deadline of those

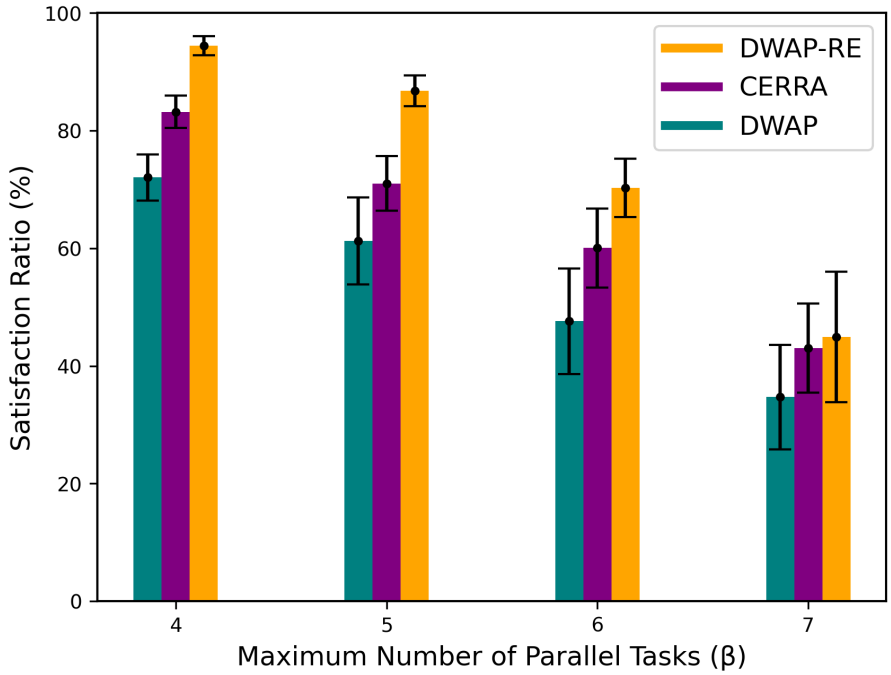

Figure 4.2: Satisfaction ratio of DWAP, DWAP-RE, and CERRA over varying number of parallel tasks (β)

tasks. The collective use of those metrics provides much more valuable insights into the performance of workers and enables the orchestrator to make more dependable resource allocation decisions. This increases the chance of assigning computationally intensive tasks to consistently high-performing workers, thus reducing the average execution time.

Fig. 4.2 shows the effect of varying the maximum number of parallel tasks, β, on the satisfaction ratio of DWAP, DWAP-RE, and CERRA. The results show that the overall satisfaction ratio decreases for all schemes as β increases. This is since workers are able to dedicate less of their resources to each of the concurrent tasks. Consequently, this increases the execution time of each task, which subsequently leads to a higher likelihood of missing the set deadlines. Additionally, DWAP-RE

significantly outperforms DWAP and CERRA by up to 42% and 22%, respectively. This is since DWAP-RE leverages the past performance of workers to determine their reputation, and one of the metrics it uses is the satisfaction ratio of previously executed tasks. This metric, combined with others, enable the orchestrator to make allocation decisions that increase the likelihood of meeting the task deadline. In contrast, DWAP overlooks the reliability issues stemming from workers' capability discrepancies. As a result, it fails to account for the risks of assigning tasks to overestimated workers that may be too slow to meet the deadline of the tasks, thus reducing the satisfaction ratio. While CERRA does consider such discrepancies, its assessment of worker reliability relies solely on the task drop rate. This approach falls short in providing insights into the worker's time efficiency and consistency in meeting the deadline of tasks.

Fig. 4.3 shows the effect of varying the maximum number of parallel tasks, β, on the task drop rate of DWAP, DWAP-RE, and CERRA. It can be observed that as β increases, the task drop rate increases in all schemes. This is because as β increases, more tasks are assigned to the same worker, elevating the risk of workers with deteriorating availability and reliability issues handling more tasks. Note that DWAP-RE significantly outperforms DWAP by up to 97%. This is markedly different from the scenario presented in Fig. 3.4, where DWAP demonstrates superior performance in a setting where there are no discrepancies or uncertainties regarding the workers' capabilities. In contrast, under the present conditions where the orchestrator lacks precise knowledge of the individual workers' capabilities, DWAP's performance significantly deteriorates. On the other hand, DWAP-RE effectively addresses the discrepancy between the perceived and actual worker performance. Unlike DWAP, DWAP-RE

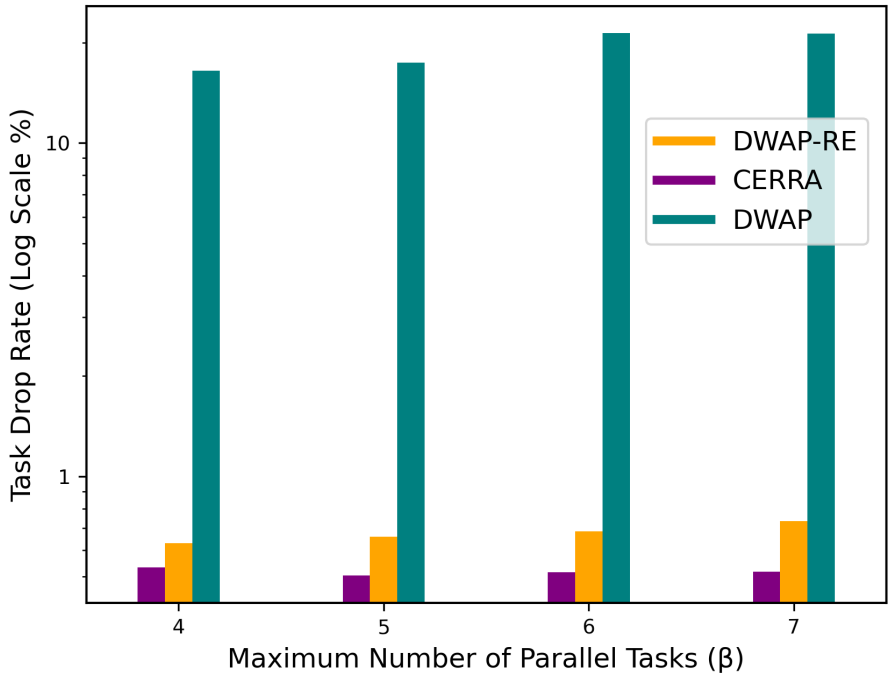

Figure 4.3: Task drop rate of DWAP, DWAP-RE, and CERRA over varying number of parallel tasks (β)

