Scheduling Firm Real-time Applications on the Edge with Single-bit Execution Time Prediction

Shaik Mohammed Salman*, Van-Lan Dao¶, Alessandro Vittorio Papadopoulos‡, Saad Mubeen†, and Thomas Nolte§
*Mälardalen University,, Västerås, Sweden, shaik.mohammed.salman@mdu.se
¶Mälardalen University, Västerås, Sweden, van.lan.dao@mdu.se
‡Mälardalen University, Västerås, Sweden alessandro.papadopoulos@mdu.se
†Mälardalen University, Västerås, Sweden, saad.mubeen@mdu.se
§Mälardalen University, Västerås, Sweden, thomas.nolte@mdu.se

***Abstract*—The edge computing paradigm brings the capabilities of the cloud such as on-demand resource availability to the edge for applications with low-latency and real-time requirements. While cloud-native load balancing and scheduling algorithms strive to improve performance metrics like mean response times, real-time systems, that govern physical systems, must satisfy deadline requirements. This paper explores the potential of an edge computing architecture that utilizes the on-demand availability of computational resources to satisfy firm real-time requirements for applications with stochastic execution and inter-arrival times. As it might be difficult to know precise execution times of individual jobs prior to completion, we consider an admission policy that relies on single-bit execution time predictions for dispatching. We evaluate its performance in terms of the number of jobs that complete by their deadlines via simulations. The results indicate that the prediction-based admission policy can achieve reasonable performance for the considered settings.**

I. Introduction

Edge computing enables end-user applications to offload parts of their computations to achieve better response times and reduce energy consumption on local devices [1], [2], [3]. Some applications require the offloaded parts to be completed before a certain time for the results to be useful [4]. If the result has not been generated within this time, the computations (or jobs) can be abandoned. We refer to these types of applications as having firm real-time requirements [5]. To satisfy firm real-time requirements, it may be necessary to ensure that sufficient servers have been provisioned such that some performance metric, such as the average number of missed deadlines or the number of completed jobs, is below or above some threshold, respectively. In embedded settings, such resource provisioning is based on worst-case conditions such as worst-case execution times and minimum inter-arrival times [6]. While this approach can satisfy the timing requirements, it comes at the cost of inefficient resource usage. Provisioning based on average-case conditions, such as mean execution times and inter-arrival times, can achieve better resource usage, but we cannot guarantee performance deterministically [7], [8]. In this paper we investigate a potential approach to manage the conflicting requirements of efficiency and performance by considering an edge computing model where we provision a set of servers based on average-case conditions, while on-demand servers are provisioned to address transient and occasional worst-case conditions.

The scheduling strategies employed in such a computing model play a critical role in determining achievable performance. When aiming to minimize mean response times in single-server settings, scheduling strategies that are aware of the execution times of offloaded jobs, such as the shortest remaining processing time (SRPT), outperform strategies like first-in-first-out (FIFO) scheduling [9]. For multi-server settings where jobs are dispatched to specific servers upon arrival, the combination of SRPT and dispatching policies like join-shortest-queue (JSQ) are known to offer better performance [10]. However, in certain scenarios, execution times can only be known post-completion. To address this, several studies have proposed using predicted execution times in the absence of prior knowledge of true execution times [11], [12], [13], [14], [15]. Particularly in the context of queuing systems and aiming to improve the performance metric of mean response times, Mitzenmacher [15] investigated the effectiveness of single-bit predictors that can determine whether a job's execution time is above or below a certain threshold. The study demonstrated that such predictors can offer benefits similar to those obtained with precise knowledge of execution times.

In addition to scheduling, load balancing or dispatching strategies can also affect performance. In terms of mean response times, load balancing techniques such as JSQ outperform strategies such as Round-Robin (RR) [16]. This benefit comes at the cost of increased overheads as JSQ policy requires the knowledge of pending jobs in each of the servers to identify the server with the least number of pending requests. For applications with response times in a few tens of milliseconds range, these overheads may be unacceptable and a low-complexity dispatching policy such as RR may be preferable. For applications with firm real-time requirements, in addition to removing jobs that can potentially miss their deadlines when already in the queue [4], it may be beneficial to admit only those jobs that can be estimated to finish by their deadlines while rejecting jobs that are most likely to miss their deadlines given the current pending jobs at the

979-8-3503-3902-4/23/$31.00 ©2023 IEEE

servers. This approach allows future jobs with possibly lower execution times and encountering reduced pending workload to be completed within deadlines, thereby improving the throughput.

