Optimal Control of Join Decision in Fork-Join Queues

Akhil Singla[1] and Seyed Iravani[1]

[1]Department of Industrial Engineering and Management Sciences, Northwestern University

November 2023

1 Abstract

In fork-join (FJ) queues, each incoming job to the system is copied and sent to all its parallel servers. A job is completed when *all* or *some* of the copies are processed by servers. These FJ queues are prevalent in diverse domains of service operations like fact-checking for social media platforms, hospital diagnosis operations, legal systems etc. FJ queueing systems can capture the trade-off between speed and accuracy that commonly arises in knowledge-based service systems in which processing a job also involves evaluation of knowledge servers leading to a decision. While literature on performance analysis of FJ queues is large, the optimal control of FJ queues are not well studied. In this paper, we study the optimal control of "join decision" in FJ queues. A join decision determines how many copies of a job should be processed before the job leaves the system. We fully characterize the structure of the optimal dynamic join decision policy with respect to the parameters that affect the trade-off between speed and accuracy. To overcome the curse of dimensionality, we propose an intuitive, easy-to-compute and easy-to-implement heuristic policy that uses total number of jobs in the system to make join decision. This heuristic is shown to perform within 2% optimality gap and performs better than some commonly used benchmark policies. The heuristic policy also allows managers to achieve a desired service levels that balances speed and accuracy of the operations.

Keywords: *Speed and accuracy trade-off, fact-checking, fork-join queues, MDP, knowledge servers, optimal control*

2 Introduction

In knowledge-based service systems (KBSS), jobs involve making decision by servers of different knowledge levels. We refer to these servers as "knowledge servers." The decisions made by these knowledge servers

have a significant impact on the routing of jobs within the service system, influencing both the speed and accuracy of processing a job. For instance, in the context of patient triage, a knowledge server, such as a nurse, may defer the completion decision (treatment) of an incoming job (patient) to a more experienced knowledge server, such as a telemedical physician. This delegation aims to improve the accuracy of the job completion decision (treatment) while potentially compromising on the speed of service (patient's waiting time).

The primary performance measures in knowledge-based service systems revolves around two key aspects: the waiting time of jobs in the system, which reflects the system's speed of service, and the accuracy of job completion decisions. In service operations management, the trade-off between speed and accuracy is controlled through several mechanisms, including the following:

- Server Speed: In some systems, the knowledge server has discretion to spend more time on a job to increase its accuracy. This results in delay of service for all jobs and thus reduces overall speed of service (e.g., Alizamir et al. (2022), Chan et al. (2014), Hopp et al. (2007)).

- Process Repetition: In certain service systems, the decision maker can increase the quality (e.g., accuracy) of service by repeating job processing at the cost of delaying service to other jobs. An example is shown in Alizamir et al. (2013) in which a series of imperfect diagnostic tests are performed before making a final diagnosis decision.

- Job Routing: In multi-stage processes, a job can be routed to different knowledge servers to improve the accuracy of job completion decision. This, however, leads to congestion in the system. For example, in Saghafian et al. (2018) with two level of knowledge servers, a lower level server can route a job to an upper level server to increase accuracy at the cost of increasing delay for the job, or make a decision at the lower level resulting in lower accuracy but smaller delay.

- Redundancy: In system with job redundancy, while the jobs can be processed (e.g., evaluated) and a decision can be made by a single server, jobs are copied and sent to multiple servers. A job completion decision is made after some of the servers (not all of them) finish processing their copies. Higher accuracy is achieved by waiting to receive evaluations from larger number of servers, which leads to more delays (i.e., lower speed). The underlying process for such a redundancy is a fork-join queueing system where each job is copied to multiple parallel servers. The job completion decision is made by a server, which we call "join server," after service completions of copies by one or more number of parallel servers (Joshi et al. (2017), Gardner et al. (2017), Liu et al. (2022)).

The focus of our study is on the speed-accuracy trade-off in service systems with redundancy created by

using additional knowledge servers in the context of fork-join queues. We focus on the following question: In order to strike the balance between speed and accuracy in fork-join queues, how do the join server make job completion decisions?

A real world example of this setting can be observed in social media platforms (SMP) that employ fork-join queues for detecting fake news through third-party fact-checking (Isola (2018), Lyons (2018)). As an example, lets assume that SMP sends copies of a posted news item to, say, 5 fact-checkers. Each of these fact-checkers evaluates the news item copy in a first-come-first-serve (FCFS) basis and provides its evaluation on whether the news item is true or false. Once SMP receives reports from fact-checkers, it decides whether the news item is true or false. While SMP work with a large network of fact-checkers to identify misinformation, the fact-checking process can take hours or even days, with an average around 5 days (Avaaz (2020)), during which news spreads at exponential speed through social media. To increase the speed of fact-checking, SMP (the join server) may make decision by waiting for some (not all) reports. SMP decides whether to wait for another evaluation report from the remaining fact-checkers for better accuracy, or to make a decision whether the news item is true or false (to reduce delay).

Indeed, the application of fork-join queues extend beyond social media platforms and encompasses a wide array of fields, each highlighting the significance of redundancy and the speed-accuracy trade-off in decision-making processes. One notable application of fork-join queues is in hospital emergency units (Carmeli et al. (2023)). In this context, during a patient's treatment at the emergency unit, the doctor conducts multiple tests, such as X-rays, blood sample tests, urine tests, CT scans, or ultrasound, in parallel. The doctor responsible for the patient's care can make a decision to continue treatment when the test results are available. The doctor could improve on speed of patient's treatment by waiting for only necessary test results (O'Sullivan et al. (2018)). Here, the doctor acts as the join server making the prediction decision for treatment of the patient.

Moreover, FJ queues are also applicable in legal systems, where judges may assign more than one expert to provide their opinions for a ruling (Canale (2021)). This practice involves the join server, a judge, to wait for evaluations from experts before making final judgement for the case. A judge can potentially wait for all or some of the opinions from the experts (Kelly (2022)).

What makes fork-join (FJ) queues different from parallel server queues? In parallel queues an arriving job is sent to only one server. In FJ queues, however, the same arriving job is copied and sent to the queues of all parallel servers. More importantly, in FJ queues, a job is completed when only m out of N parallel servers finish evaluating their copies, and the remaining $N-m$ copies are removed from the queues of $N-m$ servers. The job replication upon arrival and the removal of unprocessed jobs from parallel servers make fork-join queues challenging to analyze compared to parallel queues (Özkan (2022)).

The complex dynamics of FJ queues poses two important control decision questions: (1) For the job completion decision, how many service completions (i.e., how many evaluations out of N parallel knowledge servers) are required to achieve a desired speed and accuracy trade-off? and (2) what should the final (prediction) decision be for a job, while using all or some of the parallel servers' evaluations? In the context of social media platform's fact-checking, SMP may need to decide whether to predict the news item as true or false based on, for example, 2 out of 5 fact-checker evaluations, or to seek better accuracy by waiting for other evaluations from the remaining 3 fact-checkers. The job completion decision corresponds to the join decision, i.e., when the join server makes a decision on a job using some or all of the parallel servers' evaluations and removes the job from the system. The removal of a job involves removal of its un-evaluated copies in the parallel server queues and evaluated copies available with the join server.

Fork-Join Join Decision Model:

While the decision of waiting for further evaluations from parallel servers (who are yet to process their copies) affects the speed of service, the accuracy depends on the knowledge of parallel servers who have already given their evaluations. We study this speed-accuracy trade-off in our "fork-join decision model." In this model, each incoming job is copied to all N parallel knowledge servers and the join decision is taken by the join server after m evaluations ($m \leq N$) received from N parallel servers. In order to balance both speed and accuracy objectives, we find the structure of optimal control policy for the join server. We, particularly, solve for an optimal dynamic join decision policy, which decides if it is optimal to make final prediction of a job with only m evaluations already received from parallel servers, or to wait for any of the remaining $N-m$ servers to provide evaluation. Using information about the number of copies waiting in parallel server queues and the evaluations passed on to the join server, our proposed Markov decision process fully characterizes the structure of optimal dynamic join decision (DJD) policy.

To answer the prediction decision question, we characterize the optimal prediction decision when there are enough evaluations with the join server. Using the structure of optimal DJD policy, we develop an easy-to-compute and easy-to-implement heuristic policy to achieve results close to that of the optimal DJD policy. The heuristic policy is based only on the number of jobs waiting in the system (regardless of how jobs are allocated among the servers). It recommends "waiting" for join decision when the number of jobs in the system is below a certain threshold. We approximate these thresholds based on a queueing model. The heuristic shows that the benefit of using a dynamic policy over other static benchmark policies. We further show the use of heuristic policy to find control policies to achieve a desired trade-off.

The flow of the paper is as follows: Section 3 reviews the literature, and Section 4 formalizes our model as a fork-join model and characterizes the optimal DJD policy. In Sections 5, the need for an intuitive

heuristic is motivated followed by its details in Section 6. Section 7 describes the numerical study to validate performance of the heuristic and its benefits. Section 8 concludes the paper and explores different directions for further investigations.

3 Literature Review

This study has overlap with the literature on knowledge-based service systems with speed-accuracy trade-off as well as literature on fork-join queues. Below we present a brief discussion on each of them to highlight the contribution of our paper.

There are many studies in operations management literature that have combined control of jobs in a queue and knowledge server decision making. We group those studies based on the mechanism they use to achieve the trade-off between speed and accuracy.

Server Speed: The study Delasay et al. (2019) provides detailed discussion on how decision makers adjust single-server service speeds in response to congestion in the system. In KBSS, however, another important factor is how evaluation of a server affects the accuracy (or quality) of the job completion.

Hopp et al. (2007) analyze the discretionary task completion in a simple single or two server queue. The study finds optimal control policy to determine how much time to allocate to customers. Alizamir et al. (2022) studies the sequential search problem in a multi-stage single-server queueing setting, in which the server decides between completing the job in a stage or to continue its service while sampling for more information to make accurate decision. Sampling more information is equivalent to a diagnostic test where each test provides imperfect information about the job. Anand et al. (2011) studies a single-server queueing setting where customers join the queue based on the price, delay costs and quality of the service. The study assumes slower speed of service implies higher accuracy of the job completion. The study finds equilibrium prices, service speeds, and revenue of the service provider.

Chan et al. (2014) also adjusts the rate of service to balance accuracy and speed in a single-server queue. It uses probability of re-work as a measure of accuracy since a faster service may lead to higher probability of rework and hence, affecting the waiting time of other jobs in the system. In urgent healthcare setup, the decision maker needs to evaluate the urgency level of a patient by delaying patient's treatment service. Sun et al. (2018) models the decision of adding a triage evaluation to increase the accuracy of service but at the cost of delayed service to all patients. The paper finds that performing triage evaluation is helpful only when there are enough patients waiting in the system.

All of the above papers study systems with only a single knowledge server, whereas our study focuses on the service systems where decision makers seek the opinions of parallel knowledge servers at the same

time to find job's type. In our setting, a lower speed implies waiting for additional evaluations from parallel servers and hence, increase accuracy of job completion decision.

Wang et al. (2010), however, do employ multiple knowledge servers to answer the patient calls in diagnostic service call centers. The decision maker tries to balance among accuracy of diagnosis, waiting time of each patient, and staffing cost of employing each knowledge server. But still, the accuracy of service is measured by the duration of a call. This paper is significantly different from FJ queues since each patient is served by only one knowledge server.

Process Repetition: In Alizamir et al. (2013), same diagnostic test is performed repeatedly to increase the accuracy of final diagnostic prediction decision. Using Bayesian updating on the belief of patient's type, the paper finds a threshold on beliefs to stop additional service if the current belief increases the threshold. The paper uses only one server and the same knowledge server performs the testing. In our paper, we employ multiple servers with different knowledge levels in a FJ setting.

Job Routing: Saghafian et al. (2018) consider a two-level KBSS in which the less knowledgeable servers at the lower level can refer a job to a more knowledgeable server at the upper level for more accurate decision. This compromises the speed of service of a job by increasing congestion at the upper level of the system. The KBSS in this paper is a two-stage tandem queue where an arriving job is send to only one server at stage 1. Our paper, however, has a fork-join queueing structure in which an arriving job is copied and sent to all parallel servers.

In an another application of KBSS in healthcare operations, Feizi et al. (2023) analyses the routing of an incoming patient to one of the two types of patient treatment units (ED beds or chair-based Vertical Processing Pathway (VPP) unit). Depending on the initial triage, the patient can be routed to a VPP unit instead of a regular ED bed. This reduces the congestion for ED beds at the risk of treating the patient under low facilities. However, later, a VPP patient may have to be re-routed to ED bed by delaying its own treatment as well as of the other patients. The study finds that using VPP unit can help reduce waiting times for ED beds. Whereas, in FJ queues, parallel servers process the copy of a job in parallel before a job completion decision is made.

