HPC Project

CS528: High Performance Computing

Project Report

Task:

Trust Aware Scheduling of Online Tasks in FoG Nodes

Authors:

Aditya Gupta	210101010
Riya Mittal	210101089
Sahil Mehul Danayak	210101092

Instructor:

Aryabartta Sahu, Dept. of CSE, IITG

Contents

1 Problem

The problem at hand is to develop a trust-aware scheduling system for online tasks in Fog (Fog Computing) nodes. These tasks arrive online and are associated with users, each with specific reliability requirements and deadlines. The system consists of multiple Fog nodes, each with shared trust values and individual trust ratings from users.

The goal is to schedule tasks in a way that minimizes cost, meets reliability requirements, and satisfies deadline constraints while considering the trustworthiness of the nodes and users.

1.1 Problem Statement

Given a stream of tasks arriving online, each associated with a user and characterized by arrival time, execution time, and relative deadline, we aim to devise a trust-aware scheduling approach for Fog nodes. The system consists of M homogeneous Fog nodes, initially having shared trust values derived from individual user trust ratings, initially set at 0.5. Trust levels influence the cost of executing tasks, with the cost function defined as $C = Base + BR \times ST[m]^2$, where Base is approximately 30% of the brand constraint BR. Furthermore, nodes are associated with task failure probabilities, which vary over time, impacting task execution success. Failure probability is modeled as $FP = \exp(-f \times t)$, where f represents the failure rate of the node and t is the task's execution time. The scheduling algorithm must ensure reliability, meet deadline constraints, and minimize costs for each task.

Our objective is to develop a scheduling algorithm that optimally allocates tasks to Fog nodes, ensuring reliability, meeting deadlines, and minimizing costs.

2 Assumptions in the Study

- The cost of executing tasks on a machine follows the formula $Base + BR \times ST[m]^2$, where Base is uniformly set to 0.3 $\times$ BR across all machines.
- The base cost of each machine is randomly selected from a range of 1 to 50.
- The failure rate of each machine is recalculated after every 10000 time steps, calculated as the ratio of unsuccessful tasks to the total tasks performed. Initially, this rate is assumed to be uniform across all machines.
- The average task arrival rate is set at 0.2 per time unit.
- The maximum duration of any task is limited to 50 time units.
- The reliability measure for a user's machine is determined directly by the level of trust in that machine.
- If a task fails during execution, it will not be rescheduled, and its cost for the time it ran will be considered.

- Preemption of task is not allowed, and deadline is assumed to be hard deadline.

3 Approach

In our simulation, we initiate each time step by storing arriving tasks in our waiting area. Then, we iterate over all currently running machines, probabilistically checking for failures. If a task fails, it is terminated on that machine, and its finishing time is set to the current time. Subsequently, we calculate the shared trust for each machine by averaging the direct trust values.
Next, if we have available machines and tasks to schedule, we proceed with scheduling. Initially, we remove tasks that cannot be scheduled due to deadline constraints. We then identify pairs of machines and tasks that satisfy the reliability constraint, storing them for potential scheduling. A task can be scheduled on a machine if its reliability requirement is less than or equal to the direct trust of the user in the machine. We then sort the candidate tasks based on various heuristics, scheduling them in this determined order and removing them from the waiting area. We introduce four distinct heuristics and conduct comparative analyses under various input parameters. We aim to minimize three key metrics: the cost of execution, the number of jobs failing to meet deadlines, and the number of jobs failing due to high reliability requirements. The cost of execution for a job takes into account only the time for which it ran on the machine. To quantify the overall performance of each heuristic, we normalize and aggregate these three parameters. The total cost is computed by assigning weights to each metric and summing them. We present a detailed analysis of which heuristic performs better in minimizing which aspect of the cost.

$$\text{Total Cost} = w_1 \times C_1 + w_2 \times C_2 + w_3 \times C_3 \tag{1}$$

In this equation:

- w_1, w_2, and w_3 represent the weights assigned to each component.
- C_1 denotes the normalized cost of execution.
- C_2 denotes the normalized fraction of jobs failed due to deadline
- C_3 denotes the normalized fraction of jobs failed due to reliability

To test the algorithm, we initially generate input tasks according to certain probability distributions. Specifically, the arrival rate of tasks follows a Poisson distribution, while the execution time and reliability of each task follow uniform distributions ranging from 1 to 50 and from 0 to 1, respectively. These input tasks are stored for further processing.
During simulation, we progress through time steps, and at each time step, we match the arrival rate of tasks with the current time to emulate an online task arrival scenario. We implement a rolling horizon technique [2] to manage the scheduling of tasks, where a waiting area serves as a simulated queue. The scheduling algorithm is invoked whenever a machine completes the execution of the previous task. This approach ensures that tasks are scheduled in real time without the need for rescheduling once they are already scheduled.