Within this context, we consider applications with offloadable jobs whose execution times are given by a probability distribution and should be completed by a fixed relative deadline. Given the difficulty in knowing precise execution times of individual jobs, we assume the presence of a single-bit predictor[1] that can indicate if a job is a short job or a long a job and investigate the performance in terms of throughput, i.e, number of jobs that complete by their deadlines. The jobs are admitted or rejected on arrival based on an admission policy that estimates response time of the incoming job using the information from the single-bit predictor and dispatched following the JSQ or RR strategy with all admitted jobs scheduled in FIFO order. Furthermore, due to strong impossibility results in terms of competitive analysis, jobs must contain some, (possibly large) amount of slack [17], [18]. We include this in our work by setting the relative deadline to be ten times the mean value of the execution time distribution. We use the value ten as we target applications with similar characteristics where the ratio between their worst-case execution times and the relative deadlines is large.

Concretely, we investigate the performance via simulations by considering exponential execution time distribution and Poisson arrivals. As we consider only the edge layer, we limit our simulations to non-asymptotic conditions. Additionally, we compare the performance of the prediction-based policy to a clairvoyant policy that has complete knowledge of the execution time of each pending as well as newly arriving job, as well as a policy that estimates the execution time of each job as the mean of its distribution. Our findings suggest that the prediction-based policy performs better than the mean-approximation policy, while being only slightly inferior to the clairvoyant policy when on-demand servers are available.

II. Background & Related Work

A. Edge Computing and On-Demand Servers

Edge computing architectures extend the concepts and benefits of cloud computing such as the provisioning of additional computing capabilities depending on the current workload for applications with latency and predictability requirements. Several auto-scaling strategies have been proposed that include vertical scaling where the amount of CPU time, (for example, in containerized deployments where a server is sharing a single CPU with other servers) allocated for a given application is increased or decreased, and horizontal scaling where the number of allocated servers is increased or decreased depending on average workload monitored over some time window [19]. In contrast, our edge-computing model is designed to address instantaneous load changes and assumes that all necessary setup for executing a job on on-demand servers is done during an initialization stage. We note that auto-scaling for sustained changes in workload can be easily incorporated into our model by changing the number of reserved servers. Close to the work presented in this paper, Wang et al. [20] provided a load balancer and core allocation strategy to minimize mean response times for non-preemptive FIFO by considering heterogeneous servers with reserved and on-demand servers within a cluster. Here the on-demand servers are utilized when the queue is full or when the waiting time exceeds the maximum waiting time. In our work, jobs are dispatched to on-demand servers based on the estimated response times. A distinguishing and reasonable assumption for edge computing is that the number of available computing nodes is much less compared to those of large data centers that make up the cloud infrastructure. As a consequence of this assumption, we restrict our analysis to small-sized clusters. In addition to this, we assume that the number of on-demand servers is also limited and known in advance. Fig. 1 depicts an example architecture with two reserved and two on-demand servers. Additionally, we assume that not all of the on-demand servers are available at all times for a specific application, as these on-demand servers may be shared among multiple applications. The availability is explicitly considered, and we model it as a Bernoulli distribution.