incorporates a reputation scoring system that accounts for the incongruity between the worker anticipated and actual task execution capabilities. By leveraging this reputation-based approach, DWAP-RE mitigates the risk of assigning tasks to workers who might be overestimated, potentially leading to tasks taking longer to execute than the workers' availability window. This risk mitigation significantly decreases the task drop rate. Interestingly, CERRA slightly surpasses DWAP-RE. This is since CERRA assesses the reliability of workers based solely on the task success rate, thus assigning tasks to workers with the lowest record in terms of task drop rate. In contrast, DWAP-RE employs a more holistic reputation scoring system, encompassing a broader range of criteria beyond just the task success rate. While CERRA achieves a marginal reduction in task drop rate compared to DWAP-RE, it is worth highlighting

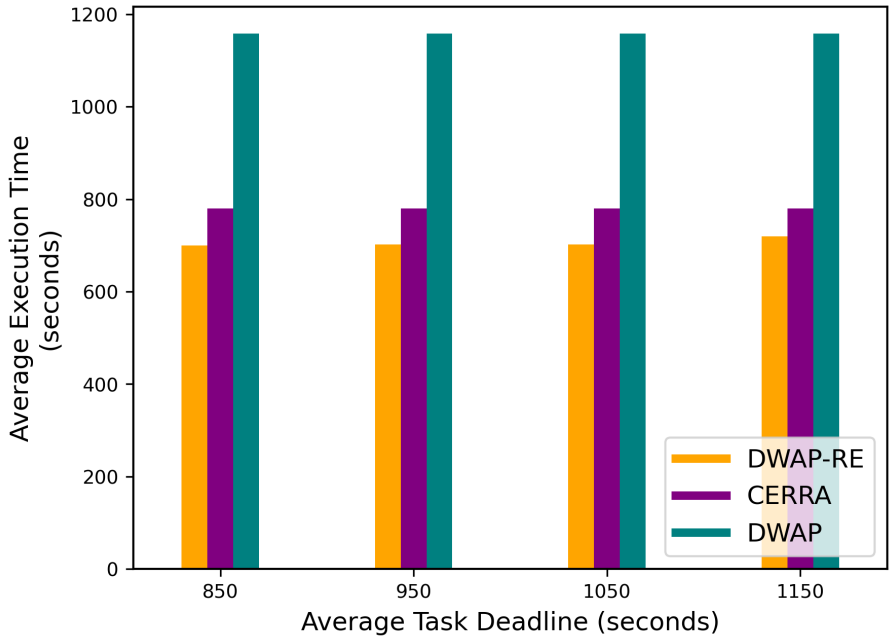

Figure 4.4: Average execution time of DWAP, DWAP-RE, and CERRA over varying task deadline (d)

that the drop rate of DWAP-RE remains consistently below 1%. This underscores the effectiveness of DWAP-RE in maintaining a remarkably low task drop rate, despite CERRA's marginal advantage in this aspect.

4.4.2.2 The Impact of the Average Task Deadline

Fig. 4.4 shows the effect of varying the average task deadline, d, on the average execution time of DWAP, DWAP-RE, and CERRA. The results indicate that the increase in task deadline has no impact on the average execution time in all schemes. This is since the task deadline is not considered in the orchestrator's resource allocation decision. Instead, the decision is primarily focused on efficiently utilizing the available resources to complete tasks as quickly as possible. It can be observed that DWAP-RE outperforms DWAP and CERRA by up to 39% and 10%, respectively.

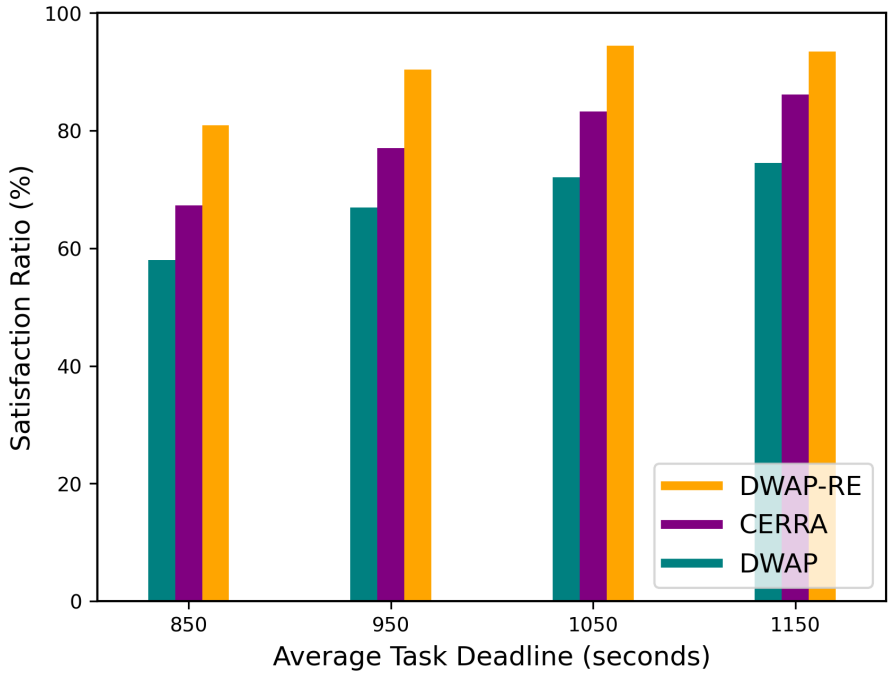

Figure 4.5: Satisfaction ratio of DWAP, DWAP-RE, and CERRA over varying task deadline (d)

This is due to the same reasons mentioned earlier.