3.1 Heuristic 1: Combined cost

This heuristic takes into account all three parameters that we are trying to optimize, the execution cost, the number of jobs that failed due to deadline, and the number of jobs that failed due to high-reliability requirements. It is directly proportional to the execution cost and inversely proportional to the remaining time to the deadline. Jobs with lower laxity are scheduled first. If both these parameters are the same, the job is scheduled on the machine with the lowest shared trust that is acceptable for that job.

3.2 Heuristic 2: Cost Optimization

This heuristic focuses only on the cost of execution of a task on a particular machine. We schedule a task that offers the least cost on available machines at any given time. It aims to minimize the total cost by minimizing the execution cost which is the dominant factor in the total cost.

3.3 Heuristic 3: Deadline-based Scheduling

The idea behind this heuristic is motivated by the Earliest Deadline First or EDF Scheduling. Jobs with lower laxity are prioritized for scheduling. If some jobs have the same laxity, then we consider the shared trust of the machine. The idea behind this is that the cost of execution depends on shared trust so we try to schedule jobs on machines with the lowest shared trust to minimize the cost of execution. Thus, it tries to minimize two parameters of the cost function: the execution cost and the number of jobs that failed due to the deadline.

3.4 Heuristic 4: User trust and Reliability

This heuristic operates by selecting a machine with a direct trust slightly above the reliability requirement of the task. The rationale behind this approach is to reduce the number of job failures resulting from reliability requirements. By avoiding the allocation of tasks with low reliability to highly trusted machines, resources are not wasted, maximizing the efficiency of machine usage.

3.5 Heuristic 5: Random Selection

This heuristic selects jobs randomly and is used for comparison purposes. The results show that all the above heuristics outperform the random heuristic in minimizing cost.

4 Pseudocode

Algorithm 1 Random Task Generation

Input: Number of users, Mean arrival rate, Maximum execution time, Value of K
Output: List of tasks for each user
for each user **do**
  for each task of the user **do**
    Arrival time ← Poisson distribution(mean arrival rate)
    Execution Time ← Uniform Distribution between 1 and max Execution Time
    User Info ← User ID
    Reliability ← Random Uniform Distribution between 0 and 1
    Starting Time ← -1
    Relative Deadline ← K+ Exec Time
  end for
end for

Algorithm 2 Task Scheduling

Input: List of arriving tasks, Number of machines, Number of users, Initial failure probability f
Output: Total cost of execution, Number of failed jobs, Number of jobs not scheduled, Number of jobs failed due to reliability requirements
while there are arriving tasks in the waiting area **do**
  Add arriving tasks to the waiting area or rolling horizon queue
  while there are tasks in the waiting area **do**
    Find list of free machines
    Filter out machines not satisfying reliability requirements
    Start scheduled task if possible on a machine based on heuristic
    if current time % 100000 == 0 **then**
      Update failure rate f
    end if
    if task started execution **then**
      Remove task from waiting area
      while task is executing **do**
        Fail task with failure rate f at each time stamp
      end while
    end if
    if task failed **then**
      Remove task from waiting area
    end if
    Update parameters like direct trust, shared trust, cost of execution based on task success/failure
  end while
end while

5 Mathematical Models

The global environment parameters were set as follows: meanArrivalRate = 5.0, numberOfUsers = 10, maxExecutionTime = 50, K (relative deadline) = 500, numTasksPerUser = 100, max Simulation time = 1×10^8, initial failure probability ($f' = 1 - e^{-f}$) as 0.001, and numberOfMachines = 10. These parameters were iterated upon to adjust the graphs for better visibility as required.
Upon task arrival, tasks were directed to a waiting area acting as a rolling horizon, where the task scheduling considered both existing tasks in the waiting area and future incoming tasks. At every 10000th timestamp, the failure probability (f) was updated using the formula:

$$\text{failure probability}(f') = \frac{\text{num of jobs failed by that machine}}{(\text{num of jobs scheduled by that machine}) + 1}$$