Redundancy: Fork-join queues are used to model the redundancy in the service within a KBSS. We now review the fork-join queues literature to highlight the novelty of our model. The studies related to fork-join queues can be divided into two groups: (1) Performance Analysis, and (2) Optimal Control.

Because of the complexity of analysis of fork-join queues, the exact analysis is very limited. We note that there are a few studies which model the fork-join queue without any redundancy where an incoming job is copied to *all* parallel servers and the job is removed from the system only when *all* of its copies have been processed. Varma and Makowski (1994) studied the mean waiting time of jobs in the system with only 2

parallel servers. Varki et al. (2008) generalises the waiting time estimation in the basic fork-join for any n number of parallel servers.

For FJ queues with redundancy, there are studies analysing the bound on waiting times for exponential and general service time distributions of parallel servers (Joshi et al. (2012)), or providing closed form expression for mean waiting time when jobs are completed as soon as only one of the parallel servers processes a copy (Gardner et al. (2017)). Raaijmakers et al. (2023) and Carmeli et al. (2023) analyze the response time distribution in FJ queues.

In fork-join queues there are primarily three types of control decision problems: (1) Fork Decision: It involves deciding how many copies to generate for an incoming job and which parallel servers should receive a copy?, (2) Join Decision: It decides when a job is completed, i.e., how many evaluations from parallel servers are required to remove a job from the system?, and (3) Parallel Server Scheduling: It involves control of sequence of jobs to process at each parallel server and the join server. For fork decision, Liu et al. (2022), Gardner et al. (2017), and Joshi et al. (2015) analyse waiting time performance by varying the number of copies to generate for any incoming job. Whereas, in our study we analyse the decision making at the join server. Joshi et al. (2017) discusses the optimal join decision while minimizing latency and cost of computing. The study assumes change in service time for a job based on number of servers used to evaluate a job. To our knowledge, our study is the first that consider the join decision in FJ queues with knowledge servers to balance speed and accuracy.

Some studies have analysed scheduling of jobs with the FJ queues. For example, Özkan and Ward (2019) studies a system in which each type of incoming job at the fork node is copied to arbitrary number of parallel servers and the join decision is made after a particular number of copies are evaluated by the parallel servers. The paper solves for a control policy to decide the sequence of copies to process at each parallel server. Our paper is different, since we analyze the optimal control of join decision, whereas, Özkan and Ward (2019) optimizes the scheduling decision at the parallel queues. Furthermore, our goal is to capture the optimal trade-off between accuracy and speed while Özkan and Ward (2019) minimizes the waiting time of the jobs in the system.

Atar et al. (2012) compares the fork-join queues with synchronized flow of jobs and fork-join queues that allow exchangeable use of evaluations from parallel servers for the join decision. The study shows equivalence between two fork-join queues under a heavy traffic regime. Our study focuses on fork-join queues with a synchronized flow of jobs, i.e., the join decision for a job can only be made by aggregating evaluations of the copies of the same job.

4 Fork-Join Decision Flow Model

Consider a fork-join queue (see Figure 1) consisting of N parallel knowledge servers. Each incoming job arrives at the fork node, where it is copied to all parallel servers. These servers generate the evaluations for each copy coming to them. There is a join server who makes the join decision upon receiving evaluations from *all* or *some* of the parallel servers. In this section, we first describe dynamics of the join server decision making. Then, we explain how knowledge servers generate their evaluations, and we define our performance measures to evaluate any decision policy. We finally introduce a Markov decision problem to capture the optimal speed-versus-accuracy trade-off. The last part of this section characterizes the structure of the optimal dynamic join decision policy.

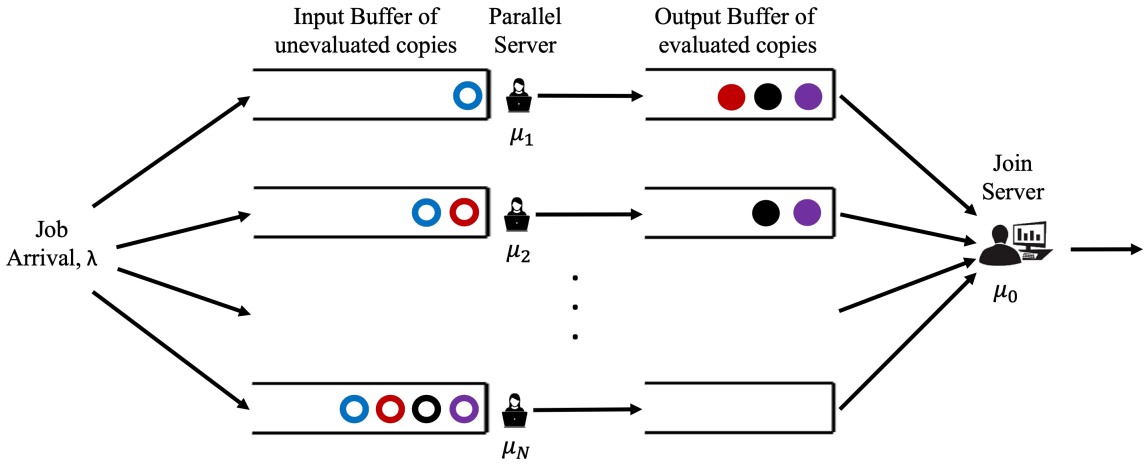

Figure 1: Fork-Join queueing system

4.1 Dynamics of the Join Server Decisions

We assume a job (e.g., a news item or a patient in emergency unit) enters the system following an arrival process Λ with mean inter-arrival time $1/\lambda$. Considering j as the index for the j^{th} arrival job, job Z_j can be of Type 1, i.e., $Y_j = 1$ (e.g., a false news item or a high risk patient) or Type 0 with $Y_j = 0$ (e.g., a true news item or a low risk patient). The type of any incoming job is unknown to all servers. At the fork node, each job Z_j is then copied and sent to all N parallel knowledge servers. Each Server $i \in \mathcal{N} \triangleq \{1, 2, ..., N\}$ has input buffer of un-evaluated copies of length $N_i(t)$ at time t. These N parallel servers can represent the third-party fact-checkers used by social media platforms, or different diagnostic testing services in a hospital emergency care, etc.

With a random service time S_i with mean $1/\mu_i$, Server i generates an evaluation $x_j^i \in [0, 1]$ for its copy of job Z_j. An evaluation score $x \in [0, 1]$ can be considered as the estimate of server's confidence that the

job copy is of Type 1. Thus, for a hypothetical server with perfect knowledge, it would always generate an evaluation score $x = 0$ for a Type-0 job and a score $x = 1$ for a Type-1 job. How these evaluations are generated is discussed further in detail in the next subsection. After a server provides the evaluation for a copy of job Z_j, the copy is then transferred to the corresponding output buffer of the server with its evaluation score attached to it. Here the vector of evaluation scores $\mathbf{x}_{k_i}^{(i)}(t) \triangleq [x_1^i, x_2^i, ..., x_{k_i}^i]$ denotes the queue of k_i scores in the output buffer of Server i at time instant t. These scores are generated by Server i after processing k_i number of copies of different jobs.

The join server (e.g., social media platform or emergency department doctor) uses these evaluations to make join decision to finally predict job as either Type 1 or Type 0. We define an active set $\mathcal{I}_j$ for job Z_j. Active set $\mathcal{I}_j \subset \mathcal{N}$ consists of the indices of servers who have already completed their evaluations for the copies of job Z_j, and now these copies are in their output buffers. The join server aggregates the evaluations from servers in active set $\mathcal{I}_j$ using an aggregation function $G^{\mathcal{I}_j} : [0,1]^{|\mathcal{I}_j|} \to [0,1]$. This function is assumed to be non-decreasing in $x \triangleq (x_j^k)_{\forall k \in \mathcal{I}_j}$, which implies that the aggregated estimate of a job being of Type 1, $G^{\mathcal{I}_j}(x_j^k)_{\forall k \in \mathcal{I}_j}$, increases with the increase in any of the active server's evaluation. Since all servers are assumed to be knowledgeable, the join server would not ignore any server's evaluation and uses them in its aggregated estimate for each job. The processing time for the join server to aggregate the given evaluations for a job and make a decision is random variable S_0 with mean $1/\mu_0$.

4.2 Evaluation Process of Parallel Servers

To model the evaluations, we consider the state of the world $Y = \{0, 1\}$, and we denote $Y = 1$ when the job is of Type 1, and $Y = 0$ denoting the job is of Type 0 where $Y = 1$ in social media platforms, for example, represents a news item being false and $Y = 0$ represents the news item being true. The type of any job Y is unknown to all parallel servers and the join server. For instance, in social media platforms, knowing exactly that a newly posted news item is true or false is not possible, but there is an overall estimate for fraction of posted jobs being true or false. We assume that the overall probability of a job being of Type 1 is denoted by $p \triangleq \mathbb{P}(Y = 1)$ and is known to parallel servers and the join server. Server i makes an initial evaluation for the type of its copy of a job and then generates an evaluation score $X_i = x$ where $x \in [0, 1]$ for its copy. Here, the evaluation score x can be considered as a normalized score, the closer the score is to one, the higher is the server's confidence that the job is of Type 1.

Now suppose we conduct two experiments. In Experiment 1, we choose 100 Type-0 jobs and run them through the server evaluation process and plot the frequency histogram of the scores generated by the server. In Experiment 2, we choose 100 Type-1 jobs, and run them through Server i. If the server has perfect knowledge in identifying the type of a job, then in Experiment 1 the server will generate score $x = 0$

for all 100 jobs and in Experiment 2 the server will generate score $x = 1$ for all its 100 jobs. In reality the servers do not have perfect knowledge, thus the histograms for scores from the server in both experiments would look like those in Figure 2. As figure 2 shows the distribution of the scores in Experiment 1 in which $Y = 0$ is left-skewed, since a knowledge server would generate more scores closer to 0. The distribution in Experiment 2, however, is right-skewed, since the server should generate scores closer to 1. These show a close approximation to Beta distribution.

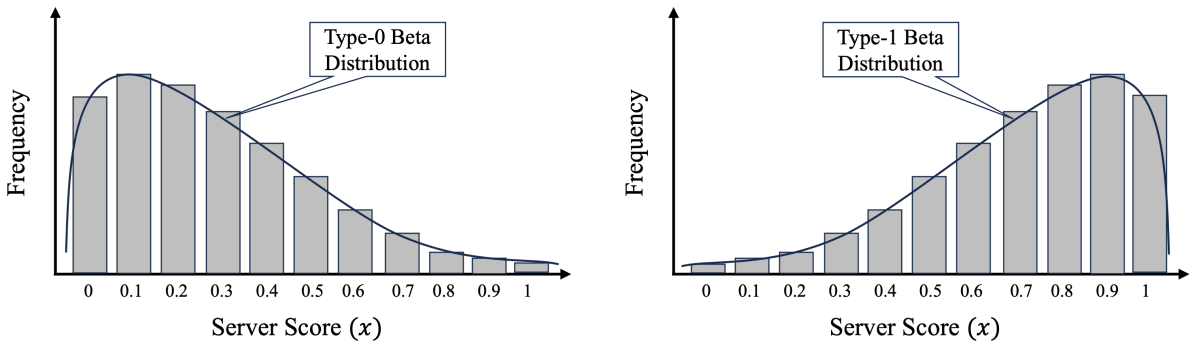

Figure 2: Beta Distribution Approximation to Score Histogram (*Left:* Experiment 1, *Right:* Experiment 2)

Similar to Saghafian et al. (2018), we, therefore, model the server's knowledge using two Beta distributions. The server's evaluation follows a Type-1 $f_i^1(x)$ (Type-0 $f_i^0(x)$) Beta distribution if initial evaluation suggests that its copy of a job is of Type 1 (Type 0), i.e., for $Y = 1$ ($Y = 0$) where,

$$f_i^y(x) = \frac{1}{B(\alpha_i^y, \beta_i^y)} x^{\alpha_i^y - 1}(1-x)^{\beta_i^y - 1},$$

where $B(\alpha_i^y, \beta_i^y) = \int_0^1 v^{\alpha_i^y - 1}(1-v)^{\beta_i^y - 1} dv$ for $y \in \{0, 1\}$.

Since the server doesn't know the true type of the job, it implies that the evaluation score X_i follows Type-1 Beta distribution where $X_i \stackrel{d}{=} X_i^1 \sim f_i^1(x)$ with probability p and Type-0 Beta distribution $X_i \stackrel{d}{=} X_i^0 \sim f_i^0(x)$ with probability $1-p$. We can then describe the knowledge distribution $f^{(i)}$, for Server i, using two Beta distributions, i.e., $f_i^1()$ with probability p and distribution $f_i^0()$ with probability $1-p$. We assume that $\alpha_i^y \geq 1, \beta_i^y \geq 1$ to ensure that f_i^y is bounded and also that evaluation score is continuous in $[0, 1]$.