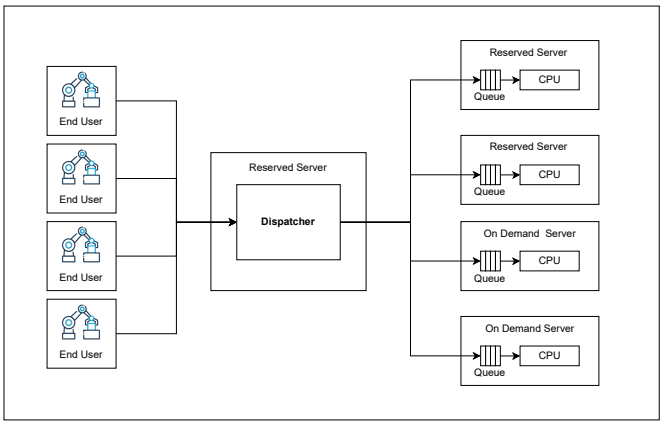

Fig. 1. System Architecture

B. Dispatching Policies

In multi-server environments, dispatching policies determine the server on which an incoming job will be executed. In the online dispatching problem that we are considering, the dispatching decision is made when the job arrives at the dispatching server. Dispatching policies, such as joining the shortest queue and its variants, such as the shortest queue among k randomly chosen servers, require knowledge of the exact number of pending jobs in each of the considered servers [9]. Gathering this information may take a significant amount of time, depending on the number of servers and the network traffic [16]. As an alternative, policies such as round-robin are agnostic to pending jobs on the servers and dispatch incoming jobs to the servers in a repeating pattern. The advantage of

[1]Although single-bit prediction is coarse, it may be easier to realize them with better accuracy compared to fine-grained predictions.

policies such as round-robin is that they do not have the overhead associated with policies that require information about the pending workload. In most existing work, the objective of dispatching policies has been to minimize mean response times. Several additional policies, such as join-the-idle-queue and join-below-threshold, where servers notify the dispatchers when they are idle or have pending jobs less than a predefined value, have been proposed to balance the trade-off between overheads and response times [6], [16]. In this paper, we aim to evaluate the performance of the prediction-based admission policy in terms of achievable throughput and consider JSQ and RR as representative dispatching policies. We augment them with admission policies with the intuition that rejecting jobs that are unlikely to meet their deadlines can reduce the amount of time servers spend doing unuseful work.

C. Execution Time Predictions

Several works have investigated the possibility of improving the performance of algorithms with machine-learned advice or predictions including classical algorithms targeting problems such as online scheduling and load-balancing [11]. The evaluation of these prediction augmented algorithms has been done in terms of competitive analysis under accurate predictions and for possibly incorrect predictions [15], [21], [22], [6]. For the online scheduling problem, some of the authors have considered predicting parameters such as job execution times [6], [13] and permutation ordering of jobs [22]. Mitzenmacher [15] studied the impact of single-bit predictors that can indicate if a job's execution time is above or below some threshold in the context of large-scale queuing systems for the performance metric of mean response times and showed that such predictors can provide benefits similar to those achievable with the knowledge of exact execution times for Poisson arrivals and certain execution time distributions. The analysis included the impact of incorrect predictions and highlighted the improvements achieved even with such incorrect predictions against the policy of choosing a queue with the least number of pending jobs. Similarly, they extended their analysis in [21] for the case where individual execution times were also predicted and showed via simulations that the benefits of the *supermarket model* in large distributed systems were retained if the predictions were *reasonably precise*. Based on the evaluations, they proposed the shortest queue selection and predicted shorted processing job first policy for use in actual systems, as it performed well in a diverse range of scenarios. In a similar context, Zhao et al.[23] extended the randomized multi-level feedback algorithm (RMLF) that makes no assumption on job execution times with predicted job execution times to minimize mean response times. Their experimental evaluation shows that the prediction-based algorithm achieves performance close to that of SRPT when the prediction error is small. If the error is large, their algorithm can achieve better performance than RMLF. An important requirement for such prediction-based enhancements is that the predictions should be learnable in practice. Keeping this in mind, we limit our attention in this paper to single-bit predictors that can identify jobs that can take a long time to execute and those that take a shorter duration. We use this information in our admission policy that estimates the response time of a new job and does a schedulability test using this response time. The work in our paper is inspired by the findings of these studies and is extended in the context of deadline constraints in terms of performance criteria while being restricted to single-bit predictions, FIFO order, and non-asymptotic conditions with probabilistic availability of on-demand servers.