Here, f' represents the probability of failure of a task on a machine at unit timestamp.
Thus, the probability of task success on a machine in a unit timestamp is given by:

$$1 - f' = e^{-f}$$

The probability of a task's success throughout its execution time (t) is given by:

$$e^{-ft}$$

Consequently, the probability of task failure during its execution time (t) is given by:

$$1 - e^{-ft}$$

The direct trust of a user is updated using the formula:

$$\text{direct trust of a machine} = \frac{\text{number of successful tasks given by user } u_i \text{ done by machine } m_i}{\text{total number of tasks given by user } u_i \text{ done by machine } m_i}$$

Additionally, each machine is initialized with 50 successful tasks from every user out of 100 tasks, setting the direct trust value to 0.5.
Tasks within the rolling horizon are scheduled (provided they meet reliability requirements) using a scheduling algorithm. In this study, we compare the proficiency of five different heuristics.

5.1 Heuristic 1: Combined Cost and Reliability

This heuristic considers both cost and reliability factors for task scheduling. The heuristic cost (*heuristic_cost*) of scheduling a task is determined by various parameters including the execution cost of the task, the laxity of the task, and the machine by the machine with the least direct trust which is able to schedule the task. The cost calculation formula incorporates these factors to optimize both cost and reliability simultaneously. The heuristic cost is given as:

$$heuristic\_cost = \frac{Execution\ Cost \times Machine\ Pref}{Laxity}$$

where the terms are given by

- $Execution\ Cost$
  $= task.executionTime \times ((0.3) \times base\_cost[machine\_index]$
  $+ (base\_cost[machine\_index] \times shared\_trust[machine\_index] \times shared\_trust[machine\_index]))$
- $Machine\ Pref = direct\_trust[machine\_index][task.userInformation] - task.reliability$
- $Laxity = task.relativeDeadline - task.executionTime - task.arrivalTime$

5.2 Heuristic 2: Cost Optimization

This heuristic focuses solely on optimizing the cost aspect of task scheduling. The cost calculation formula is similar to Heuristic 0 but excludes the reliability and latency factor, resulting in a simpler cost optimization approach.

$$heuristic\_cost = (task.executionTime) \times ((0.3) \times base\_cost[machine\_index] + (base\_cost[machine\_index] \times shared\_trust[machine\_index] \times shared\_trust[machine\_i$$

5.3 Heuristic 3: Deadline-based Reliability Scheduling

This heuristic optimizes task scheduling by considering both deadline constraints and machine trustworthiness. It prioritizes tasks with shorter remaining time until deadline and assigns them to machines with lower shared trust values, ensuring reliability.

$$heuristic\_cost = (task.relativeDeadline - task.executionTime - task.arrivalTime) \times shared\_trust[machine\_index]$$

5.4 Heuristic 4: User Trust and Reliability

This heuristic chooses the machine by the machine with the least direct trust which is able to schedule the task.

$$heuristic\_cost = (direct\_trust[machine\_index][task.userInformation] - task.reliability)$$

5.5 Heuristic 5: Random Selection

This heuristic involves random selection of tasks without considering any specific optimization criteria. It serves as a baseline for comparison with other heuristics.

$$heuristic\_cost = 1$$

6 Experimental Setup

In our simulation, we designed an experimental setup to model a dynamic task scheduling environment. The setup involved generating tasks for multiple users, each characterized by various parameters such as the mean arrival rate, the number of tasks per user, and the reliability of tasks. To begin the simulation, we generated a list of tasks based on the specified inputs, ensuring each task possessed attributes, including arrival time, execution time, user information, relative deadline, and reliability. Tasks were then sorted based on their arrival time, a crucial step in simulating the chronological progression of events. As the simulation time advanced, tasks whose arrival coincided with the current simulation time were placed in the waiting area, signifying their readiness for scheduling. This systematic approach enabled the emulation of real-world task scheduling scenarios, facilitating the evaluation of various scheduling algorithms and system configurations.