We further characterize the probability of any job is of Type 1 given the random evaluation score $X_i = x$ for its copy from Server i as:

$$P_i(x) \triangleq \mathbb{P}(Y = 1 | X_i = x) = \frac{p f_i^1(x)}{p f_i^1(x) + (1-p) f_i^0(x)}$$

We assume for the rest of our analysis that the servers are rational in their evaluations. Intuitively for rational servers $P_i(x)$, the probability of the true type of job being Type 1, is increasing in the evaluation

score x for its copy by Server i.

4.3 Evaluation Aggregation process of the Join Server

Recall that active set $\mathcal{I}_j$ denotes the indices of servers whose evaluation scores are with the join server, i.e., in the output buffers of servers in $\mathcal{I}_j$. Each server in set $\mathcal{I}_j$ has generated the evaluation score x_j^i for their copy of job Z_j. For job Z_j, with aggregated score $G^{\mathcal{I}_j}(x_j^i)_{\forall i \in \mathcal{I}_j}$ from servers in set $\mathcal{I}_j$, the join server evaluates the probability that job Z_j is a Type-1 job based on its knowledge (modeled by a Beta distribution $f^y_{\mathcal{I}_j}(.)$ for $y \in \{0,1\}$). Thus, for aggregated score $x = G^{\mathcal{I}_j}(x_j^i)_{\forall i \in \mathcal{I}_j}$, the probability that job Z_j is a Type-1 job is:

$$P_{\mathcal{I}_j}(x) \triangleq \mathbb{P}(Y=1|x) = \frac{p f^1_{\mathcal{I}_j}(x)}{p f^1_{\mathcal{I}_j}(x) + (1-p) f^0_{\mathcal{I}_j}(x)}$$

Hence, for every possible subset $\mathcal{I}_j$ in the power set of $\mathcal{N}$, i.e., for all combinations of m servers out of all N servers, we have corresponding Beta knowledge distributions $f^y_{\mathcal{I}_j}()$ (for $y \in \{0,1\}$) to evaluate the probability of a job being a Type-1 job, given the evaluations from servers in the active set.

Assumption 1 (Error reduction through additional scores):

$$P_{\mathcal{I}_j^1}(G^{\mathcal{I}_j^1}(x,y^1)) - P_{\mathcal{I}_j^1}(G^{\mathcal{I}_j^1}(x',y^1)) \leq P_{\mathcal{I}_j^2}(G^{\mathcal{I}_j^2}(x,y^2)) - P_{\mathcal{I}_j^2}(G^{\mathcal{I}_j^2}(x',y^2)), \tag{1}$$

where, $x, x'(x' < x)$ *are evaluation scores from Server* $i \in \mathcal{N}$ *for job* Z_j, $\mathcal{I}_j^v \triangleq S^v \bigcup \{i\}$, $v \in \{1,2\}$, *with* $S^2 \subset S^1 \subset P(\mathcal{N}) \setminus \{\emptyset, \mathcal{N}\}$ *and the evaluation scores* $y^v \triangleq (x^u)_{\forall u \in S^v}$, $v \in \{1,2\}$.

Assumption 1 implies that for given evaluation scores x and x' from a server among parallel servers for a copy of job Z_j where $x' \leq x$, the difference in probability of job Z_j being of Type 1 decreases with any additional evaluation score from other parallel servers. Here we use an example with $N = 3$ parallel servers to illustrate Assumption 1 when Server 1 generates the evaluation for a copy first (w.l.o.g.). In Case-1 Server 1 gives score of x for the copy of job Z_j and in Case-2, Server 1's score is x' for the same copy such that $x' \leq x$. Now the join server waits for evaluation from other servers and it receives a score y from Server 2 for the copy of same job Z_j. Thus, $\mathcal{I}_j^1 = \{1,2\}$. Assumption 1 states that,

$$P_{\mathcal{I}_j^1}(G^{\mathcal{I}_j^1}(x,y)) - P_{\mathcal{I}_j^1}(G^{\mathcal{I}_j^1}(x',y)) \leq P_1(x) - P_1(x')$$

The right hand side is the difference of the probabilities that job Z_j is of Type 1 using only Server 1's scores x and x'. Similarly, the left hand side is the difference in probabilities when the join server has scores from two servers.

To further understand the assumption as an example consider a case, when Server 3 generates a score of z for its copy of same job Z_j, i.e., $\mathcal{I}_j^2 = \{1,2,3\}$. The assumption states:

$$P_{\mathcal{I}_j^2}(G^{\mathcal{I}_j^2}(x,y,z)) - P_{\mathcal{I}_j^2}(G^{\mathcal{I}_j^2}(x',y,z)) \leq P_{\mathcal{I}_j^1}(G^{\mathcal{I}_j^1}(x,y)) - P_{\mathcal{I}_j^1}(G^{\mathcal{I}_j^1}(x',y)) \leq P_1(x) - P_1(x')$$

The left hand side is the difference in probability of job Z_j being of Type 1 given scores from all three servers. Both inequalities intuitively describes the difference in probabilities for a job being of Type 1 should decrease as the join server receives more information about a job. As the join server receives evaluation score from additional servers (Server 2 and then, Server 3) the difference in estimation of true type of job decreases. This error in the estimate of true type of the job from only one server's evaluation is overcome by evaluations of other servers.

4.4 Capturing Speed and Accuracy Trade-off

Using evaluations from servers in the active set $\mathcal{I}_j \subset \mathcal{N}$ for a job Z_j, and $|\mathcal{I}_j| = m$, the join server can take three actions: (1) $\mathcal{A}_w$: wait for one more evaluation from the parallel servers in $\mathcal{N} \setminus \mathcal{I}_j$; (2) $\mathcal{A}_1$: predict the job as Type-1 job, and (3) $\mathcal{A}_0$: predict the job as Type-0 job. The first action $\mathcal{A}_w$ improves accuracy of the final decision at the cost of reducing speed due to waiting for additional evaluation. Whereas, two actions $\mathcal{A}_1$ and $\mathcal{A}_0$ are job prediction decisions when the join server, with m out of N evaluations, decides not to wait for additional evaluation from $N-m$ servers, and it takes a prediction decision $\mathcal{A}_1$ or $\mathcal{A}_0$ to avoid delay (i.e., increasing speed) at the cost of lower of accuracy.

To balance the two objectives of maximizing accuracy and speed when they are in conflict with each other, we define weights for both objectives. We introduce a per unit time holding cost h for a job waiting in the system. Holding cost captures the speed of the fork-join system to process jobs. To capture accuracy, we consider the cost of prediction error. Specifically, we define error cost $c_1(c_0)$ when the job's true type is $Y = 1\,(Y = 0)$ but the join server predicts the job as Type-0 (Type-1) job, i.e., the join server takes action $\mathcal{A}_0(\mathcal{A}_1)$. For given aggregated score x from servers in $\mathcal{I}_j \subset \mathcal{N}$, we can find the expected Type-1 and Type-0 prediction error costs. When the join server predicts job Z_j as Type-0 job, the expected Type-1 prediction error cost is $c_1 P_{\mathcal{I}_j}(x)$; similarly the expected Type-0 prediction cost is $c_0(1 - P_{\mathcal{I}_j}(x))$.

Assumption 2: $\exists x \in (0,1)$ *where, x is an aggregated score from evaluations made by servers in set $\mathcal{I}_j$ for the copies of job Z_j, we have:*

$$c_1 P_{\mathcal{I}_j}(x) \leq c_0(1 - P_{\mathcal{I}_j}(x)) \tag{2}$$

Assumption 2 implies that there exists an aggregated score x for which Type-0 prediction has lower expected error cost than that of Type-1 prediction made by the join server. Otherwise, we will have the trivial case in which the join server will predict all jobs as Type-1 jobs.

4.5 The Optimal Dynamic Join Decision Policy

In this section we formally write the optimization model to obtain a control policy that captures the optimal trade-off between speed and the accuracy objectives. Let $\bar{\pi}$ be any control policy of the join server. We

represent the number of jobs waiting in the input buffer or being served by parallel Server $i \in \mathcal{N}$ under policy $\bar{\pi}$ as $N_i^{\bar{\pi}}(t)$, and the number of jobs waiting in the output buffer of parallel Server i under control policy π as $N_i^{s\bar{\pi}}(t)$. We note that for each server, the sum of the copies in its input and output buffers plus the one copy being served would be the same. This is because they all receive copies of the same job and all copies of a job is removed from all queues after its completion at the same time. We also define $\mathcal{E}_0^{\bar{\pi}}(t)$ and $\mathcal{E}_1^{\bar{\pi}}(t)$ as the cumulative number of type-0 and type-1 prediction errors made, respectively, up to time t by the join server under policy $\bar{\pi}$. Thus, using the above notation, the long-run total average cost of the system per unit time under control policy $\bar{\pi}$ is:

$$\phi(\bar{\pi}) \triangleq \liminf_{t\to\infty} \frac{1}{t}\mathbb{E}\left[\int_0^t h[N_i^{\bar{\pi}}(u) + N_i^{s\bar{\pi}}(u)]\mathrm{du} + c_0\mathcal{E}_0^{\bar{\pi}}(t) + c_1\mathcal{E}_1^{\bar{\pi}}(t)\right],$$

With Π being the set of all the control policies for join server, we seek to find the optimal policy π^* that minimizes the long-run average cost per unit time:

$$\pi^* = \underset{\bar{\pi}\in\Pi}{\operatorname{argmin}}\ \phi(\bar{\pi}). \tag{3}$$

4.6 The Markov Decision Problem (MDP)

Recall that the join decision control policy answers the two key questions: (1) Should the join server make a prediction decision using the available evaluations, or should it wait for another evaluation? and (2) If the join server decides not to wait, what should be the join server's final prediction for a job (i.e., Type-1 job or Type-0 job)? We model the join server's decision making as Markov decision process to find the answers to both questions.

We would like to emphasis that, due to complex dynamics of job arrivals and departures in fork-join queues, literature on fork-join queues are limited to simple Markovian systems (Liu et al. (2022), Özkan (2022)). Similarly, we also assume a Markovian fork-join queue in which the inter-arrival times and server's service times are exponentially distributed and are independent of each other. As we will show, this assumption allows us to fully characterize the optimal dynamic join decision policy. One can use Hyper-exponential or Erlang distribution to model the processing times to mimic other distributions. While this adds to the complexity of the analysis, it will not provide significantly different insights from what we find using our MDP model.

We can rescale time (without loss of generality) and assume that the event rate is $\Lambda' := \lambda + \sum_{i=0}^{N} \mu_i = 1$, and we formulate our Markov Decision process as follows:

- *State of the System is* $s \triangleq (\mathbf{n}, \mathbf{X})$, where $\mathbf{n} \in \mathbb{Z}^{+N}$ with $\mathbf{n} \triangleq (n_1, n_2, ..., n_N)$ being a vector of queue lengths of number of copies in the input buffer and the one in-process of all N servers, and $\mathbf{X} =$

$(\mathbf{x}_{k_1}^{(1)}, \mathbf{x}_{k_2}^{(2)}, \mathbf{x}_{k_3}^{(3)}, .., \mathbf{x}_{k_N}^{(1)})$ is the vector of evaluation scores in the output buffers of all N parallel servers. For a given $i \in \mathcal{I}_j$, the i^{th} vector of $\mathbf{X}$, denoted by $(\mathbf{X})_i \triangleq \mathbf{x}_{k_i}^{(i)} = (x_j^i, x_{j+1}^i, ..., x_{j+k_i-1}^i)$ is the vector of evaluation scores generated by Server i for $k_i \in \mathbb{Z}^+$ number of copies such that the total jobs in the system $n = n_u + k_u = n_v + k_v$, $\forall\, u, v \in \mathcal{N}$. The join server has already completed jobs $Z_0, Z_1, ..., Z_{j-1}$, hence it has an in-process job Z_j. As an example, Figure 3 shows the state of the MDP with $N = 3$ parallel servers. The system has four jobs: Server 1 has three copies in its output buffer with their evaluation scores; Server 2 has provided evaluation for two copies, and Server 3 still has the copies of all 4 jobs in its input buffer or in processing stage. Thus, the state of system in the figure is $s = (\mathbf{n}, \mathbf{X})$, where $\mathbf{n} = (n_1, n_2, n_3)$ and $\mathbf{X} = (x_1^1, x_2^1, x_3^1, x_1^2, x_2^2)$ and $\mathcal{I}_1 = \{1, 2\}$ (representing the active set for red colored job in Figure 3).

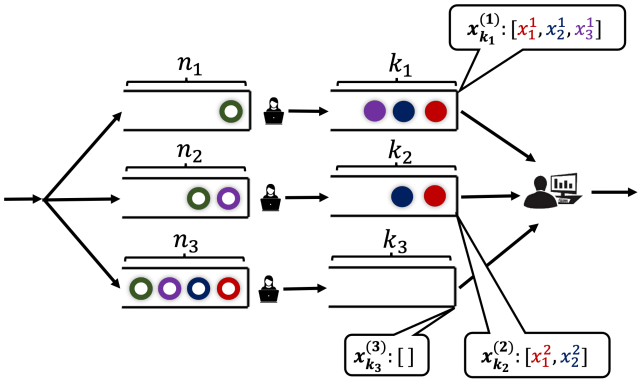

Figure 3: MDP State s for $N = 3$ Parallel Servers

- *Decision Epochs* are service completion at the join server, arrival epochs and service completion epochs of parallel servers.