Fig. 4.5 shows the effect of varying the average task deadline, d, on the satisfaction ratio of DWAP, DWAP-RE, and CERRA. It can be observed that as the deadline increases, the satisfaction ratio increases in all schemes. This is due to the increased flexibility provided by longer deadlines, which affords a more lenient temporal envelope for task execution. In addition, longer task deadlines act as buffers against unforeseen delays that result from the disparity between the worker's perceived and actual performance, allowing a greater proportion of tasks to be completed within their allotted deadline. Note that DWAP-RE outperforms DWAP and CERRA by up to 35% and 17%, respectively. It can be observed that the leverage of DWAP-RE

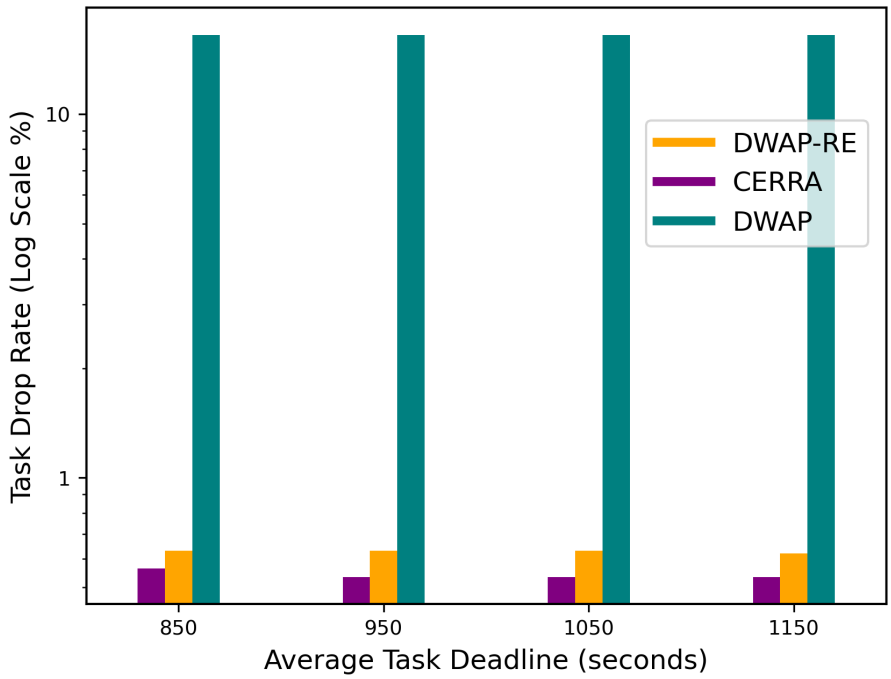

Figure 4.6: Task drop rate of DWAP, DWAP-RE, and CERRA over varying task deadline (d)

compared to DWAP and CERRA significantly manifests as the task deadlines become more strict. This is since as task deadlines become more stringent, the pressure to efficiently allocate resources intensifies. DWAP-RE's advanced reputation-based resource allocation mechanism excels in such scenarios. By factoring in workers' historical satisfaction ratio, execution time, task computation workload, and success rate, DWAP-RE makes more reliable resource allocation decisions, ensuring a higher probability of timely and successful task completion. In contrast, DWAP, which fails to account for the reliability of workers, and CERRA, which is not as adept as DWAP-RE at considering these intricate factors, might encounter challenges in maintaining efficient resource allocation under stricter deadlines.

Fig. 4.6 shows the effect of varying the average task deadline, d, on the task

drop rate of DWAP, DWAP-RE, and CERRA. It can be noted that augmenting the task deadline does not exert a discernible impact on the task drop rate. This observation is rooted in the fact that the orchestrator's resource allocation strategy remains indifferent to the deadline variations. Moreover, it is noteworthy that even if a deadline is exceeded, the task continues to be executed without getting dropped.

4.4.2.3 The Impact of the Average Task Computation Workload

Fig. 4.7 shows the effect of varying the average task computation workload, q, on the average execution time of DWAP, DWAP-RE, and CERRA. It can be observed that as the task computation workload increases, the average execution time increases in all schemes. This is attributed to the fact that more intensive tasks are inherently more demanding and thus require longer completion time. Note that DWAP-RE significantly outperforms DWAP and CERRA by up to 41% and 16%, respectively. This can be attributed to the fact that DWAP-RE includes task computation workload as one of the factors in the worker reputation evaluation process. Workers who consistently and reliably handle computationally intensive tasks, while achieving proficiency in other criteria, are likely to receive a high reputation score. Consequently, these skilled workers are more likely to be selected for executing intensive tasks, thereby enhancing the overall efficiency of the system.

Fig. 4.8 shows the effect of varying the average task computation workload, q, on the satisfaction ratio of DWAP, DWAP-RE, and CERRA. The results illustrate that as the task computation workload increases, the satisfaction ratio decreases in all schemes. This is since tasks with higher intensity require more time to be completed, subsequently increasing the risk of missing their deadline. It can be observed that

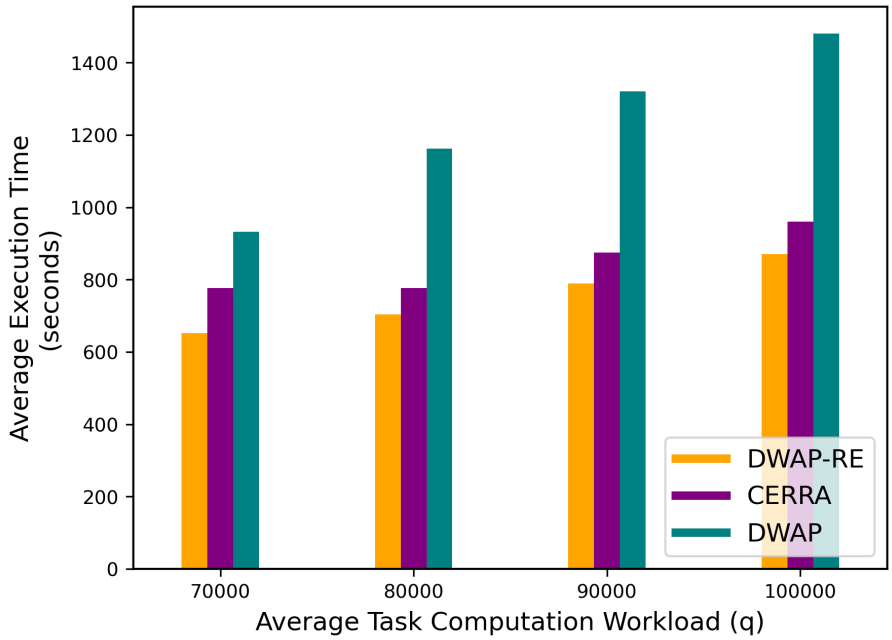

Figure 4.7: Average execution time of DWAP, DWAP-RE, and CERRA over varying task computation workload (q)

DWAP-RE outperforms DWAP and CERRA by up to 36% and 18%, respectively. This is due to the same reasons previously discussed pertaining to DWAP-RE's consideration of the average task computation workload of previously allocated tasks, along with other criteria, in the reputation scoring system.