Scheduling Firm Real-time Tasks: Gao et al. [4] proposed scheduling strategies for firm semi-periodic real-time tasks in single-server settings, where jobs are released periodically and have the same relative deadline, but execution times have an arbitrary probability distribution. They investigated several optimization criteria, including the Deadline Miss Ratio (DMR). They introduced three new control parameters to decide at run-time whether to interrupt a job before its deadline. The parameters include (i) an upper bound on completion times, based on which a job is dropped if it is not completed by this time. This bound is a value between periodic inter-arrival time and relative deadline, (ii) an upper bound on job execution times, based on which jobs with execution times exceeding this value are rejected, and (iii) an upper bound on waiting time based on which a job that has waited until this bound will be dropped. In addition to this, they considered four admission policies which include (i) admitting all jobs, (ii) admitting jobs until a fixed number of jobs are in the queue, (iii) admitting jobs with some fixed probability and (iv) admitting jobs following a repeating pattern. Their evaluation shows that the key control parameter is the upper bound on the waiting time of each job achieving the best DMR. In comparison to this work, our work uses admission policies that estimate the response times based on the job execution time distribution and the number of pending jobs on a specific server while letting admitted jobs stay in the queue until their completion or until their deadline.

III. System Model

We now describe our system model and our assumptions in detail.

a) Specifying System Load: In queuing systems, it is essential that job arrival rates be less than departure rates to avoid queue build-up over time. For applications with execution time variability, a system designer has the option to consider inter-arrival times proportional to worst-case execution time or some value between the worst-case value and the mean of the execution time distribution. If the arrival times are proportional to worst-case values, the total number of jobs released over a fixed duration of time can be much less than when the arrival times are close to the mean values. Since we consider applications with firm real-time requirements, we define system load such that the arrival times are proportional to mean execution times rather than worst-case values. Our reasoning is based on the intuitive idea that even though executing a large number of jobs can result in more of the jobs missing their deadlines, the number of jobs that complete by their deadlines can be

larger than the number of jobs released when the arrival times are set to values proportional to worst-case execution times. This higher number of completed jobs can provide better functional performance than the scenario where no jobs miss their deadlines but only fewer jobs are released. Therefore, we model system load such that the average arrival rate is proportional to the mean of the execution time distribution and scales accordingly to the different number of reserved servers. Because we define system load based on average case parameters, there may be a situation where incoming jobs over a short duration of time take longer time to execute resulting in temporary overload. To manage this temporary overload, we consider the availability of on-demand computing resources within the edge layer.

b) Job Model: We assume that jobs have an execution time distribution with a known mean value μ. The exact execution time of a job remains unknown until its completion. All arriving jobs have a fixed relative deadline D. The jobs have a Poisson arrival process with a constant arrival rate λ. Whenever a job arrives, a dispatcher should decide if the job will be admitted or rejected. If admitted, each job remains in the server queue until completion or until its deadline. We assume that the fixed relative deadline is such that there is some amount of slack l with respect to the mean value μ. In this paper, we set l equal to ten times the mean μ. We set the system load proportional to the number of reserved servers. A job is said to be schedulable if its estimated response time is less than or equal to its relative deadline and unschedulable otherwise.

c) Server Model: We assume a cluster of homogeneous servers divided into a set of reserved servers R and a set of on-demand servers S. Each reserved server has its own queue and executes the jobs of a single application. Each admitted job is added to the queue of one of the servers on its arrival. Once assigned, the job stays in this queue until its completion. All admitted jobs are sequenced in FIFO order in each server queue and are executed non-preemptively. For the on-demand servers, each server can execute jobs for multiple applications and has a separate queue for each application. The server is considered available for a specific application if it meets one of the following criteria: (i) it is idle, (ii) it is executing jobs of the same application and has no pending jobs from higher-priority applications, or (iii) it is executing jobs of a lower-priority application. If the server does not meet any of these criteria, it is considered unavailable for the application. In our simulations, we model the availability of the on-demand servers using a Bernoulli distribution. Although this availability model is simple, it enables a straight-forward quantitative comparison of the different admission and dispatching policy combinations.

IV. Admission Policies and Dispatching

In our on-arrival dispatching model, an admission policy determines whether a job should be accepted or rejected. In this work, we consider three admission policies that estimate the response time of an incoming job by taking into account the pending jobs on a given server (see Section IV-A). Admission or rejection can be done in two steps. In the first step, the dispatcher looks for a server within the set of reserved servers. If it fails to admit a job onto a reserved server, it searches for an available on-demand server and checks for schedulability. A job is admitted if the policy considers it to be schedulable on a reserved server, and then it is sent to that server by the dispatcher. If a job is unschedulable on a reserved server, the dispatcher tries to find an available on-demand server and tests the schedulability of the job on this on-demand server. It dispatches the job to it if it is schedulable. Otherwise, the job is rejected. Algorithm 1 describes our dispatching process, which takes as input the identifiers of the reserved servers and on-demand servers, the dispatching policy, the relative deadline, and the specific admission policy.