- *Actions:* At every service completion epoch at the join server, the server aggregates all the available evaluation scores for the in-process job which are available in output buffers of parallel servers and takes one of the following three actions:

  - Wait ($\mathcal{A}_w$): wait for the next evaluation for the same job from parallel servers,
  - Predict Type 0 ($\mathcal{A}_0$): completes the job with the prediction that it is a Type-0 job,
  - Predict Type 1 ($\mathcal{A}_1$): completes the job with the prediction that it is a Type-1 job.

Considering state space $\mathcal{S} \triangleq \{s = (\mathbf{n}, \mathbf{X}) \,|\, \mathbf{n} = (n_1, n_2, ..., n_N), \mathbf{X} = (\mathbf{x}_{k_1}^{(1)}, \mathbf{x}_{k_2}^{(2)}, ..., \mathbf{x}_{k_N}^{(1)}),\ n_u + k_u = n_v + k_v,\ \text{and}\ n_u, k_u \geq 0,\ \forall\, u, v \in \mathcal{N}\}$, to describe the optimality equations of the MDP, we divide the state space $\mathcal{S}$ into three sets: (i) No Evaluations set ($\mathcal{S}_0$), (ii) m Evaluations set ($\mathcal{S}_m$), and (iii) All Evaluations set ($\mathcal{S}_N$), where for any state $s \in \mathcal{S}$,

- No Evaluations set ($\mathcal{S}_0$) is defined when $k_1, k_2, ..., k_N = 0$. This corresponds to the states in which there are no actions for the join server to take, since it has not yet received any evaluation. Thus, when there are n total number of jobs in the system, for any $s_0 \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_0$, we have $s_0 = n\mathbf{1}_N$, where $\mathbf{1}_N$ is a $1 \times N$ dimensional vector of ones. Letting $V(s_0)$ be the relative value function for state s_0, the optimality equation is:

$$\begin{aligned} \pi^* + V(s_0) = h(n) + \lambda V(\mathbf{n} + \mathbf{1}_N) + \sum_{i=1}^{N} \mu_i \mathbb{I}_{\{n>0\}} \int_0^1 f^{(i)}(a) V(\mathbf{n} - \mathbf{e}_i^N, a) da \\ + \sum_{i=1}^{N} \mu_i \mathbb{I}_{\{n=0\}} V(s_0) + \mu_0 V(s_0), \end{aligned} \tag{4}$$

  where $\mathbf{e}_i^N$ is a $1 \times N$ dimensional vector of zeros with its i^{th} component $[\mathbf{e}_i^N]_i = 1$, recall π^* as the optimal long-run average cost and the indicator function $\mathbb{I}_{\{B\}} = 1$ if condition B is true, and 0, otherwise.

  In the first line of equation (4), the first term on the right hand side represents the total holding cost for one period, and the second term represents the transition to state with an additional job in all parallel servers' input buffers when an arrival occurs. The third term shows the transitions upon service completion at each parallel server by generating the evaluation score based on its knowledge distribution. Lastly, both terms in the second line represent no transitions for parallel server service completion with empty input buffers and for the join server service completion, since the output buffers are also empty.

- m Evaluations set ($\mathcal{S}_m$) is defined when m out of N ($0 < m < N$) parallel servers have non-zero output buffers of size $k_i > 0$, for $i \in \mathcal{I}_j$, and $k_i = 0$, $\forall\ i \in \mathcal{N} \setminus \mathcal{I}_j$. This corresponds to the states in which the join server has m ($= |\mathcal{I}_j| < N$) evaluations out of N to make a decision for an in-process job Z_j. Therefore, for states $s_m \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$ with $\mathbf{n} = n\mathbf{1}_N - \sum_{i \in \mathcal{I}_j} k_i \mathbf{e}_i^N$, $\mathbf{X} = (\mathbf{x}_{k_u}^{(u)})_{\forall u \in \mathcal{I}_j}$ and $x_j^u \triangleq [\mathbf{x}_{k_u}^{(u)}]_1$, $\forall u \in \mathcal{I}_j$, the optimality equation is as follows:

$$\begin{aligned} \pi^* + V(s_m) = h(n) + \lambda V(\mathbf{n} + \mathbf{1}_N, \mathbf{X}) + \sum_{i=1}^{N} \mu_i \mathbb{I}_{\{n_i>0\}} \int_0^1 f^{(i)}(a) V(\mathbf{n} - \mathbf{e}_i^N, \phi(\mathbf{X}, i, a)) da \\ + \sum_{i=1}^{N} \mu_i \mathbb{I}_{\{n_i=0\}} V(s_m) + \mu_0 \min \begin{cases} V(\mathbf{n} - \sum\limits_{i \in \mathcal{N} \setminus \mathcal{I}_j} \mathbf{e}_i^N, \sigma^{\mathcal{I}_j}(\mathbf{X}, j)) + c_1 P_{\mathcal{I}_j}(x) & : \mathcal{A}_0, \\ V(\mathbf{n} - \sum\limits_{i \in \mathcal{N} \setminus \mathcal{I}_j} \mathbf{e}_i^N, \sigma^{\mathcal{I}_j}(\mathbf{X}, j)) + c_0(1 - P_{\mathcal{I}_j}(x)) & : \mathcal{A}_1, \\ V(s_m) & : \mathcal{A}_w. \end{cases} \end{aligned} \tag{5}$$

where the aggregated score for job Z_j is $x = G^{\mathcal{I}_j}(x_j^u)_{\forall u \in \mathcal{I}_j}$, the evaluation score vector after removing job Z_j from system is denoted by $\sigma^{\mathcal{I}_j}(\mathbf{X}, j)$ and the updated evaluation score vector $\phi(\mathbf{X}, i, a)$ adds the new score a from Server i to the score vector.

Formally, we define $\sigma^{\mathcal{I}_j}(\mathbf{X}, j)$ such that $(\sigma^{\mathcal{I}_j}(\mathbf{X}, j))_v = (x_{j+1}^{(v)}, x_{j+2}^{(v)}, ..., x_{j+k_v-1}^{(v)}),\ \forall\, v \in \mathcal{I}_j$. Note that $\sigma^{\mathcal{I}_j}(\mathbf{X}, j)$ is a $1 \times (\bar{k} - m)$ dimensional vector with $\bar{k} \triangleq \sum_{c=1}^{N} k_c$.

Now, we define a $1 \times (\bar{k} + 1)$ dimensional vector $\phi(\mathbf{X}, i, a)$ using $\bar{c} = \sum_{\forall v \in \mathcal{I}_j,\, v \leq i} k_v$,

$$[\phi(\mathbf{X}, i, a)]_v = \begin{cases} [\mathbf{X}]_v, & : \forall v \in \{1, 2, ..., \bar{c}\}, \\ a, & : v = \bar{c} + 1, \\ [\mathbf{X}]_{v-1}, & : \forall v \in \{\bar{c} + 2, ..., \bar{k} + 1\} \end{cases}$$

The optimality equation (5) for $\mathcal{S}_m$ set of states is similar to $\mathcal{S}_0$ equation (4) with additional optimal decision for the join server with m out of N servers' score, as represented by the term in the third line of the right hand side.

- All Evaluations set ($\mathcal{S}_N$) corresponds to states with all $k_1, k_2, ..., k_N > 0$. These are the states in which the join server has received evaluations from all servers for job Z_j to complete the job and make a prediction decision.

  For $s_N \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_N$ with $\mathbf{n} = n\mathbf{1}_N - \sum_{i \in \mathcal{N}} k_i \mathbf{e}_i^N,\ \mathbf{X} = (\mathbf{x}_{k_1}^{(1)}, \mathbf{x}_{k_2}^{(2)}, ..., \mathbf{x}_{k_N}^{(N)})$ and $x_j^i \triangleq [\mathbf{x}_{k_i}^{(i)}]_1,\ \forall i \in \mathcal{N}$, the optimality equation is, therefore,

$$\begin{aligned} \pi^* + V(s_N) = h(n) + \lambda V(\mathbf{n} + \mathbf{1}_N, \mathbf{X}) + \sum_{i=1}^{N} \mu_i \mathbb{I}_{\{n_i > 0\}} \int_0^1 f^{(i)}(a) V(\mathbf{n} - \mathbf{e}_i^N, \phi(\mathbf{X}, i, a)) da \\ + \sum_{i=1}^{N} \mu_i \mathbb{I}_{\{n_i = 0\}} V(s_N) + \mu_0 \min \begin{cases} V(\mathbf{n}, \sigma^{\mathcal{N}}(\mathbf{X}, j)) + c_1 P_{\mathcal{N}}(x) & : \mathcal{A}_0, \\ V(\mathbf{n}, \sigma^{\mathcal{N}}(\mathbf{X}, j)) + c_0 (1 - P_{\mathcal{N}}(x)) & : \mathcal{A}_1. \end{cases} \end{aligned} \tag{6}$$

  where, the aggregated score for job Z_j is $x = G^{\mathcal{I}_j}(x_j^i)_{\forall i \in \mathcal{I}_j}$.

  The second term in the second line of the right hand side consists of only two possible actions for the join server to take. Since the join server has received evaluations from all servers, waiting for another evaluation (i.e., action $\mathcal{A}_w$) is not feasible.

THEOREM 1 (***Optimal Stationary Policy***): *For a stable FJ queue (i.e., $\lambda < \min_{i \in \mathcal{N}} \mu_i \leq \max_{i \in \mathcal{N}} \mu_i < \mu_0$), an average-cost optimal stationary policy exists for the MDP defined by equations* (4)-(6)*, ensuring constant average cost, with concurrent convergence of the value iteration algorithm.*

The detailed proof can be found in the appendix. The following remainder of the section characterises the structure of the optimal MDP solution.

4.7 Structure of the Optimal Dynamic Join Decision (DJD) Policy

To dynamically decide what number of evaluations are needed for the join server to make a decision for a job, we first characterize the optimal DJD policy with respect to the evaluation scores. Then, we characterize the structure with respect to the queue lengths, particularly, the total number of jobs in the system and the number of evaluated copies with the join server. At the end, we also answer the second question of finding the optimal prediction decision (i.e., Type-1 or Type-0 prediction) when a join decision must be made.

4.7.1 Optimal DJD Policy: Score Thresholds

In fork-join queues, the dynamic join decision is the control policy to remove the copies of a job from the system after receiving certain number of its evaluations, say m, from the servers. The optimal m changes dynamically depending on the number of copies in input and output buffers of parallel servers as well as on the evaluation scores for different copies of the job available with the join server. The following proposition characterizes the dependence of the optimal DJD policy on evaluation scores of the servers. The proof of all analytical results including Proposition 1 are presented in the appendix.

PROPOSITION 1 ***(Double-Threshold Policy for Scores):*** *If there exist states* $s_{\underline{x}} \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$ *with* $[\mathbf{x}_{k_i}^{(i)}]_1 = \underline{x}_j^i, \forall i \in \mathcal{I}_j$ *and* $s_{\bar{x}} \triangleq (\mathbf{n}, \mathbf{X}') \in \mathcal{S}_m$ with $[\mathbf{x}'^{(i)}_{k_i}]_1 = \bar{x}_j^i, \forall i \in \mathcal{I}_j$ *for which predicting* $\mathcal{A}_0$ *and* $\mathcal{A}_1$ *are optimal actions, respectively, then there exist thresholds* $\underline{x}^*_{\mathcal{I}_j}(n)$ *(and* $\bar{x}^*_{\mathcal{I}_j}(n)$*) below (above) which all the aggregated scores* $x \leq \underline{x}^*_{\mathcal{I}_j}(n)$ *(*$x \geq \bar{x}^*_{\mathcal{I}_j}(n)$*) result in* $A^*(s_x) = \mathcal{A}_0$ *(*$A^*(s_x) = \mathcal{A}_1$*).*

For a given aggregated evaluation score x from m servers in set $\mathcal{I}_j$, the above result suggests that if x is very small ($x \leq \underline{x}^*$) or very large ($x \geq \bar{x}^*$), the join server makes job completion decision with the current m evaluations and does not wait for another evaluation. In the case of $x \in (\underline{x}^*, \bar{x}^*)$, then the optimal DJD policy suggests waiting for an additional evaluation from the servers (see Figure 4).