7 Result and Analysis

7.1 Graphs

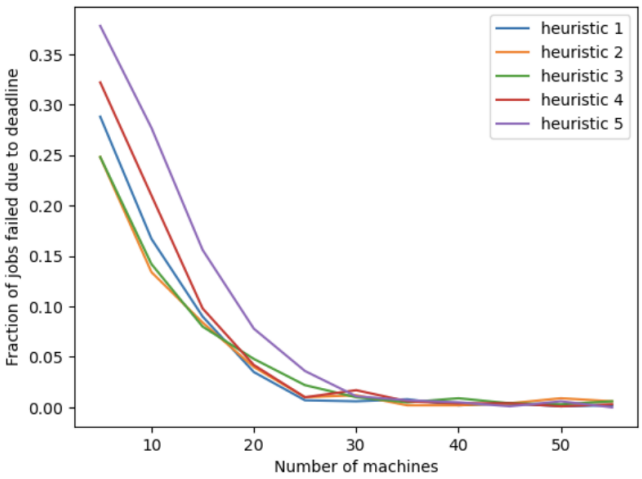

Figure 1: $K = 500$, Number of users = 10, Number of tasks per user=100

The reduction in the number of unscheduled jobs due to deadline constraints is observed with an increasing number of machines. As the number of machines increases, the capacity to schedule jobs concurrently also grows, resulting in higher job throughput.

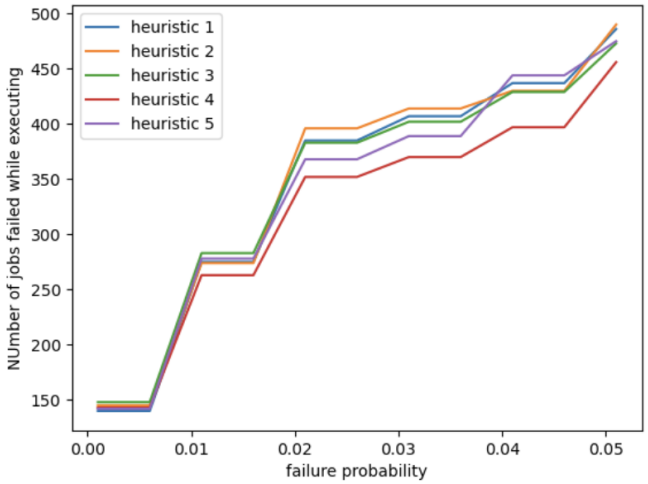

Figure 2: $K = 600$, Number of users = 10, Number of tasks per user = 100, Number of machines = 50
This graph illustrates a direct correlation between the number of jobs failing and the failure rate attributed to the machine. As the failure rate increases, so does the number of job failures.

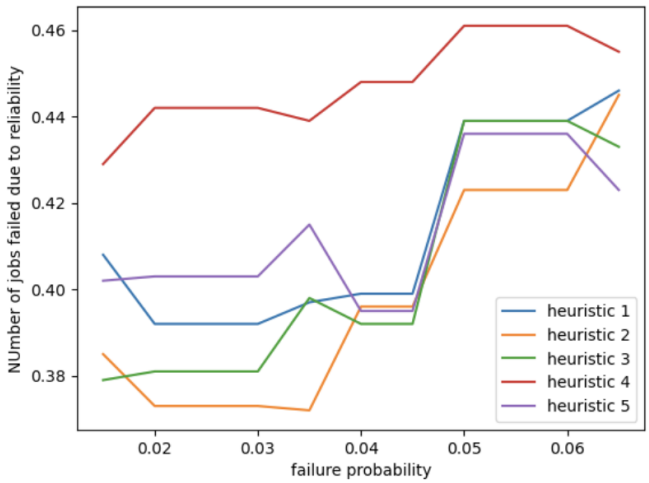

Figure 3: $K = 600$, Number of users = 10, Number of tasks per user = 100, Number of machines = 50
This graph demonstrates that as the failure probability increases, a greater number of jobs experience failure. Consequently, both the direct trust and shared trust in the machine decrease, leading to an inability to meet the reliability requirements of tasks. Consequently, the number of job failures attributed to reliability also increases