If the configured dispatching policy is JSQ, the dispatcher selects the server with the shortest queue among the reserved servers. If the dispatching policy is RR it cyclically selects a server. When a new job arrives and a reserved server has been identified, the dispatcher uses one of the response time estimators and checks if the estimated response time is less than or equal to the relative deadline. If so, the job is sent to the identified server (lines 9-11). If the response time estimate

Algorithm 1 Dispatcher

```
1:  Input: Incoming job J, set of reserved servers R and set of on demand servers S, dispatching policy P, deadline D, response time estimation policy A
2:  Output: The id of server if job schedulable, NULL otherwise.
3:  function GETBESTSERVER(J, R, S, P, D)
4:      if P == JSQ then
5:          id ← get_shortest_queue_server(R)
6:      else if P == RR then
7:          id ← get_next_rr_server(R)
8:      end if
9:      f_i ← get_estimated_response_time(id, J, A)
10:     if f_i ≤ D then
11:         return id                ▷ job deemed to be schedulable
12:     else if f_i > D then
13:         S_x ← get_available_on_demand_servers(S)
14:         if P == JSQ then
15:             id ← get_shortest_queue_server(S_x)
16:         else if P == RR then
17:             id ← get_next_rr_server(S_x)
18:         end if
19:         f_i ← get_estimated_response_time(id, J, A)
20:         if f_i ≤ D then
21:             return id            ▷ job deemed to be schedulable
22:         else
23:             return NULL          ▷ job deemed to be unschedulable
24:         end if
25:     end if
26: end function
```

is not within the deadline, the dispatcher identifies the subset of available on-demand servers and selects a server according to the configured policy. The dispatcher then uses one of the response time estimators and checks if the estimated response is less than or equal to the relative deadline. If so, the job is sent to the identified on-demand server (lines 19-21). If the estimated response time is not within the deadline, the job is rejected.

A. Response Time Estimation

We now describe the response time estimation policies used to admit or reject the jobs. We consider three policies based on how the job execution times are considered, (i) mean-approximation policy, (ii) clairvoyant policy, and (iii) single-bit prediction policy.

a) Mean-approximation policy: In this policy, we use mean μ of the execution time distribution to estimate the response time f_i on the selected server i. If the number of pending jobs on this server is given by N_i, the estimated response time is given by

$$f_i = (N_i + 1) \cdot \mu. \tag{1}$$

We consider the mean-approximation policy because of its low computational overhead.

b) Clairvoyant policy: In this policy, we assume the knowledge of exact execution times. The response time f_i on any server i is given by

$$f_i = x_j + \sum_{k=0}^{N_i} x_k, \tag{2}$$

where x_k is the exact execution time of each job k assigned to server i and x_j is the execution time of the newly arrived job.

c) Single-bit prediction policy : In this policy, we assume that there exists a predictor which can indicate if a job is a short job or a long job. A job is said to be a short job if its true execution time is less than μ and long otherwise. The response time f_i of a job on any server i where N_i^s is the number of pending jobs classified as short and N_i^l is the number of pending jobs classified as long is given by

$$f_i = (N_i^s \cdot \mu_s) + (N_i^l \cdot \mu_l) + (\tau \cdot \mu_s + (1 - \tau) \cdot \mu_l), \tag{3}$$

where μ_s and μ_l are specified by the designer and τ is the output of predictor indicating if the new job is a short job or a long job. We assume that a server can identify both N_i^s and N_i^l and make this information available to the dispatcher.

For all of the estimation methods, the admission test returns true if the following condition is satisfied:

$$f_i \leq D. \tag{4}$$

V. Evaluation

We use simulation to evaluate the performance of the prediction-based policy in terms of throughput. We consider throughput as the ratio of jobs that completed before or at their deadlines and the total number of jobs that arrived during the simulation interval. We set the simulation interval to ten thousand time units and take the average of the measured throughput for ten simulation runs. We generated the inter-arrival times of the jobs and the execution time distributions using the exponential distribution class of the C++ library. We set the mean of execution time distribution μ equal to ten and the arrival rate as proportional to the inverse of μ. Each generated job is assigned a deadline equal to ten times the mean of the distribution. We set μ_s equal to five and μ_l equal to fifteen. The number of reserved servers was varied between two, four, and eight but we only present the results for the scenario where the number of servers was set to eight. Similarly, we set the number of on-demand servers equal to the number of reserved servers. We make the assumption that there are no overheads involved in obtaining the state of the queues from the servers.