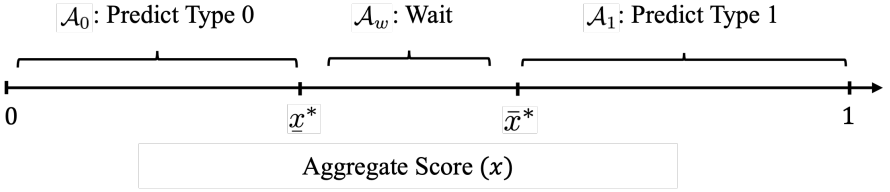

Figure 4: Double-threshold policy for a given number of jobs in the system n

Proposition 1 provides a double-threshold structure on the aggregated score for the decision of the join server when there are n jobs waiting in the system and m out of N evaluations are available. The join server can either make decision to wait for another evaluation while incurring holding cost (i.e., lower speed to process the job) or, it can compromise on accuracy and make a decision with m (instead of $m+1$ or more)

evaluations to avoid waiting. It is also important to note the impact of thresholds $\underline{x}^*$ and $\bar{x}^*$ on the total number of jobs in the system. The larger the distance between $\underline{x}^*$ and $\bar{x}^*$, the wider the optimal region for wait decision, resulting in higher accuracy but lower speed of service. The holding cost h and error costs of c_0 and c_1 affect the width of the optimal wait region and capture the optimal trade-off between the delay cost and the prediction error costs.

The following proposition describes how the values of score thresholds $\underline{x}^*$ and $\bar{x}^*$ change with the number of jobs waiting in the system.

PROPOSITION 2 ***(Monotonicity of the Double-Threshold Policy in the Total Number of Jobs):*** *For state $s_n \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$, the evaluation score thresholds $\underline{x}^*(n)$ and $\bar{x}^*(n)$ (here $n = [\mathbf{n}]_1 + k_1$) are, respectively, nondecreasing and nonincreasing in the total number of jobs in the system (n). Thus, the double-threshold policy has the following monotone property that "wait" region shrinks with increasing the total number of jobs in the system, i.e.,*

$$A^*(s_n) = \mathcal{A}_w \implies A^*(s_{n-1}) = \mathcal{A}_w$$

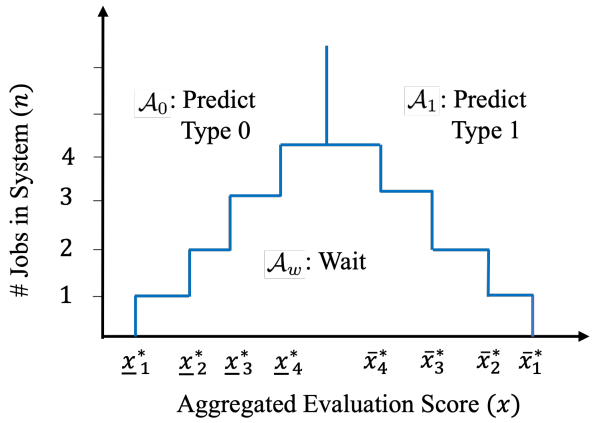

Figure 5: Monotonicity of double-threshold policy in the total number of jobs n in the system

Following Proposition 2, Figure 5 shows that the optimal DJD policy has a narrower double-threshold structure when there are more number of jobs waiting in the system. This is because, the join server would incur higher holding costs of waiting for additional score when there are already a large number of jobs waiting in the system. Thus, when there are either very high number of jobs waiting, or when the per unit holding cost relative to the error cost is too high, the waiting region beginning to disappear.

We have characterized the structure of the optimal DJD policy with respect to the evaluation scores. In the following subsection, we now describe the structure of the optimal DJD policy with respect to the total number of jobs as well as the number of evaluated copies waiting in the output buffers of each parallel server.

4.7.2 Optimal DJD Policy: Queue Length Thresholds

We can further characterize the join server decisions for a given aggregated score with respect to the change in the total number of jobs and the length of the output buffer of each server, i.e., the number of evaluated copies from a server.

LEMMA 1 ***(Optimal Threshold on the Total Number of Jobs in the system):*** *For state* $s_n \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$, *there exists a threshold on the number of jobs in the system* $n^*(x, (k_i)_{\forall i \in \mathcal{I}_j})$ *such that for an aggregated score* $x = G^{\mathcal{I}_j}([(\boldsymbol{X})_i]_1)_{\forall i \in \mathcal{I}_j}$, *from servers in active set* $\mathcal{I}_j$, *and* $k_i \triangleq |(\mathbf{X})_i|$ $\forall i \in \mathcal{I}_j$, *the optimal action with total* $n = [\boldsymbol{n}]_1 + k_1$ *jobs in the system follows:*

$$A^*(s_n) = \begin{cases} \mathcal{A}_w & : n \leq n^*(x, (k_i)_{\forall i \in \mathcal{I}_j}) \\ \mathcal{A}_0 \text{ or } \mathcal{A}_1 & : n > n^*(x, (k_i)_{\forall i \in \mathcal{I}_j}) \end{cases} \tag{7}$$

Lemma 1 shows that the join server waits for another evaluation from a parallel server only when the total number of jobs in the system are less than a certain threshold $n^*(x, (k_i)_{\forall i \in \mathcal{I}_j})$. The threshold depends on the aggregated score for the job and also, the number of copies in the output buffers of each server. We find a similar threshold structure, as in Lemma 1, on the number of copies in the output buffer of any Server $i \in \mathcal{N}$.

LEMMA 2 ***(Optimal Threshold on the Number of Copies in the Output Buffers):*** *For state* $s_k \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$, *there exists a threshold* $k_i^*(x, n)$, $\forall i \in \mathcal{I}_j$ *such that for an aggregated score* $x = G^{\mathcal{I}_j}([(\boldsymbol{X})_i]_1)_{\forall i \in \mathcal{I}_j}$, *from servers in active set* $\mathcal{I}_j$, *and total jobs in the system* $n \triangleq [\mathbf{n}]_1 + |(\mathbf{X})_1|$, *the optimal action with number of copies* $k \triangleq |(\boldsymbol{X})_i|$ *in the output buffer of Server* i *follows:*

$$A^*(s_k) = \begin{cases} \mathcal{A}_w & : k \leq k_i^*(x, n) \\ \mathcal{A}_0 \text{ or } \mathcal{A}_1 & : k > k_i^*(x, n) \end{cases} \tag{8}$$

Lemma 2 describes the output buffer length threshold $k_i^*(x, n)$, $\forall i \in \mathcal{I}_j$ for Server i, which is dependent on the aggregated score for the job being processed at join server and the total number of jobs in the system. Therefore, we now focus on the dependence of the optimal queue length thresholds of the DJD policy with respect to k_i, i.e., the number of copies in the output buffer of Server i, and with respect to n, i.e., the total number of jobs in the system (= input buffer + output buffer + the job in service). The following result shows that the optimal DJD policy has a step-wise threshold policy with respect to these two queue lengths.

LEMMA 3 ***(Monotonicity of Queue Length Thresholds w.r.t. Length of Output Buffer):*** *For a state* $s \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$, *with a fixed aggregated score* $x \triangleq G^{\mathcal{I}_j}(x_j^1, x_j^2, ..., x_j^m)$ *where* $x_j^i \triangleq [\mathbf{x}_{k_i}^{(i)}]_1$ $\forall i \in \mathcal{I}_j$, *the optimal threshold on the total number of jobs in the system* $n^*(x, (k_u)_{\forall u \in \mathcal{I}_j})$ *is nonincreasing in* k_i $(\forall i \in \mathcal{I}_j)$.

The above results characterises the dependence of queue length threshold $n^*(x, (k_u)_{\forall u \in \mathcal{I}_j})$ on the number

of copies in the output buffer of any parallel server. We further show the relation between optimal threshold on length of output buffer and the total number of jobs in the system.

LEMMA 4 ***(Monotonicity of Queue Length Thresholds w.r.t. Total Number of Jobs in the System):*** *Consider a state* $s \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$*, with a fixed aggregated score* $x \triangleq G^{\mathcal{I}_j}(x_j^1, x_j^2, ..., x_j^m)$*, where* $x_j^i \triangleq [\mathbf{x}_{k_i}^{(i)}]_1 \ \forall i \in \mathcal{I}_j$*, then for any* $i \in \mathcal{I}_j$ *the threshold* $k_i^*(x, n)$ *on the output buffer of Server* i *is nonincreasing in* n $(= [\mathbf{n}]_1 + k_1)$.

Lemma 3 and 4 lead to the step-wise threshold policy for the optimal DJD policy with respect to the total number of jobs in the system and the length of output buffers. The following proposition fully characterises the step-wise threshold policy for any given aggregated score.

PROPOSITION 3 ***(Step-Wise Threshold Policy for Queue Lengths):*** *For a state* $s \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$*, with a fixed aggregated score* $x \triangleq G^{\mathcal{I}_j}(x_j^1, x_j^2, ..., x_j^m)$*, where* $x_j^i \triangleq [\mathbf{x}_{k_i}^{(i)}]_1 \ \forall i \in \mathcal{I}_j$*, the optimal DJD policy has the step-wise threshold structure in* n *and* k_i $(\forall i \in \mathcal{I}_j)$.

The above result fully captures the optimal join decision making for the join server. The join server decides to wait for another evaluation if the number of jobs in the system and the length of output buffer of all servers are below their respective optimal thresholds, i.e., lies within the "wait" region of the policy in Figure 6. Otherwise, the join decision is taken with only m servers' evaluations to predict the job as Type 1 or Type 0.

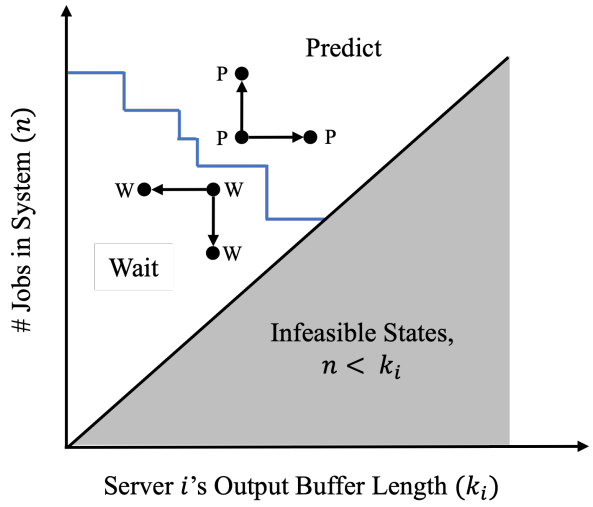

Figure 6: Step-wise threshold policy for a given score x (P: Predict, W: Wait)

Intuitively, the optimal DJD policy tries to restrict the lengths of input and output buffers of the servers with thresholds on each buffer length based on the total jobs in the system. These thresholds, resulting in a step-wise threshold policy, are dependent on the current aggregated score with the join server. Thus, we further characterize the monotonicity of step-wise threshold policy w.r.t. the aggregated evaluation score.

LEMMA 5 ***(Monotonicity of the Optimal Threshold on Total Number of Jobs w.r.t. Aggregated Evaluation Score):*** *Suppose there exists a state* $s \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$, *with an aggregated evaluation score* $x \triangleq G^{\mathcal{I}_j}(x_j^1, x_j^2, ..., x_j^m)$, *where* $x_j^i \triangleq [\mathbf{x}_{k_i}^{(i)}]_1 \ \forall i \in \mathcal{I}_j$, *for which action* $\mathcal{A}_0$ $(\mathcal{A}_1)$ *has lower cost than action* $\mathcal{A}_1$ $(\mathcal{A}_0)$, *then the optimal threshold on the total number of jobs in the system, i.e.,* $n^*(x, (k_u)_{\forall u \in \mathcal{I}_j})$, *is non-decreasing (non-increasing) in* x.

The step-wise threshold policy is described by both queue length thresholds, i.e., the optimal total number of jobs in the system threshold (n^*) as well as the optimal total number of copies in the output buffer threshold (k_i^*). Lemma 5 shows the dependence of n^* w.r.t. the aggregated score, we now show the dependence of k_i^* on the aggregated score x.

LEMMA 6 ***(Monotonicity of the Optimal Threshold on Length of Output Buffer w.r.t. Aggregated Evaluation Score):*** *Suppose there exists a state* $s \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$, *with an aggregated evaluation score* $x \triangleq G^{\mathcal{I}_j}(x^1, x^2, ..., x^m)$, *where* $x^i \triangleq [\mathbf{x}_{k_i}^{(i)}]_1 \ \forall i \in \mathcal{I}_j$, *for which action* $\mathcal{A}_0$ $(\mathcal{A}_1)$ *has lower cost than* $\mathcal{A}_1$ $(\mathcal{A}_0)$, *then for any* $i \in \mathcal{I}_j$, *the threshold* $k_i^*(x, n)$ *on the output buffer of Server* i *is non-decreasing (non-increasing) in* x (here, $n = [\mathbf{n}]_1 + k_1$).

The monotonicity of each queue length threshold in Lemma 5 and 6 can now describe the monotonicity of step-wise threshold policy with respect to the aggregated score x. The following results formalizes the monotonicity of the step-wise threshold policy.