Fig. 4.9 shows the effect of varying the average task computation workload, q, on the task drop rate of DWAP, DWAP-RE and CERRA. Note that as the task computation workload increases, the task drop rate increases in all schemes. This is since more computationally intensive tasks are associated with a higher risk of exceeding the worker's availability duration due to their longer execution time. Note that DWAP-RE significantly outperforms DWAP by up to 97%. This is due to the same reasons previously discussed pertaining to DWAP-RE's ability to address the

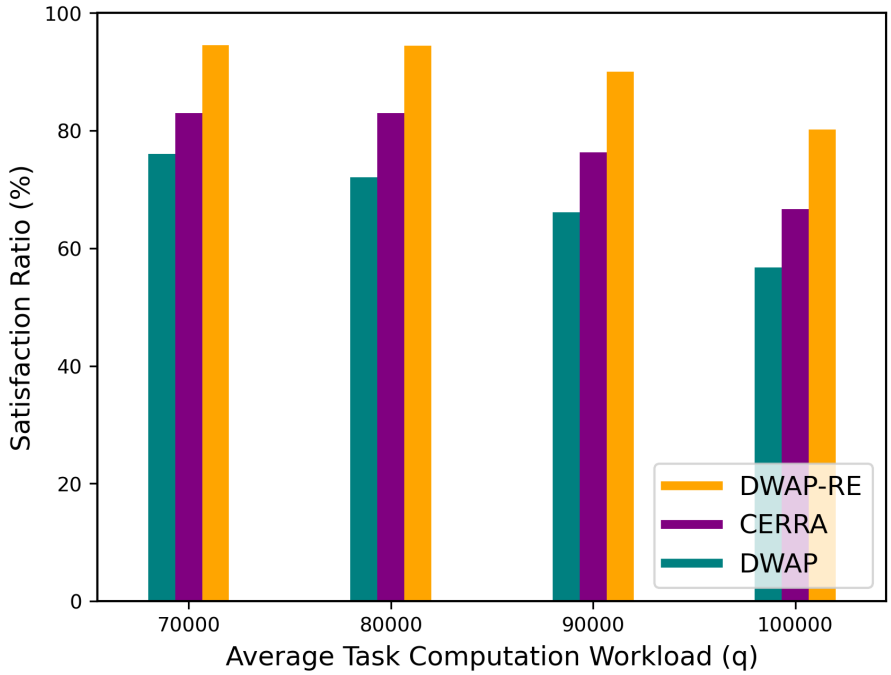

Figure 4.8: Satisfaction ratio of DWAP, DWAP-RE, and CERRA over varying task computation workload (q)

discrepancy between the perceived and actual worker performance. It can be particularly observed that the performance improvement rendered by DWAP-RE compared to DWAP increases as q increases. This is since unlike DWAP, DWAP-RE significantly alleviates the risk of assigning computationally intensive tasks to overestimated workers that end up prolonging the execution time beyond the worker's availability window. It can also be observed that CERRA achieves a marginal reduction in task drop rate compared to DWAP-RE, due to the fact that CERRA asesses the reliability of workers based solely on the task success rate, whereas DWAP-RE considers multiple other criteria along with the latter. However, it is worth highlighting that the drop rate of DWAP-RE remains consistently below 1%. This accentuates DWAP-RE's outstanding performance in upholding an exceptionally low task drop rate, even

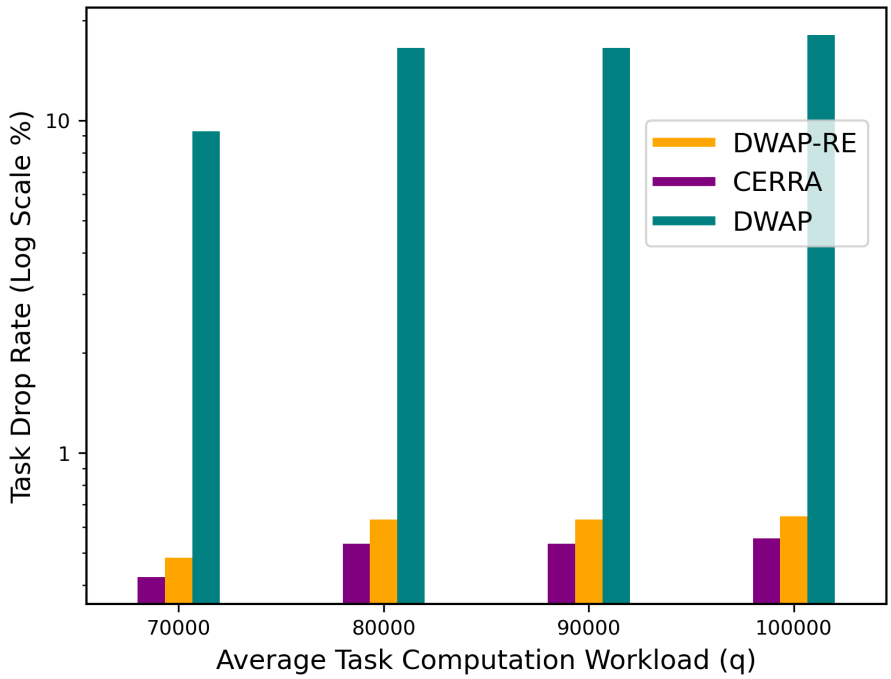

Figure 4.9: Task drop rate of DWAP, DWAP-RE, and CERRA varying task computation workload (q)

in light of CERRA's slight advantage in this respect.