A. Performance of prediction-based admission policy

We compare the performance of the admission policy that uses the information provided by the single-bit predictor against a baseline solution that uses the exact execution time information and another solution that uses the mean of the service time distribution to estimate the response times of newly arriving jobs before deciding to admit or reject the jobs. We evaluate this performance for varying loads, dispatching policies, and availability of on-demand servers.

As seen in Fig. 2 when only using the reserved servers, the difference between the throughput achieved by the prediction-based policy and the clairvoyant policy becomes more apparent as the load is increased. This behavior is observed for both dispatching policies. When on-demand servers are always available, the observations remain consistent. The benefits of the prediction-based policy are more evident when compared to the mean-approximation policy at higher loads and always available on-demand servers.

> Prediction-based admission policy provides better throughput performance compared to mean-approximation policy while the clairvoyant policy provides the best throughput among all the considered policies.

B. Performance of dispatching policies

Our evaluation indicates that both JSQ and RR have similar throughput performance for loads greater than one when only using reserved servers for identical admission policies. This indicates that the dispatching policy has very little impact on the throughput compared to the impact of the response time estimation policies. When on-demand servers are considered to be always available, the performance difference remains negligible as seen in Fig. 2 and Fig. 3.

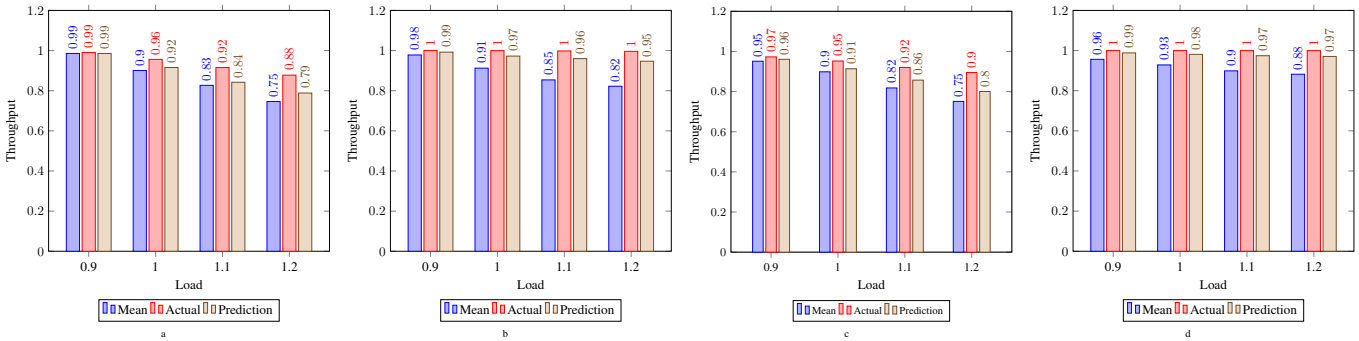

Fig. 2. Throughput of various admission tests and load for exponentially distributed service times for (a) JSQ, (b) RR, (c) JSQ with on-demand servers, and (d) RR with on-demand servers.