PROPOSITION 4 ***(Monotonicity of Step-wise Threshold Policy w.r.t. Aggregated Evaluation Score):*** *Suppose there exists a state* $s \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_m$, *with an aggregated evaluation score* $x \triangleq G^{\mathcal{I}_j}(x_j^1, x_j^2, ..., x_j^m)$, *where* $x_j^i \triangleq [\mathbf{x}_{k_i}^{(i)}]_1 \ \forall i \in \mathcal{I}_j$, *for which action* $\mathcal{A}_0$ $(\mathcal{A}_1)$ *has lower cost than* $\mathcal{A}_1$ $(\mathcal{A}_0)$, *then the step-wise threshold policy's "wait" region expands (shrinks) in* x. To understand Proposition 4, lets take

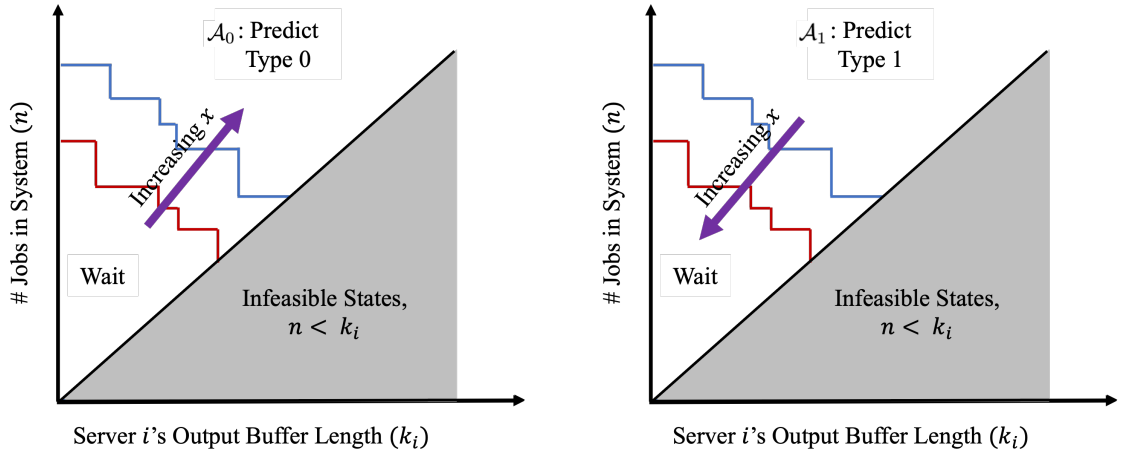

Figure 7: Monotonicity of step-wise threshold policy w.r.t. aggregated evaluation score

the case where the current aggregated evaluation score, from m servers, would result in prediction decision

$\mathcal{A}_0$ (assuming $\mathcal{A}_0$ has lower cost than $\mathcal{A}_1$). As illustrated in the Figure 7, this suggests that the step-wise threshold structure would move away from origin, expanding the wait region if the aggregated score goes away from zero. For instance, given an aggregated evaluation from m servers, the join server may want to wait for another evaluation because the scores from other servers can be very high or low depending upon their knowledge distributions. But, the join server is willing to wait more when the aggregated score is inconclusive (i.e., away from zero and one), whereas, with score very close to zero, waiting is restricted with shrunken "wait" region of optimal DJD policy.

4.7.3 The Optimal Score Thresholds for Prediction

Finally, we analyze the optimal prediction decision when only *some* or *all* of the servers have completed evaluation of their copies for a particular job. The prediction policy determines the optimal prediction between Type 0 and Type 1, when optimal DJD policy dictates not to wait for another evaluation, or when the join server has evaluations from all servers. When the join server has evaluations from servers in set $\mathcal{I}_j$ for a job Z_j, the optimal prediction decision for the join server is as follows:

PROPOSITION 5 ***(Prediction Score Thresholds):*** *For states $s \triangleq (\mathbf{n}, \mathbf{X}) \in \mathcal{S}_N \bigcup \mathcal{S}_m$ in which aggregated evaluations denoted by $x \triangleq G^{\mathcal{I}_j}([(\mathbf{X})_i]_1)_{\forall i \in \mathcal{I}_j}$, there exist threshold $x^{p*}_{\mathcal{I}_j}$ that determines the optimal prediction $A^*(s)$, for the in-process job Z_j as follows:*

$$A^*(s) = \left\{ \begin{array}{ll} \mathcal{A}_0, & : \text{if } x \leq x^{p*}_{\mathcal{I}_j} \\ \mathcal{A}_1, & : \text{otherwise} \end{array} \right\}$$

where, we have the critical fractile value

$$x^{p*}_{\mathcal{I}_j} \triangleq (P_{\mathcal{I}_j})^{-1} \left(\frac{c_0}{c_0 + c_1} \right)$$

and, $\forall\ y \in [0,1]$:

$$(P_{\mathcal{I}_j})^{-1}(y) \triangleq \inf\{x \in [0,1] : P_{\mathcal{I}_j}(x) \geq y\}.$$

Threshold $x^{p*}_{\mathcal{I}_j}$ depends on Type 1 and Type 0 prediction error costs c_1 and c_0, respectively, and on the knowledge of servers in set $\mathcal{I}_j$. We note this result is slightly different from Proposition 1 of Saghafian et al. (2018) since that result holds true for only one evaluation score. In our case, there is an aggregated score using at least m evaluations. Thus, we have a critical fractile for each combination of servers who can provide evaluation score. The set of values $x^{p*}_{\mathcal{I}_j}$ for each subset $\mathcal{I}_j$ of power set of $\mathcal{N}$ defines the prediction policy for the join server. With this threshold, when deciding to complete the job, join server can then make its prediction based on the aggregated score.

We note that the critical fractile is similar in structure to finding order quantities in newsvendor problems. With underage cost c_0, the overage cost c_1 and the demand distribution of $P_{\mathcal{I}_j}$, the critical fractile value

resembles closely with the optimal order quantities. Intuitively, the critical ratio $\frac{c_0}{c_0+c_1}$ suggests that the join server needs a minimum confidence level that a job is a Type-1 job given the evaluations from servers in $\mathcal{I}_j$. If the system suggests a very high Type-0 prediction error cost c_0, when join server incurs high cost of predicting a type-0 job as type-1 job, the critical fractile asks for high aggregated score from the servers to predict any job as Type-1 job. When the aggregated evaluation $G^{\mathcal{I}_j}(x_j^1, x_j^2, ..., x_j^N)$ is small, then Type-0 prediction is optimal, otherwise Type-1 prediction is optimal.

To summarize, recall our two key questions: (1) Should the join server make a decision using the available evaluations, or should it wait for another evaluation? and (2) If the join server decides not to wait, what should be the join server's final prediction for a job (i.e., Type-1 job or Type-0 job)? The optimal DJD policy (characterized in Proposition 1 and 3) answers the first question. The optimal prediction score thresholds in Proposition 5 provide answer to the second question. The optimal dynamic join decision policy helps us find the optimal number of evaluations needed to balance the costs of speed and accuracy. We now motivate a simple heuristic policy which approximates the optimal DJD policy to overcome the curse of dimensionality.

5 Curse of Dimensionality in MDP

Given the structure of the optimal MDP solution, we can solve for exact dynamic join decision policy and the prediction decisions for a given set of parameters using value iteration algorithm. To achieve a desired balance between speed and accuracy, we can solve for the optimal DJD control policy for the join server. But, even if we truncate the sizes of servers' input buffers to A, and their output buffers to B, and if each of N parallel servers can give an evaluation score from a range of C different scores, the state space of MDP grows exponentially as $A \times N \times C^{N \times B}$ (approximately). This curse of dimensionality makes the computation for optimal policy impractical, as shown in Table 1:

(A, B)	$(3, 2)$	$(4, 3)$	$(5, 4)$	$(6, 5)$	$(7, 6)$	$(8, 7)$
MDP State Space Size	580	$6,932$	$77,382$	$830,014$	$8,668,040$	$88,780,584$

Table 1: MDP State Size Growth with $N = 2, C = 3$

Moreover, to implement the MDP solution, the join server uses the plots of step-wise threshold policy (as in Figure 6) for every aggregated score $x_{\mathcal{I}_j}$ from any set of servers $\mathcal{I}_j \subset P(\mathcal{N})$ ($P(\mathcal{N})$: the power set of $\mathcal{N}$). It leads to computation of $B \times C \times |P(\mathcal{N})|$ number of thresholds for implementation of optimal DJD policy. The computation size increases proportional to the size of output buffer and range of the different scores from the servers. These motivate the need for a heuristic approach which is easy to compute and easy to implement. Thus, in the next section, we design a heuristic approach which simplifies the structure of

optimal DJD policy and suggests a simple join decision policy for the join server based on total number of jobs in the system. We will show that the proposed heuristic which we call "Line Heuristic" (LH) performs very close to that of the optimal solution. In the following section, we first define our line heuristic and then describe how to approximate the LH policy using a queueing model.

6 The Line Heuristic Policy

We propose a heuristic that approximates the step-wise structure of the optimal DJD policy (Proposition 3, Figure 6) with a horizontal line. For any given evaluation from servers in set $\mathcal{I}_j$, this horizontal line acts as a threshold on the total number of jobs in the system. The line separates waiting decisions from prediction decision, see Figure 8. As Figure 8 shows, while the optimal DJD policy makes a wait or prediction decision based on the aggregated score x, the total number of jobs in the system (n), and the number of copies in the output buffer of all servers $(k_i)_{\forall i \in \mathcal{I}_j}$, whereas the line heuristic makes such decisions based only on the aggregated score x and the total number of jobs in the system n. The step-wise threshold policy is defined by the optimal queue length threshold $n^*(x, k_i)$ on the total number of jobs in the system as well as $k_i^*(x, n)$ the length of output buffers of each server (Proposition 3). However, the line heuristic need computation of only one threshold $n_x^{\mathcal{I}_j}$ for every aggregated score x from servers in set $\mathcal{I}_j \subset P(\mathcal{N})$.

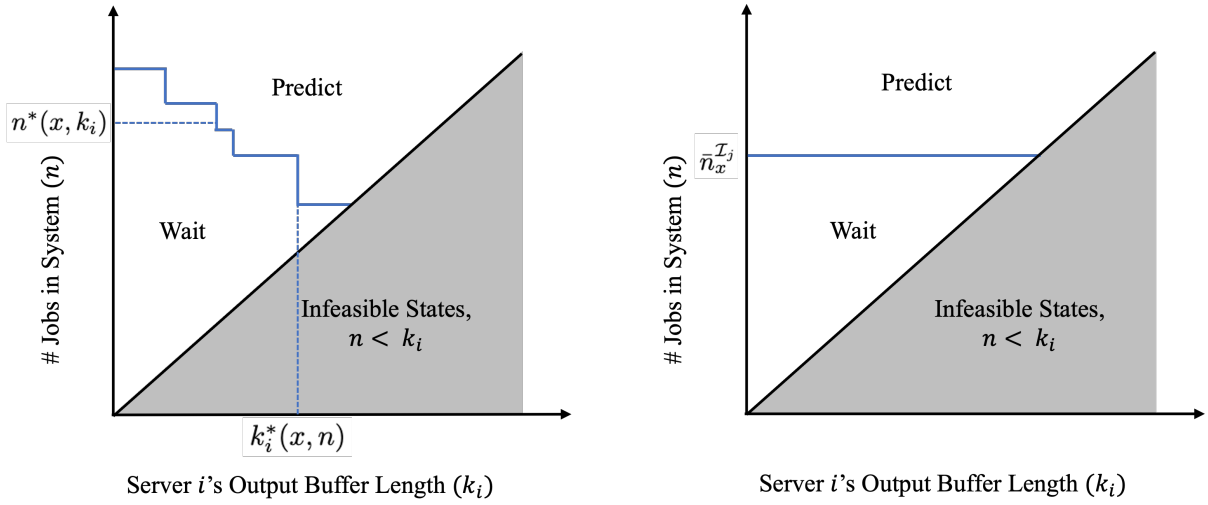

Figure 8: Approximation of Step-wise Threshold Policy for given aggregated score x from $\mathcal{I}_j$ servers

We call this heuristic a Line Heuristic (LH) policy, since it approximates the optimal step-wise threshold policy with a horizontal line. For a given aggregated score x, our LH policy is defined by a set of thresholds $T_x \triangleq \{\bar{n}_x^S \,|\, \forall S \in P(\mathcal{N})\}$. For example, in Figure 8, the threshold $\bar{n}_x^{\mathcal{I}_j}$ suggests join server to complete the job Z_j only when it has aggregated score x from $\mathcal{I}_j$ set of servers and the total number of jobs in the system

are greater than $\bar{n}_x^{\mathcal{I}_j}$. The LH policy is described in Algorithm 1. For every aggregated score x, the set

Algorithm 1: Join server decision making using the Line Heuristic Policy $\mathcal{L}$

Input: Set of LH thresholds T_x for each aggregated score x
Step-1: Identify the total number of jobs n in the system, and the aggregated score x from the servers in set $\mathcal{I}_j$:
if $n < \bar{n}_x^{\mathcal{I}_j}$ **then**
  take action $\mathcal{A}_w$ to wait for additional evaluation.
else if $n \geq \bar{n}_x^{\mathcal{I}_j}$ **then**
  suggest join server to complete the job and predict based on prediction policy, i.e.,
  if $x < x_{\mathcal{I}_j}^{p*}$ **then**
    take action $\mathcal{A}_0$, predict the job as Type-0
  else if $x \geq x_{\mathcal{I}_j}^{p*}$ **then**
    take action $\mathcal{A}_1$, predict the job as Type-1
  end
end

of thresholds T_x fully characterizes the LH policy. Hence, the performance of LH policy depends on the choice of these thresholds. The thresholds are dependent on the aggregated evaluations from the servers and their knowledge distributions. To estimate the LH thresholds, an exhaustive search is very expensive. It is computationally more expensive then finding a solution to MDP. For a given aggregated score x from $\mathcal{I}_j \in P(\mathcal{N})$ servers, we approximate the LH thresholds by using a queueing model.