Chapter 5

Conclusion and Future Work

5.1 Summary and Conclusion

Democratizing edge computing can revolutionarily transform the prolific yet underutilized computational resources of end devices, also referred to as Extreme Edge Devices (EEDs), into on-demand computing groups of distributed workers. This can speed up the feasible deployment of pervasive IoT applications. However, such transformative potential can be impeded by the challenging issues imposed by the highly dynamic nature of EEDs and the impact of such dynamicity on the Quality of Service (QoS). In this thesis, we have proposed a framework that accounts for the high dynamicity of EEDs and the resulting repercussions pertaining to their intermittent availability and the continuous fluctuations in their capabilities. To counteract the high risk of intermittent availability, we have proposed the Dynamic Worker Availability Prediction (DWAP) scheme. DWAP is the first scheme that predicts the availability of EEDs in a way that captures the high volatility of EED-enabled computing environments. DWAP uses the Continuous-Time Markov Chain (CTMC) model to predict worker availability while continually refining its parameters to incorporate new incoming

data. To generate such predictions, we have utilized a dataset based on real-world Google cluster workload traces. We have then used the predicted availability of workers to make availability-aware resource allocation decisions.

In addition to accounting for the intermittent availability of workers, we have addressed the reliability issues associated with the dynamic fluctuations in workers' capabilities. For that purpose, we have extended DWAP and proposed the DWAP-Reputation Enhanced (DWAP-RE) scheme. DWAP-RE accounts for potential discrepancies between the orchestrator's perception of the capabilities of workers and their actual performance. This is of paramount importance given the intrinsic dynamic user access behavior and computation load in EED-enabled computing environments. To mitigate the effects of such dynamicity, DWAP-RE not only predicts the availability duration of workers but also evaluates their reputation scores, providing insights into the historical reliability of workers based on previous task executions. Moreover, DWAP-RE acknowledges the fact that executing tasks in a timely manner is as important as successfully executing them, specially for time-critical tasks. To capture that importance, DWAP-RE employs a comprehensive reputation scoring system that is based on multiple criteria. Subsequently, DWAP-RE uses the reputation scores of workers to make reliability-aware resource allocation decisions, further improving the QoS.

Extensive experiments have shown that DWAP yields significant improvements of 74% and 59% in terms of the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), respectively, compared to a representative of state-of-the-art prediction schemes. Furthermore, DWAP yields 97% and 48% reduction in task drop rate

compared to prominent availability-oblivious and availability-aware resource allocation schemes, respectively. This is attributed to DWAP's superior capability to precisely predict the availability duration of workers. Moreover, DWAP-RE significantly outperforms prominent reliability-oblivious and reliability-aware resource allocation schemes by up to 43% and 16%, and 42% and 22% in terms of execution time and satisfaction ratio, respectively. In addition, it outperforms prominent reliability-oblivious resource allocation schemes by 97% in terms of task drop rate. Such performance underscores DWAP-RE's proficiency in addressing the discrepancies between the orchestrator's perception of the capabilities of workers and their actual performance and making resource allocation decisions that reduce delay and task drop rate, while ensuring timely completion of tasks.

5.2 Future Work

In the future, we plan to implement Deep Learning algorithms, such as Long Short-Term Memory (LSTM) [60], to discern patterns within user access behavior and computation load. LSTM, renowned for its aptitude in learning from temporal sequence data over time, will be harnessed to uncover intricate patterns inherent in user access behavior and computation load. However, one of the key challenges we anticipate involves dealing with data that falls Out-of-Distribution (OoD). This represents scenarios and data points that deviate significantly from the patterns the model is trained on. Given the highly volatile and dynamic nature of user access behavior and computational loads, encountering OoD data is not a rare exception—it is often the norm. Thus, LSTMs may encounter difficulties when attempting to make predictions on such OoD data. To counteract this concern, we plan to integrate Uncertainty

Quantification (UQ) techniques [61–63] into the prediction model. By utilizing UQ, we can provide a probabilistic interpretation to predictions on OoD data, thereby improving the orchestrator's ability to deal with uncertain scenarios, while maintaining desirable QoS.

To leverage the aforementioned potential of UQ, we intend to adopt Deep Evidential Regression (DER) [64], which is a recent approach that uses the concept of evidential theory in Deep Learning to manage and quantify uncertainty in predictions. Unlike traditional methods that require training multiple versions of a model (like ensembles) [62] or applying techniques such as dropout to estimate uncertainty [63], DER modifies the output layer of a single deep learning model to produce the parameters of an evidential distribution. In doing so, this single model not only delivers predictions but also quantifies the associated uncertainty level. This approach conserves computational resources while providing a nuanced comprehension of model uncertainty. The proficiency of DER in producing evidential distributions from a single model pass, along with its capability to handle OoD data and quantify predictive uncertainty, positions it as an efficient tool for our dynamic and volatile environment.

Furthermore, given the inherent uncertainties in the environment, the orchestrator might make suboptimal resource allocation decisions. Such decisions, which involve assigning tasks to workers who are not ideally suited for them, culminate in an opportunity loss: a better-suited worker might have handled the task more efficiently, optimizing the overall outcome. To address this, we plan to incorporate the output from the UQ in the resource allocation framework, with the objective of minimizing the expected opportunity loss.

References

[1] Mark Hung. Leading the IoT, gartner insights on how to lead in a connected world. *Gartner Research*, 1:1–5, 2017.

[2] David Reinsel-John Gantz-John Rydning, J Reinsel, and J Gantz. The digitization of the world from edge to core. *Framingham: International Data Corporation*, 16, 2018.

[3] Ruslan Kain, Sara A Elsayed, Yuanzhu Chen, and Hossam S Hassanein. Multistep prediction of worker resource usage at the extreme edge. In *Proceedings of the 25th International ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pages 25–32, 2022.

[4] Marah De'bas, Sara A Elsayed, and Hossam S Hassanein. Multitiered worker-oriented resource allocation at the extreme edge. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 5674–5679, 2022.

[5] Larry Peterson, Tom Anderson, Sachin Katti, Nick McKeown, Guru Parulkar, Jennifer Rexford, Mahadev Satyanarayanan, Oguz Sunay, and Amin Vahdat. Democratizing the network edge. *ACM SIGCOMM Computer Communication Review*, 49(2):31–36, 2019.

[6] Distributive. `https://distributive.network`, Accessed on March, 2023.

[7] Ibrahim M Amer, Sharief MA Oteafy, Sara A Elsayed, and Hossam S Hassanein. QoS-based task replication for alleviating uncertainty in edge computing. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 5147–5152, 2022.