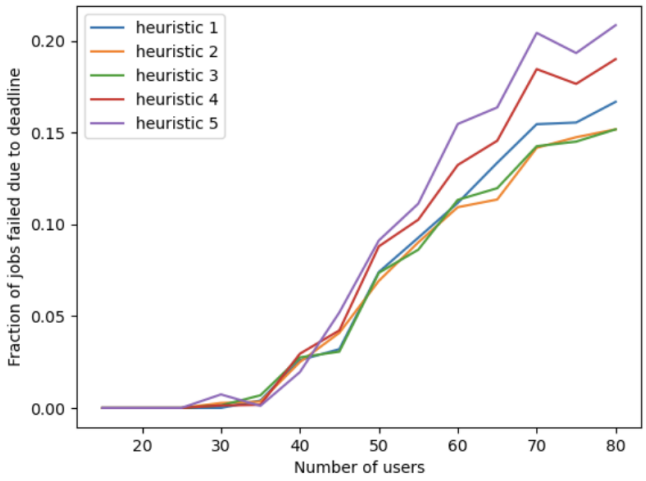

Figure 4: $K = 500$, Number of tasks per user = 50, Number of machines = 50,
This graph illustrates that as the number of users increases, there is a corresponding rise in the number of tasks directed towards the machines. Consequently, the workload on the machines intensifies, resulting in an increased incidence of job failures attributed to deadline constraints.

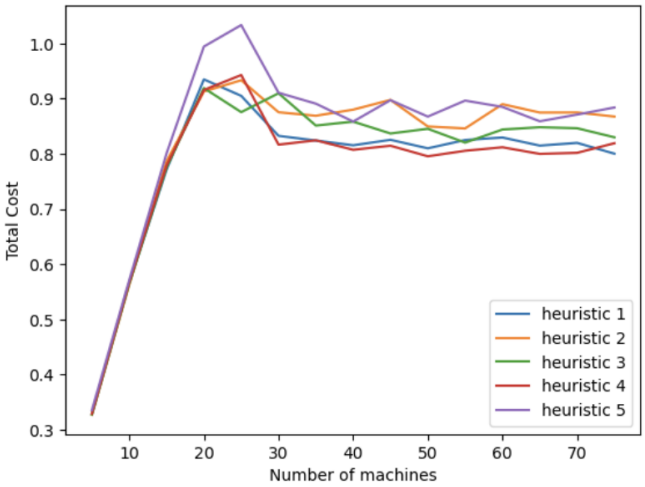

Figure 5: $K = 50$, Number of users = 80, Number of tasks per user = 1000
This graph shows that total cost first increases rapidly on increasing the number of machines then decreases slightly and then becomes constant. This trend can be attributed to the interplay between two conflicting factors: the execution cost and the number of jobs failing due to deadline constraints. Initially, as the number of machines increases, the number of jobs failing due to deadline constraints decreases, increasing the execution cost. However, the increase in the cost of execution outweighs the reduction in the number of jobs failing due to deadlines, leading to an overall rise in the total cost. After reaching a certain threshold, a slight decrease in the total cost is observed. This occurs because the term corresponding to the number of jobs failing due to deadline constraints begins to dominate over the cost of execution. Beyond this point, further increases in the number of machines do not significantly affect the total cost, as the number of machines becomes sufficient to mitigate the impact of deadline failures.

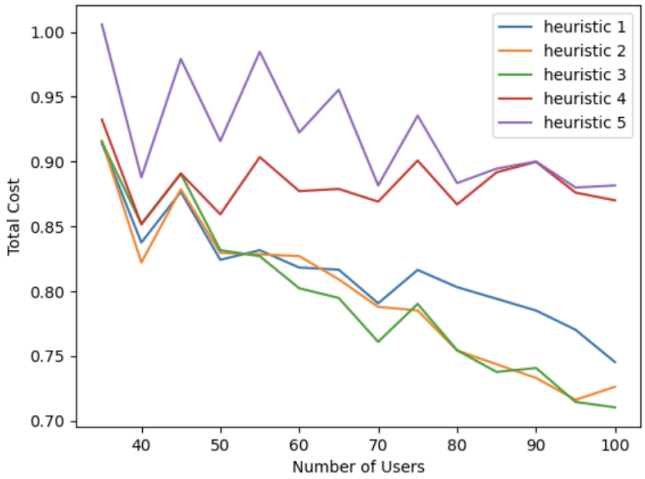

Figure 6: $K = 500$, Number of tasks per user = 50, Number of machines = 50
The observed graph indicates a consistent decrease in the average total cost across various heuristics. This trend may be ascribed to a reduction in the percentage of jobs failing to meet the deadline or reliability criteria, possibly stemming from increased flexibility in job selection. Notably, the inferior performance of heuristic 4 is attributed to its disregard for task processing costs when selecting a machine, which predominates over decrease in other terms.