We now introduce the queueing model to estimate thresholds on total number of jobs waiting in the system. For any given LH policy $\mathcal{L}$, along with the input parameters (i.e., job arrival rate, parallel server and join server's service rates and knowledge distributions), we can estimate the resulting waiting time W (i.e., speed measure) and expected prediction errors $\mathcal{E}$ (i.e., accuracy measure) of the join decision based on the queueing model. It helps estimate the set of LH thresholds with least waiting as well as least errors. In the next section, we numerically show that our approximate LH policy performs very close to the optimal MDP solution.

6.1 Queueing Model to approximate LH Policy

We use a queueing model (similarly in Song et al. (1999)) to estimate the Line heuristic policy thresholds. The queueing model has N parallel servers with their corresponding output queues. Define state $\mathbf{s} = (\mathbf{n}, \mathbf{K})$ where, $\mathbf{n} = (n_1, n_2, ..., n_N) \in \mathbb{Z}^{N+}$, where n_i is the length of input buffer including the one job in service of Server i, and $\mathbf{K} = (k_1, k_2, ..., k_N) \in \mathbb{Z}^{N+}$, where k_i is the length of output buffer of Server i with evaluated copies in it. State space $\mathcal{S}^h \triangleq \{(\mathbf{n}, \mathbf{K}) : n_i + k_i = n_j + k_j, n_i, k_i \geq 0 \ \forall i, j \in \mathcal{N}\}$. In Table 2, we recreate the state space with similar input parameters as in Table 1 to compare the size of $\mathcal{S}^h$ state space with the MDP state space $\mathcal{S}$.

(A, B)	$(3,2)$	$(4,3)$	$(5,4)$	$(6,5)$	$(7,6)$	$(8,7)$
MDP State Space Size	580	6,932	77,382	830,014	8,668,040	88,780,584
Markov Chain States	28	60	110	182	280	408

Table 2: State Size Comparison between MDP and Queueing Model with $N = 2, C = 3$

6.1.1 Underlying Markov Chain

We define the dynamics of the Markov chain for a given set of thresholds from the LH policy $\mathcal{L}$. For any score $x \in [0,1]$, we denote the set of LH thresholds $T_x \triangleq \{\bar{n}_x^S \in \mathbb{Z}^+ \,|\, \forall S \in P(\mathcal{N})\}$. We further restrict the state space with upper bound on input and output buffer size with $n_i \leq \overline{N} \in \mathbb{Z}^+$ and $k_i \leq \overline{K} \in \mathbb{Z}^+,\ \forall i \in \mathcal{N}$. We note that this truncation would restrict the maximum number of the jobs in the system to $M \triangleq \min\{\overline{N}, \overline{K}\}$. For a state $\mathbf{s} = (\mathbf{n}, \mathbf{K}) \in \mathcal{S}^h$, we further define an active set $I^{\mathbf{s}}$ representing the indices of servers whose output buffers are non-empty, i.e., $I^{\mathbf{s}} = \{i : \forall i \in \mathcal{N}\, s.t.\, [\mathbf{K}]_i > 0\}$.

Thus, forming a simplified fork-join queue in which the transitions happen when (1) a simultaneous arrival of the jobs happens to all parallel queues, (2) any server in parallel queues ends its service, and (3) the join server finally removes the job from the system. Hence, with transition rate λ, the state $\mathbf{s} \in \mathcal{S}^h$ transitions to $\mathbf{s}' = (\mathbf{n}', \mathbf{K})$, where

$$n_j' = \begin{cases} n_j + 1 & : \text{ if } n_j < \overline{N}, \forall j \in \mathcal{N} \\ n_j & : \text{ otherwise} \end{cases}$$

with rate μ_i, for $i, j \in \mathcal{N}$, state $\mathbf{s} = (\mathbf{n}, \mathbf{K})$ transitions to state $\mathbf{s}' = (\mathbf{n}', \mathbf{K}')$, where

$$n_j' = \begin{cases} n_j - 1 & : \text{ if } i = j \text{ and } n_j > 0, k_j < \bar{K}, \\ n_j & : \text{ otherwise} \end{cases} \qquad k_j' = \begin{cases} k_j + 1 & : \text{ if } i = j \text{ and } n_j > 0, k_j < \bar{K}, \\ k_j & : \text{ otherwise.} \end{cases}$$

Finally, with rate μ_0 and for $j \in \mathcal{N}$, state $\mathbf{s} = (\mathbf{n}, \mathbf{K})$ with its servers in active set $I^{\mathbf{s}}$ transitions to $\mathbf{s}' = (\mathbf{n}', \mathbf{K}')$, where

$$n_j' = \begin{cases} n_j - 1 & : \text{ if } n_j + k_j \geq \bar{n}_x^{I^{\mathbf{s}}} \text{ and } k_j = 0, \\ n_j & : \text{ otherwise} \end{cases} \qquad k_j' = \begin{cases} k_j - 1 & : \text{ if } n_j + k_j \geq \bar{n}_x^{I^{\mathbf{s}}} \text{ and } k_j > 0, \\ k_j & : \text{ otherwise.} \end{cases}$$

6.1.2 Balance Equations and Steady State Probabilities

We now find the steady state distribution of the Markov chain using the set of LH thresholds $T_x = \{\bar{n}_x^S \in \mathbb{Z}^+ \,|\, \forall S \in P(\mathcal{N})\}$. Since it is a bounded and irreducible Markov Chain, its stationary distribution exists uniquely (Ross (1995)). We can find the balance equations using the above state transition equations with rate transition matrix donated as $\mathbf{R}$. We find a structure in the transitions of the chain. This structure

makes is easier to write the balance equations.

We order states $s \in \mathcal{S}^h$ of the Markov chain such that states are ordered in nondecreasing number of total jobs in the system, and the lexicographic ordering of $\mathbf{K}$ in state is maintained. For instance, in a small system with state $s = (n_1, n_2, k_1, k_2)$, we order as $\{(0,0,0,0), (1,1,0,0), (1,0,1,0), (0,1,0,1), (0,0,1,1), (2,2,0,0), (1,2,1,0), (2,1,0,1), (1,1,1,1), (0,1,2,1), (1,0,1,2), ...\}$. Here, first state has zero jobs, next four size 1 states have one job, after that size 2 states with two jobs and also, the lexicographic ordering of k_1, k_2 is maintained between size 1 and size 2 states. Under this ordering sequence, we denote the rate transition matrix as $\bar{\mathbf{R}}$ and let $\pi = (\pi_s)_{s \in \mathcal{S}^h}$ be the stationary probabilities for states in $\mathcal{S}^h$. Then, the block-tridiagonal matrix $\bar{\mathbf{R}}$ takes the form of

$$\bar{\mathbf{R}} = \begin{pmatrix} \mathbf{A}_0^D & \mathbf{A}_0^R & 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \mathbf{A}_1^L & \mathbf{A}_1^D & \mathbf{A}_1^R & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \mathbf{A}_n^L & \mathbf{A}_n^D & \mathbf{A}_n^R & \dots & 0 & 0 \\ 0 & 0 & \dots & \dots & 0 & \mathbf{A}_{n+1}^L & \mathbf{A}_{n+1}^D & \mathbf{A}_{n+1}^R & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & \dots & \dots & \dots & \mathbf{A}_{M-1}^L & \mathbf{A}_{M-1}^D & \mathbf{A}_{M-1}^R \\ 0 & 0 & \dots & \dots & \dots & \dots & \dots & 0 & \mathbf{A}_M^L & \mathbf{A}_M^D \end{pmatrix}$$

where, $\mathbf{A}_i^L, \mathbf{A}_i^D, \mathbf{A}_i^R$ matrices $\forall\, i \in \{0, 1, 2, .., M\}$ are constructed using the transition equations. These matrices are quite sparse and easily identifiable using the transition equations. To avoid heavy notation, we omit the details of these matrices. However, we provide details for the $N = 2$ system with two parallel servers and state space $\mathbf{s} = (n_1, n_2, k_1, k_2)$ to describe the structure.

Matrix $\mathbf{A}_2^L$ with size of $D_2 \times D_1 = 7 \times 4$ and when $\bar{n}_x^{I^{\mathbf{s}}} > 2$:

$$\mathbf{A}_2^L = \begin{pmatrix} \mathbf{0}_{4\times 4} \\ \cdots\cdots\cdots \\ \mu_0 \mathbb{I}_4 \end{pmatrix}$$

where, $\mathbf{0}_{4\times 4}$ is 4×4 square matrix with all zeros and $\mathbb{I}_4$ is a 4×4 unit matrix with 1 in its diagonal.
Matrix $\mathbf{A}_2^D$ with size of $D_2 \times D_2 = 7 \times 7$:

$$\mathbf{A}_2^D = \begin{pmatrix} \mu_3 - 1 & \mu_2 & \mu_1 & 0 & 0 & 0 & 0 \\ 0 & -\lambda - \mu_1 & 0 & \mu_1 & 0 & 0 & 0 \\ 0 & 0 & -\lambda - \mu_2 & \mu_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & \mu_2 & \mu_1 & 0 \\ 0 & 0 & 0 & 0 & \mu_2 - 1 & 0 & \mu_1 \\ 0 & 0 & 0 & 0 & 0 & \mu_1 - 1 & \mu_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda - \mu_3 \end{pmatrix}$$

Matrix $\mathbf{A}_2^R$ with size of $D_2 \times D_3 = 7 \times 10$:

$$\mathbf{A}_2^R = \left(\begin{array}{c:c} \lambda \mathbb{I}_7 & \mathbf{0}_{7\times 3} \end{array} \right)$$

Therefore, $\pi = (\pi_s)_{s\in\mathcal{S}^h}$, the balance equations $\pi\bar{\mathbf{R}} = 0$ are as follows:

$$\begin{aligned} \hat{\pi}_0 \mathbf{A}_0^D + \hat{\pi}_1 \mathbf{A}_1^L &= 0, \\ \hat{\pi}_{\overline{N}-1} \mathbf{A}_{\overline{N}-1}^R + \hat{\pi}_{\overline{N}} \mathbf{A}_{\overline{N}}^D &= 0, \\ \hat{\pi}_i \mathbf{A}_i^R + \hat{\pi}_{i+1} \mathbf{A}_{i+1}^D + \hat{\pi}_{i+2} \mathbf{A}_{i+2}^L &= 0, \quad \forall\, i \in \{0, 1, 2, ..., M-2\}. \end{aligned}$$

where, $\hat{\pi}_i \triangleq (\pi_{\mathbf{s}})_{\mathbf{s}\in\mathcal{S}^h}$ is defined the row vector of steady state probabilities for states $\mathbf{s} \in \mathcal{S}^h$ that has the system size $n_{\mathbf{s}} = i$, i.e., i is the total of length of input buffer, one in-service job and length of output buffer. Also, the normalization equation implies that:

$$\sum_{\mathbf{s}\in\mathcal{S}^h} \pi_{\mathbf{s}} = 1.$$

We can thus solve for steady state probabilities by using the balance equations and the normalization equation which gives:

$$\hat{\pi}_i = \hat{\pi}_{i-1} \mathbf{G}_i, \; \forall i \in \{1, 2, 3, ..., M\}$$

where, $\mathbf{G}_i = -\mathbf{A}_{i-1}^R (\mathbf{A}_i^D + \mathbf{G}_{i+1} \mathbf{A}_{i+1}^L)^{-1}$ and $\mathbf{G}_M = -\mathbf{A}_{M-1}^R \mathbf{A}_M^D$.

Also using normalization equation we have,

$$\hat{\pi}_0 + \hat{\pi}_0 \sum_{i=1}^{\overline{N}} \Pi_{k=1}^i \mathbf{G}_k = 1$$

With these steady state probabilities, we can to find the performance of the line heuristic with set of thresholds T_x, i.e., measuring the average waiting time of jobs (W) and the expected prediction error cost ($\mathcal{E}$).

6.1.3 Performance Measures

The two important performance measures for our analysis are the average waiting time of jobs in the system and the expected prediction error.