[8] Rawan F El Khatib, Sara A Elsayed, Nizar Zorba, and Hossam S Hassanein. Optimal proactive resource allocation at the extreme edge. In *ICC 2022-IEEE International Conference on Communications*, pages 5657–5662, 2022.

[9] Ehab Saleh and Chandrasekar Shastry. Task migration in volunteer computing systems. In *2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, pages 2076–2079, 2022.

[10] Sachini Jayasekara, Shanika Karunasekera, and Aaron Harwood. Optimizing checkpoint-based fault-tolerance in distributed stream processing systems: Theory to practice. *Software: Practice and Experience*, 52(1):296–315, 2022.

[11] Yousef Alsenani, Garth Crosby, and Tomas Velasco. Sara: A stochastic model to estimate reliability of edge resources in volunteer cloud. In *2018 IEEE international conference on EDGE computing (EDGE)*, pages 121–124, 2018.

[12] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent, and David P Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of SETI@home. *IEEE Transactions on Parallel and Distributed Systems*, 22(11):1896–1903, 2011.

[13] Jason Sonnek, Abhishek Chandra, and Jon Weissman. Adaptive reputation-based scheduling on unreliable distributed infrastructures. *IEEE Transactions on Parallel and Distributed Systems*, 18(11):1551–1564, 2007.

[14] Gary A McGilvary, Adam Barker, and Malcolm Atkinson. Ad hoc cloud computing. In *2015 IEEE 8th international conference on cloud computing*, pages 1063–1068, 2015.

[15] Xiaofeng Wang, Chee Shin Yeo, Rajkumar Buyya, and Jinshu Su. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. *Future Generation Computer Systems*, 27(8):1124–1134, 2011.

[16] Alessandro Celestini, Alberto Lluch Lafuente, Philip Mayer, Stefano Sebastio, and Francesco Tiezzi. Reputation-based cooperation in the clouds. In *Trust Management VIII: 8th IFIP WG 11.11 International Conference, IFIPTM 2014, Singapore, July 7-10, 2014. Proceedings 8*, pages 213–220, 2014.

[17] Yousef Alsenani, Garth V Crosby, Tomas Velasco, and Abdulrahman Alahmadi. Remot reputation and resource-based model to estimate the reliability of the host machines in volunteer cloud environment. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 63–70, 2018.

[18] Adnan Umer, Adnan Noor Mian, and Omer Rana. Predicting machine behavior from google cluster workload traces. *Concurrency and Computation: Practice and Experience*, 35(5):e7559, 2023.

[19] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–14, 2020.

[20] Peter Mell and Tim Grance. The NIST definition of cloud computing. 2011.

[21] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[22] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: Saving IoT from the cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[23] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[24] Cisco Fog Computing Solutions. Unleash the power of the internet of things. *Cisco Systems Inc*, 2015.

[25] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S Tucker. Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Communications*, 34(5):1728–1739, 2016.

[26] Fabio Giust et al. Mec deployments in 4g and evolution towards 5g. 24(2018):1–24, 2018.

[27] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[28] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—A key technology towards 5G. *ETSI white paper*, 11(11):1–16, 2015.

[29] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[30] Pengzhan Hao, Yongshu Bai, Xin Zhang, and Yifan Zhang. Edgecourier: An edge-hosted personal service for low-bandwidth document synchronization in mobile cloud storage services. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–14, 2017.

[31] Yaser Jararweh, Loai Tawalbeh, Fadi Ababneh, and Fahd Dosari. Resource efficient mobile computing using cloudlet infrastructure. In *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 373–377. IEEE, 2013.

[32] Abdalla A Moustafa, Sara A Elsayed, and Hossam S Hassanein. Community-oriented resource allocation at the extreme edge. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 5583–5588. IEEE, 2022.

[33] Tessema M Mengistu and Dunren Che. Survey and taxonomy of volunteer computing. *ACM Computing Surveys (CSUR)*, 52(3):1–35, 2019.

[34] Arpita Ray, Chandreyee Chowdhury, Subhayan Bhattacharya, and Sarbani Roy. A survey of mobile crowdsensing and crowdsourcing strategies for smart mobile device users. *CCF Transactions on Pervasive Computing and Interaction*, 5(1):98–123, 2023.

[35] A Stephen McGough, Matthew Forshaw, John Brennan, Noura Al Moubayed, and Stephen Bonner. Using machine learning to reduce the energy wasted in volunteer computing environments. In *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, 2018.

[36] Jurairat Phuttharak and Seng W Loke. A review of mobile crowdsourcing architectures and challenges: Toward crowd-empowered internet-of-things. *Ieee access*, 7:304–324, 2018.

[37] Bakhta Meroufel and Ghalem Belalem. Optimization of checkpointing/recovery strategy in cloud computing with adaptive storage management. *Concurrency and Computation: Practice and Experience*, 30(24):e4906, 2018.

[38] Vania Boccia, Luisa Carracciuolo, Giuliano Laccetti, Marco Lapegna, and Valeria Mele. Hadab: Enabling fault tolerance in parallel applications running in distributed environments. In *Parallel Processing and Applied Mathematics: 9th*

International Conference, PPAM 2011. Revised Selected Papers, Part I 9, pages 700–709. Springer, 2012.

[39] Bakhta Meroufel and Ghalem Belalem. Service to fault tolerance in cloud computing environment. *WSEAS Transactions on Computers*, 14(1):782–791, 2015.

[40] Eugen Feller, John Mehnert-Spahn, Michael Schoettner, and Christine Morin. Independent checkpointing in a heterogeneous grid environment. *Future Generation Computer Systems*, 28(1):163–170, 2012.

[41] D Manivannan, Qiangfeng Jiang, Jianchang Yang, and Mukesh Singhal. A quasi-synchronous checkpointing algorithm that prevents contention for stable storage. *Information Sciences*, 178(15):3110–3117, 2008.

[42] Andrew Stephen McGough and Matthew Forshaw. Analysis of reinforcement learning for determining task replication in workflows. In *European Workshop on Performance Engineering*, pages 117–132. Springer, 2022.

[43] Yuxuan Sun, Jinhui Song, Sheng Zhou, Xueying Guo, and Zhisheng Niu. Task replication for vehicular edge computing: A combinatorial multi-armed bandit based approach. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2018.