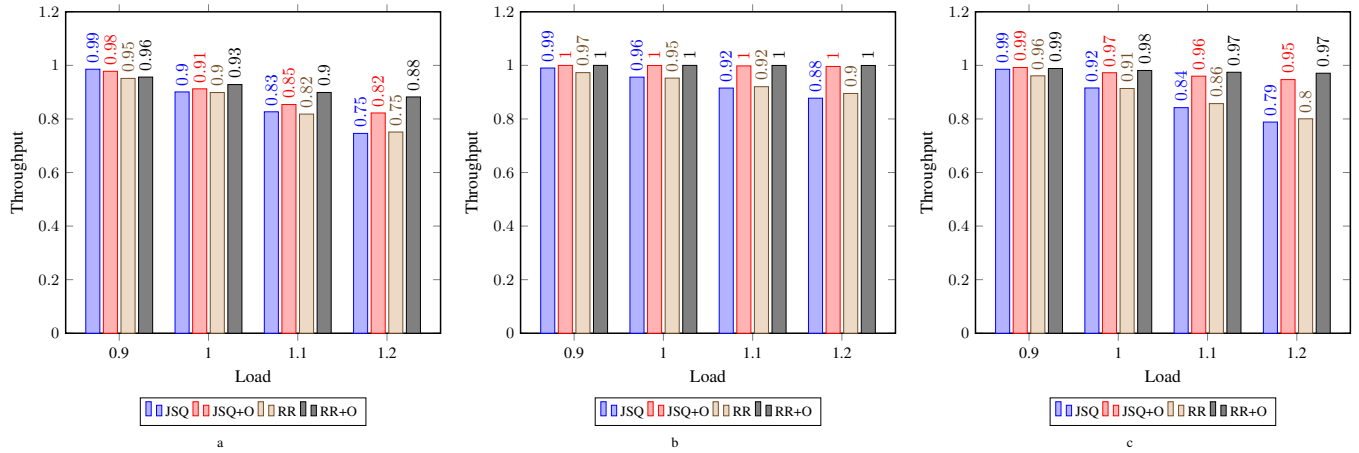

Fig. 3. Throughput under various load conditions when using on-demand servers in addition to reserved servers for admission policies with (a) mean (b) exact execution time (c) Single-bit job type prediction

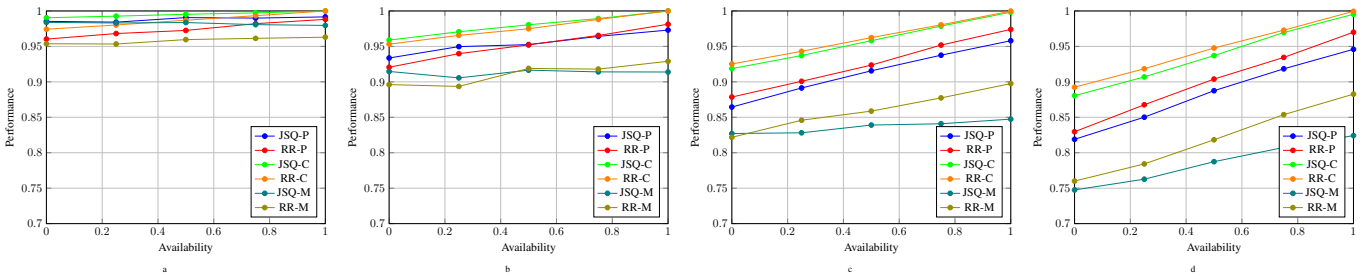

Fig. 4. Throughput for varying availability probability of on-demand servers when (a) Load = 0.9, (b) Load = 1, (c) Load = 1.1, and (d) Load =1.2

Both JSQ and RR have similar performance when using identical admission policies on reserved servers. When on-demand servers are always available, RR achieves similar average throughput compared to JSQ.

C. Performance impact of availability of on-demand servers

We investigate the impact of the availability of on-demand servers on the achievable throughput for different admission policies and dispatching combinations. As seen in Fig. 3 When the load is close to 0.9, the impact of on-demand server availability is negligible for all the admission and dispatching policies. However, when the load is increased, the availability of on-demand servers provides considerable improvement in throughput. Clairvoyant policy dominates the performance for all of the availability values. Increasing availability results in improved performance for all of the admission policies as seen in Fig. 4. Additionally, combining the prediction-based policy with either of the dispatching policies outperforms the mean-approximation policy while closely following the clairvoyant policy.

Increased availability of on-demand servers improves throughput significantly under overload conditions for all admission policies compared to using only reserved servers.

VI. Conclusion

We have studied the performance of a single-bit prediction based admission policy for the problem of online dispatching and scheduling of jobs with stochastic execution and inter-arrival times, along with deadline constraints for firm real-time systems in a multi-server edge computing environment. Using simulations, we evaluated and compared the performance of the prediction based policy against the mean-approximation and the clairvoyant admission policy with two well-known dispatching strategies, JSQ and RR. We have also considered an architecture that provides access to on-demand servers in addition to a set of reserved servers. Our results indicate that the achievable throughput is primarily influenced by the estimation accuracy of the admission policy and availability of on-demand servers, rather than the dispatching policy. In addition, the single-bit prediction policy outperformed mean-approximation policy while falling short of the performance of the clairvoyant policy.