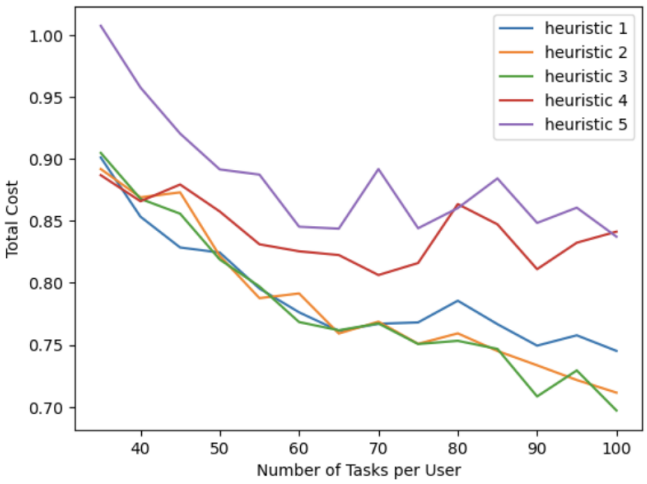

Figure 7: $K = 500$, Number of users = 50, Number of machines = 50
This graph shows a general decreasing trend in the average total cost for the heuristics as the number of tasks per user increases. This can also be attributed percentage in no of jobs missing the deadline or failing the reliability requirement to more flexibility in choice of jobs.

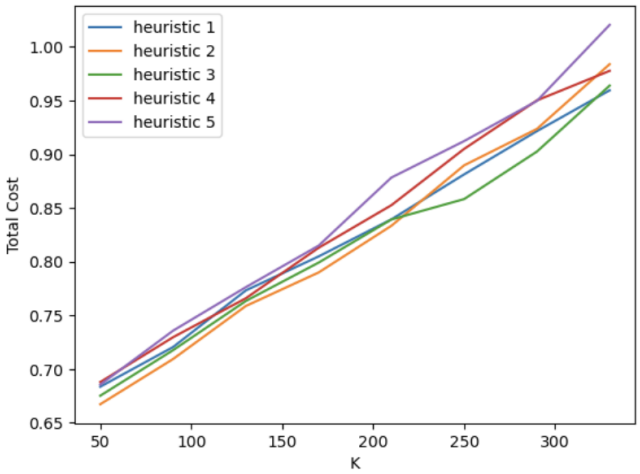

Figure 8: Number of users = 50, Number of machines = 10, Number of tasks per user = 100 The depicted graph illustrates a rise in the average total execution cost with an increase in the relative deadline parameter, K. This phenomenon occurs because as the deadline extends, while the proportion of jobs missing the deadline diminishes, the shared trust among machines escalates. Consequently, this results in a quadratic escalation in the cost of task execution, given the direct proportionality between the cost of task processing and shared trust, ultimately leading to an overall increase in cost.

8 Conclusion and Future Work

In this study, we have explored four distinct heuristics designed to address the focal question. Each heuristic offers unique advantages, providing users with a range of options tailored to their specific objectives. Our analysis reveals that heuristic 3, denoted by the color green in the graphs, consistently demonstrates commendable performance concerning our defined objective function.

Moving forward, we plan to integrate machine learning techniques to develop predictive models capable of optimizing heuristic selection based on historical performance data and real-time feedback. Furthermore, conducting field trials or simulations in real-world environments will allow us to validate the system's effectiveness and gather user feedback for further refinement.

9 References

[1] Amanjot Kaur and Nitin Auluck. Real-time trust aware scheduling in fog-cloud systems. *Concurrency and Computation: Practice and Experience*, 35(10):e7680, 2023.

[2] K. D. Le and J. T. Day. Rolling horizon method: A new optimization technique for generation expansion studies. *IEEE Transactions on Power Apparatus and Systems*, PAS-101(9):3112–3116, 1982.

[3] Anil Singh, Nitin Auluck, Omer Rana, Andrew Jones, and Surya Nepal. Scheduling real-

time security aware tasks in fog networks. *IEEE Transactions on Services Computing*, 14(6):1981–1994, 2019.

[4] Xiaomin Zhu, Huangke Chen, Laurence T Yang, and Shu Yin. Energy-aware rolling-horizon scheduling for real-time tasks in virtualized cloud data centers. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1119–1126. IEEE, 2013.