[44] Daniel Nurmi, John Brevik, and Rich Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, 2005. Proceedings 11*, pages 432–441. Springer, 2005.

[45] Karthick Ramachandran, Hanan Lutfiyya, and Mark Perry. Decentralized approach to resource availability prediction using group availability in a P2P desktop grid. *Future Generation Computer Systems*, 28(6):854–860, 2012.

[46] John Violos, Tita Pagoulatou, Stylianos Tsanakas, Konstantinos Tserpes, and Theodora Varvarigou. Predicting resource usage in edge computing infrastructures with CNN and a hybrid bayesian particle swarm hyper-parameter optimization model. In *Intelligent Computing: Proceedings of the 2021 Computing Conference, Volume 2*, pages 562–580. Springer, 2021.

[47] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Complexity vs. performance: empirical analysis of machine learning as a service. In *Proceedings of the 2017 Internet Measurement Conference*, pages 384–397, 2017.

[48] Xiaoheng Deng, Jun Li, Enlu Liu, and Honggang Zhang. Task allocation algorithm and optimization model on edge collaboration. *Journal of Systems Architecture*, 110:101778, 2020.

[49] Davide Vega, Roc Meseguer, Felix Freitag, and Sergio F Ochoa. Effort-based incentives for resource sharing in collaborative volunteer applications. In *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 37–42. IEEE, 2013.

[50] Google. Cluster data 2019 traces, Accessed on September, 2023.

[51] Yuhui Lin, Adam Barker, and Sheriffo Ceesay. Exploring characteristics of inter-cluster machines and cloud applications on google clusters. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2785–2794, 2020.

[52] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–17, 2015.

[53] Mustafa M Al-Sayed, Sherif Khattab, and Fatma A Omara. Prediction mechanisms for monitoring state of cloud resources using markov chain model. *Journal of Parallel and Distributed Computing*, 96:163–171, 2016.

[54] Jinxi Li, Deke Guo, Junjie Xie, and Sheng Chen. Availability-aware provision of service function chains in mobile edge computing. *ACM Transactions on Sensor Networks*, 19(3):1–28, 2023.

[55] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[56] Gurobi. Gurobi optimizer reference manual. `https://www.gurobi.com/`, 2022. Accessed on March, 2023.

[57] Deepika Patil and Eyhab Al-Masri. Seamless service migration across multi-access edge computing (MEC) environments. In *2021 IEEE 3rd Eurasia Conference on IoT, Communication and Engineering (ECICE)*, pages 369–375. IEEE, 2021.

[58] Vatsal Gajera, Rishabh Gupta, Prasanta K Jana, et al. An effective multi-objective task scheduling algorithm using min-max normalization in cloud computing. In *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pages 812–816. IEEE, 2016.

[59] Thomas L Saaty. *What is the analytic hierarchy process?* Springer, 1988.

[60] Alex Graves and Alex Graves. Long short-term memory. *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 37–45, 2012.

[61] Moloud Abdar et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

[62] Rahul Rahaman et al. Uncertainty quantification and deep ensembles. *Advances in Neural Information Processing Systems*, 34:20063–20075, 2021.

[63] Harrison Cusack and Alina Bialkowski. The effect of training data quantity on monte carlo dropout uncertainty quantification in deep learning. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2023.

[64] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. *Advances in Neural Information Processing Systems*, 33:14927–14937, 2020.

Appendix A

Derivation of the State Transition Probabilities

The state transition probability matrix for the state diagram shown in Fig. 3.2 is given as follows:

$$P(t) = \begin{bmatrix} P_{1,1}(t) & P_{1,2}(t) \\ P_{2,1}(t) & P_{2,2}(t) \end{bmatrix}$$

Thus, Eq.3.5 becomes:

$$\begin{bmatrix} P'_{1,1}(t) & P'_{1,2}(t) \\ P'_{2,1}(t) & P'_{2,2}(t) \end{bmatrix} = \begin{bmatrix} P_{1,1}(t) & P_{1,2}(t) \\ P_{2,1}(t) & P_{2,2}(t) \end{bmatrix} \begin{bmatrix} -\lambda & \lambda \\ \mu & -\mu \end{bmatrix}$$

Multiplying the matrices leads to the following equations:

$$P'_{1,1}(t) = P_{1,1}(t)(-\lambda) + P_{1,2}(t)(\mu) = P_{1,1}(t)(-\lambda - \mu) + \mu \tag{A.1}$$

$$P'_{2,2}(t) = P_{2,1}(t)(\lambda) + P_{2,2}(t)(-\mu) = P_{2,2}(t)(-\lambda - \mu) + \lambda \tag{A.2}$$

While $P_{i,j}(t)$ needs to be computed for each pair $i, j \in \{1, 2\}$, we know that

$$P_{1,2}(t) + P_{1,1}(t) = P_{2,1}(t) + P_{2,2}(t) = 1, \qquad \forall t \geq 0 \tag{A.3}$$

Thus, it is sufficient to solve just for $P_{1,1}(t)$ and $P_{2,2}(t)$. Furthermore, since the system is symmetric, it is sufficient to solve for $P_{1,1}(t)$ and interchange λ to μ and vice versa to obtain $P_{2,2}(t)$.

In Eq. (A.1), the homogeneous part adopts the differential equation form $y' = ay$. The solution, which can be obtained by employing the method of integrating factors or separation of variables, is given as:

$$P^h_{1,1}(t) = Ce^{-(\lambda+\mu)t}, \qquad C \in \mathbb{R} \tag{A.4}$$

The particular solution of Eq. (A.1), obtained by setting the derivative of $P^p_{1,1}(t)$ to zero, results in a constant value:

$$\frac{d}{dt}P^p_{1,1}(t) = 0 = -(\lambda + \mu)P^p_{1,1}(t) + \mu$$

$$P^p_{1,1}(t) = \frac{\mu}{\lambda + \mu} \tag{A.5}$$

Summing the homogeneous (A.4) and particular (A.5) solutions yields $P_{1,1}(t)$:

$$P_{1,1}(t) = P^h_{1,1}(t) + P^p_{1,1}(t) = Ce^{-(\lambda+\mu)t} + \frac{\mu}{\lambda + \mu} \tag{A.6}$$