Acknowledgement

The research leading to these results has received funding from the Knowledge Foundation (KKS), under the projects FIESTA (Project No. 20190034) and SACSys (Project No. 20190021), and under the Swedish Research Council (VR), under the project PSI (Project No. 2020-05094).

References

[1] J. Zilic, A. Aral, and I. Brandic, "Efpo: Energy efficient and failure predictive edge offloading," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, pp. 165–175.

[2] J. Zilic, V. De Maio, A. Aral, and I. Brandic, "Edge offloading for microservice architectures," in *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–6. [Online]. Available: https://doi.org/10.1145/3517206.3526266

[3] M. Jansen, A. Al-Dulaimy, A. V. Papadopoulos, A. Trivedi, and A. Iosup, "The spec-rg reference architecture for the edge continuum," 2022. [Online]. Available: https://arxiv.org/abs/2207.04159

[4] Y. Gao, G. Pallez, Y. Robert, and F. Vivien, "Dynamic scheduling strategies for firm semi-periodic real-time tasks," *IEEE Transactions on Computers*, vol. 72, pp. 55–68, 1 2023.

[5] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[6] Y. Zhao, R. Zhou, and H. Zeng, "Design optimization for real-time systems with sustainable schedulability analysis," *Real-Time Systems*, vol. 58, no. 3, pp. 275–312, 2022.

[7] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.-S. Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proceedings real-time technology and applications symposium*. IEEE, 1995, pp. 164–173.

[8] J. Diaz, D. Garcia, K. Kim, C.-G. Lee, L. Lo Bello, J. Lopez, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, 2002, pp. 289–300.

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

[10] I. Grosof, Z. Scully, and M. Harchol-Balter, "Load balancing guardrails: Keeping your heavy traffic on the road to low response times (invited paper)," in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 10. [Online]. Available: https://doi.org/10.1145/3406325.3465359

[11] M. Purohit, Z. Svitkina, and R. Kumar, "Improving online algorithms via ml predictions," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[12] Y. Azar, S. Leonardi, and N. Touitou, "Flow time scheduling with uncertain processing time," in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1070–1080. [Online]. Available: https://doi.org/10.1145/3406325.3451023

[13] Z. Scully, I. Grosof, and M. Mitzenmacher, "Uniform bounds for scheduling with job size estimates," *arXiv preprint arXiv:2110.00633*, 2021.

[14] M. Akbari-Moghaddam and D. G. Down, "Seh: Size estimate hedging scheduling of queues," *ACM Transactions on Modeling and Computer Simulation*, 2023.

[15] M. Mitzenmacher, "Queues with small advice," in *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*. SIAM, 2021, pp. 1–12.

[16] X. Zhou, F. Wu, J. Tan, Y. Sun, and N. Shroff, "Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–30, 2017.

[17] F. Eberle, N. Megow, and K. Schewior, "Optimally handling commitment issues in online throughput maximization," in *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, ser. LIPIcs, F. Grandoni, G. Herman, and P. Sanders, Eds., vol. 173. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 41:1–41:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2020.41

[18] ——, "Online throughput maximization on unrelated machines: Commitment is no burden." *ACM Transactions on Algorithms*, 12 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3569582

[19] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Comput. Surv.*, vol. 51, no. 4, jul 2018. [Online]. Available: https://doi.org/10.1145/3148149

[20] S. Wang, X. Li, Q. Z. Sheng, R. Ruiz, J. Zhang, and A. Beheshti, "Multi-queue request scheduling for profit maximization in iaas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2838–2851, 2021.

[21] M. Mitzenmacher and M. Dell'Amico, "The supermarket model with known and predicted service times," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 2740–2751, 11 2022.

[22] A. Lindermayr and N. Megow, "Permutation predictions for non-clairvoyant scheduling," in *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 357–368. [Online]. Available: https://doi.org/10.1145/3490148.3538579

[23] T. Zhao, W. Li, and A. Y. Zomaya, "Real-time scheduling with predictions," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 331–343.