The constant C can be determined by applying the initial condition $P_{1,1}(0) = 1$, which gives:

$$P_{1,1}(0) = 1 = C + \frac{\mu}{\lambda + \mu}$$

Solving for C subsequently yields:

$$C = \frac{\lambda}{\lambda + \mu}$$

Substituting the value of C in Eq. (A.6) results in the final expression of $P_{1,1}(t)$ as follows:

$$P_{1,1}(t) = \frac{\lambda}{\lambda + \mu} e^{-(\lambda+\mu)t} + \frac{\mu}{\lambda + \mu} \tag{A.7}$$

Deriving $P_{1,2}(t)$ from the equation $P_{1,2}(t) = 1$ - $P_{1,1}(t)$ yields:

$$P_{1,2}(t) = -\frac{\lambda}{\lambda + \mu} e^{-(\lambda+\mu)t} + \frac{\lambda}{\lambda + \mu} \tag{A.8}$$

Leveraging the symmetry of the system, $P_{2,2}(t)$ can be obtained from $P_{1,1}(t)$:

$$P_{2,2}(t) = \frac{\mu}{\lambda + \mu} e^{-(\lambda+\mu)t} + \frac{\lambda}{\lambda + \mu} \tag{A.9}$$

Subsequently, deriving $P_{2,1}(t)$ from the equation $P_{2,1}(t) = 1$ - $P_{2,2}(t)$ yields:

$$P_{2,1}(t) = -\frac{\mu}{\lambda + \mu} e^{-(\lambda+\mu)t} + \frac{\mu}{\lambda + \mu} \tag{A.10}$$

Appendix B

Dataset Description and Preprocessing Details

B.1 Dataset Description

In 2011, Google published a 1-month trace of its Borg cluster management system which includes 12.6k machines in one cell. In 2019, Google updated and expanded this dataset. The new version covers a larger scale, including 96.4k machines spread across 8 cells, with each cell comprising around 12k machines [19, 50].

Google's Borg system is a cluster management tool that operates hundreds of thousands of jobs. These jobs come from several thousands of distinct applications and are executed across multiple cells, each comprising up to tens of thousands of machines. Each cell has a central controller called the Borgmaster and an agent, the Borglet, on every machine. The Borgmaster manages tasks by communicating with Borglets, handling user requests, allocating resources, scheduling tasks, and monitoring the system's health. Meanwhile, each Borglet starts or stops tasks on its machine and keeps the Borgmaster updated about its status. Users of Borg are typically Google developers and system administrators/site reliability engineers (SREs) that run Google's applications and services. When these developers and SREs want to run

an application or service, they submit a job to the Borg system. Each job consists of one or more tasks, and each task is a running instance of the program that the developer or SRE wants to run. Tasks could fail due to reasons such as hardware failures, software errors, resource exhaustion, user intervention and preemption. Furthermore, not all resources of a machine are typically dedicated to processing tasks. For instance, a portion of the machine's resources are used to run the operating system and various background services that are necessary for the operation of the machine and the network.

The dataset is accessible only through Google's BigQuery due to its size (2.4TiB). BigQuery is a fully-managed, serverless data warehouse solution offered by Google Cloud. It is designed to handle large datasets, providing fast SQL analytics. This allows formulating queries to obtain specific data, rather than downloading the entire dataset. The dataset comprises 5 tables. These tables encompass information on machine attributes (i.e., capacities), machine states, executed tasks, task states, and resource utilization. Machines are identified by unique IDs, which serve as a reference for all machine-related operations and queries. Each machine is associated with specific CPU and memory capacities. Additionally, a machine could have three possible events: Add (the machine is added to the cluster), Remove (the machine is removed from the cluster), or Update (the machine's available resources are updated). Tasks are also identified by unique IDs, ensuring that each task can be tracked and referenced independently. Tasks could be in the following possible states: Submitted, Queued, Scheduled, Evicted, Failed, or Finished. Additionally, the resource usage of each task is also recorded.

Some of the information in the dataset (such as the CPU and memory capacities

CPU Capacity	Memory Capacity	Number of Machines
0.591796875	0.333496094	1560
0.259277344	0.166748047	2643
0.708984375	0.333496094	1332
1	0.5	2103
0.259277344	0.333496094	343
0.38671875	0.166748047	967
0.38671875	0.333496094	808
0.591796875	0.166748047	530
0.958984375	0.5	956
0.958984375	1	161
1	0.25	474
0.708984375	0.666992188	845
1	1	74

Table B.1: Machine Distribution by CPU and Memory Capacity in Google Dataset

of machines) is obfuscated for confidentiality reasons. Google achieves this by using a normalization approach: both CPU and memory sizes are divided by the maximum size observed throughout the entire trace. Consequently, the memory capacity of the machine with the highest memory is represented as 1.0. Similarly, the CPU capacity of the machine with the most highest CPU is also represented as 1.0. This ensures that the relative capacities of machines can be analyzed and the exact specifications remain confidential.

Table B.1 shows the distribution of the approximately 12k machines based on their CPU and memory capacities from Cluster g. A noteworthy observation derived from the data is the heterogeneous composition of the cluster, indicating a considerable variety in the capabilities of the individual machines.

B.2 Preprocessing Details

Among all 8 cells, cell *g* is the most heterogeneous [51], comprising 13 different types of machines with varying CPU and memory capacities. Thus, we use data obtained solely from cell *g*. This cell comprises roughly 12k machines.

For our model, which focuses on the general states of machine availability, we preprocess the machine events as follows: both 'Add' and 'Update' events are categorized as 'Available', while 'Remove' events are categorized as 'Unavailable'. To preserve the heterogeneous nature and volatility of the environment, we select the 100 most volatile machines from the available pool of 12k. We employ a stratified sampling technique to ensure this selection is representative. In this method, machines are divided into distinct strata (i.e., categories) based on their CPU and memory capacities. From each category, we sample the most volatile machines while also ensuring that our sample retains proportionality with the larger population of 12k machines.

Furthermore, the traces provide timestamps indicating when specific events occur, without detailing the total operational or repair times of the machines. Moreover, the counts for 'leave' events (i.e., 'failures'), or 'add' events (i.e., 'repairs') are not explicitly provided either. These metrics are vital for determining the Mean Time Between Failure (MTBF) and Mean Time to Repair (MTTR) necessary for our CTMC model. Thus, using the raw timestamp data paired with the associated machine events, we construct queries to deduce these four metrics.