Average Waiting Time (W)*:* Having steady state probabilities for a given set of thresholds $T_x = \{\bar{n}_x^S \in \mathbb{Z}^+ \,|\, \forall S \in P(\mathcal{N})\}$, we can compute the average waiting time (W). The average waiting time also depends on the input parameters such as arrival rate λ, parallel servers service rate $\mu \triangleq (\mu_i)_{i\in\mathcal{N}}$, and the join server service rate μ_0.

Thus, the average waiting time,

$$W(T_x, \lambda, \mu, \mu_0) = \frac{\sum \mathbf{s} \in \mathcal{S}^h \pi_{\mathbf{s}} n_s}{\lambda b} \tag{9}$$

where, $b \triangleq \sum_{\mathbf{s}\in I_b} \pi_{\mathbf{s}}$ is the probability that the system is blocked due to limited input buffer size ($\overline{N}$) and limited output buffer size ($\overline{K}$) where set $I_b \triangleq \{\mathbf{s} \in \mathcal{S}^h \,|\, \max_{i\in\{1,2,\ldots,N\}} [\mathbf{s}]_i \geq \overline{N} \text{ or } \max_{i\in\{N+1,N+2,\ldots,2N\}} [\mathbf{s}]_i \geq \overline{K}\}$, and $n_s \triangleq [\mathbf{s}]_1 + [\mathbf{s}]_{N+1}$ is the number of jobs in the system in state $\mathbf{s}$.

The numerator of equation (9) represents the average number of jobs in the queueing system. The denominator denotes the effective arrival rate into the queueing system, taking into account the blockage. Using Little's Law, equation (9) gives the average waiting time W of using thresholds in set T_x.

Expected Prediction Error Cost ($\mathcal{E}$)*:* For a given aggregated score x from servers in set $\mathcal{I}_j$, LH policy uses thresholds from set T_x. Here, Z_j represents the job being processed by the join server. Therefore, we find the expected prediction cost $\mathcal{E}$ from the queueing model, as:

$$\mathcal{E}(T_x, c_1, c_0, \lambda, \mu, \mu_0, F, x, \mathcal{I}_j) = \sum_{\mathbf{s}\in\mathcal{S}^h} \pi_{\mathbf{s}} E_s$$

where, E_s is the expected prediction cost in state $\mathbf{s}$ of the Markov chain defined as:

$$E_s \triangleq \left\{ \begin{array}{ll} 0, & \forall \mathbf{s} \in \mathcal{S}^h \text{ s.t. } I^{\mathbf{s}} = \emptyset \text{ or, } n_s < \bar{n}_x^{I^{\mathbf{s}}} \\ \min\left\{c_1 P_{\mathcal{I}_j}(x),\, c_0(1 - P_{\mathcal{I}_j}(x))\right\}, & \forall \mathbf{s} \in \mathcal{S}^h \text{ s.t. } I^{\mathbf{s}} = \mathcal{I}_j \text{ and, } n_s \geq \bar{n}_x^{\mathcal{I}_j} \\ \mathbb{E}_{f^{(I^{\mathbf{s}})}|x}\left[\min\left\{c_1 P_{I^{\mathbf{s}}}(X_{I^{\mathbf{s}}}),\, c_0(1 - P_{I^{\mathbf{s}}}(X_{I^{\mathbf{s}}}))\right\}\right], & \forall \mathbf{s} \in \mathcal{S}^h \text{ s.t. } I^{\mathbf{s}} \neq \mathcal{I}_j, \emptyset \text{ and, } n_s \geq \bar{n}_x^{I^{\mathbf{s}}} \end{array} \right\} \tag{10}$$

Recall that, for state $\mathbf{s}$, set $I^{\mathbf{s}}$ was defined as the set of servers with at least one evaluation in their output buffers and $n_{\mathbf{s}}$ was defined as the number of jobs in the system. The top term on the right-hand side of (10) represents zero cost when the join server doesn't make any join decision for jobs with no evaluations or when there are fewer jobs than the threshold on the total number of jobs. The middle term represents the prediction error cost based on Proposition 5 prediction policy when total number of jobs in the system are more than the LH threshold. Proposition 5 effectively suggests choosing a decision which has minimum prediction error cost for a given aggregated score x from $\mathcal{I}_j$ servers. The last term represents the expected

prediction error cost for evaluations from any other set of servers (i.e, $S \neq \mathcal{I}_j$) when the number of jobs are greater than the respective threshold. It calculates the average prediction error cost based on score $X_{\mathcal{I}^s}$ from $I^{\mathbf{s}}$ servers given that servers in set $\mathcal{I}_j$ has an aggregated score of x.

With the overall expected prediction error cost and average holding cost, we can find the performance of set of LH thresholds T_x.

6.1.4 Estimation of LH Policy Thresholds

We have estimated the average waiting time and the expected prediction errors cost of using set of thresholds T_x based on our queueing model. We now propose a step-by-step procedure to completely characterize the line heuristic policy with set of thresholds T_x for every aggregated score x. The thresholds are chosen in order to minimize the total average holding cost and the expected prediction error costs per unit time from the queueing model.

For a given aggregated score x from set of $\mathcal{I}_j$ servers for job Z_j, the total cost of using the thresholds T_x is denoted by $C(T_x, x, \mathcal{I}_j)$, where

$$C(T_x, x, \mathcal{I}_j) \triangleq h(\lambda b)W(T_x, \lambda, \mu, \mu_0, F) + \mathcal{E}(T_x, c_1, c_0, \lambda, \mu, \mu_0, F, x, \mathcal{I}_j) \tag{11}$$

Using the cost function $C(T_x, x, \mathcal{I}_j)$ to evaluate a set of thresholds, we follow the following steps to estimate the optimal set of thresholds $T_x^* = \{\bar{n}_x^{*S} \in \mathbb{Z}^+ \,|\, \forall S \in P(\mathcal{N})\}$ for any aggregated score $x \in (0, 1)$:

Step 1: For the states where the join server has not received any evaluation from parallel servers, i.e., the set of servers with non-empty output buffers is an empty set ($\mathcal{I}_j = \emptyset$), the LH threshold should always suggest waiting for evaluations instead of a prediction decision. Therefore, the LH threshold in the set T_x^*, corresponding to set $\mathcal{I}_j = \emptyset$, $\bar{n}_x^{*\mathcal{I}_j} = M + 1$. Since size of the system is defined by the truncation of input and output buffers $M = \min\{\overline{N}, \overline{K}\}$. The threshold is high enough to allow waiting for at least one evaluation.

Step 2: For the states where the join server has received evaluation from all parallel servers for a job Z_j, i.e., the set of servers with non-empty output buffers $\mathcal{I}_j = \mathcal{N}$. Therefore, the LH threshold in the set T_x^*, $\bar{n}_x^{*\mathcal{I}_j} = 0,\ \forall x \in [0, 1]$, when $\mathcal{I}_j = \mathcal{N}$. This prevents LH policy to incur waiting cost when all the copies of a job Z_j are already evaluated.

Step 3: To compute the cost minimizing set of LH threshold $\bar{n}_x^{*\mathcal{I}_j}$ for a given aggregated score x from $\mathcal{I}_j$ set of servers, we start with any set T_x and use our queueing model's rate transition matrix to find its steady state probabilities. Based on equation (11), we get an estimate for total average holding cost and the expected prediction error cost, $C(T_x, x, \mathcal{I}_j)$.

Step 4: For every $\mathcal{I}_j \in P(\mathcal{N}) \setminus \{\emptyset, \mathcal{N}\}$, we now find the optimal LH threshold $\bar{n}_x^{*\mathcal{I}_j}$. Therefore, we denote:

$$(\tilde{n}_x^S)_{\forall S \in P(\mathcal{N})} \triangleq \arg\min_{\forall T_x} C(T_x, x, \mathcal{I}_j)$$

Hence, the optimal LH threshold is $\bar{n}_x^{*\mathcal{I}_j} \triangleq \tilde{n}_x^{\mathcal{I}_j}$. Since we estimate the LH threshold from the set T_x^* when the aggregated score x is given by only the servers in set $\mathcal{I}_j$.

The above procedure iteratively computes for the set of LH thresholds T_x^* for any given aggregated score x. The computation time to look for best threshold depends only on the size of truncation $\overline{N}, \overline{K}$ of input and output buffer queues of the queueing model. Step 4 iterates to the order of set size $|P(\mathcal{N})|$. Thus, the queueing model based computation of LH thresholds leads to a faster solution as compared to repeatedly solving the huge markov decision process for an exhaustive search. We further present the numerical study of our Line heuristic policy $\mathcal{L}$ with the estimated thresholds T_x^* and compare it to the optimal MDP solution.

7 Numerical Study

We now present the numerical study to evaluate the performance of our heuristic policy. We design an experiment to cover a wide range of parameter inputs. Each parameter is designed to take on Low, Medium, or High value. Specifically,

- The range on truncation size for the computations: $(\overline{N}, \overline{K}) \in \{(3,2),(5,4),(6,5)\}$
- Number of parallel servers: $N \in \{2,4\}$
- The ratio of per unit Type 1 prediction error cost to holding cost: $c_1/h \in \{0.5, 5, 50, 500\}$
- Similarly, the ratio of per unit Type 0 prediction error cost to holding cost: $c_0/h \in \{0.5, 5, 50, 500\}$
- For different utilization levels, the ratio of arrival rate to minimum of the servers' service rate: $\lambda/\min\{\mu_i\}_{\forall i \in \mathcal{N}} \in \{0.5, 0.75, 0.95\}$.

Utilization	Low - 50%	Medium - 75%	High - 90%
(λ, μ_1, μ_2)	$(0.1084, 0.2166, 0.225)$	$(0.0915, 0.121, 0.3375)$	$(0.0595, 0.063, 0.4275)$

Table 3: Range on the values of (λ, μ_1, μ_2)

- Two cases for server knowledge distributions: {Faster server is less knowledgeable, Faster server is more knowledgeable} where for less knowledge server, the difference in the mean of its both Type-1 and Type-0 Beta distributions is 1.06 and for high knowledge server, the difference is 1.6.
- True probability of Type-1 job in system: $p \in \{0.25, 0.5\}$

This provides a total number of 672 examples for our numerical study.

7.1 Performance of Line Heuristic

We compare the performance of the Line heuristic (LH) with that of the optimal MDP as well as the following three benchmark policies:

1. *"Always Predict" AP Policy:* Under AP policy, the join server never waits and always makes a prediction decision as soon as it receives the first evaluation from a server who processes a copy of a job first. AP policy results in the fastest speed, i.e., lowest waiting time since the join server never waits for additional evaluations. On the other hand, it results in the lowest accuracy. Since the join server makes prediction decision with only one evaluation. In liver or kidney transplantation, the candidates wait under multiple queues and are being served based on AP Policy (Merion et al. (2004)).

2. *"Always Wait" AW Policy:* Under AW policy, the join server makes a prediction decision only when it receives all evaluations, i.e., after all servers finish working on their copy of the job. Thus, AW policy results in the highest accuracy but the lowest speed. The AW policy has been in practice by social media platforms for third-party fact-checking (Liu et al. (2022), Halimeh et al. (2017)) where they wait for evaluations from all the fact-checkers before making a final decision for the posted content.

3. *Static Policy:* Our line heuristic is a dynamic heuristic policy as the number of evaluations needed for the join decision varies depending on the set of evaluations available with the join server. A static policy always makes a prediction decision when it receives exactly m evaluations from the servers. We note AP and AW are also static policies with $m = 1$ and $m = N$, respectively. We investigate the benefit of choosing a dynamic heuristic policy over an "optimal" static join decision policy, i.e., a static policy with m^* resulting in minimum cost among all static policies with parameter m. In cloud computing systems (Joshi et al. (2015)), we see application of static policies, where a online content download request is replicated on multiple machines and any m completions are sufficient.

We now present a summary of our results for all 672 examples in Table 4. We measure the performance of the heuristics based on three metrics (i) We compare the percent error gaps of each heuristic from the optimal MDP solution. We find for MDP solution using value iteration algorithm for the given input parameters. We particularly look at the average optimality gap of the heuristic from the MDP solution when the gap is non-zero, (ii) we check the maximum percentage gap of the heuristic from the optimal solution, and (iii) we measure the number of cases for which the percentage gap stays within 1% for each of the heuristics. In the Table 4, AP is Always Predict Policy, AW is Always Wait Policy, LH is Line Heuristic policy, and Static: a static policy m with minimum error gap.

Observation 1: *Our Line heuristic policy performs very close to optimal MDP solution.*

Performance Metric	AW	AP	Static	LH
Average % Gap (for non-zeros)	250.64	33.6	2.75	1.03
Maximum % Gap	869.4	1259.2	38.7	22.6
# of Cases with Gap $< 1\%$	21	591	591	634

Table 4: Optimality gap of heuristic policies

The results from Table 4 shows close approximation to the optimal MDP solution with our line heuristic policy with an average of 1.1% error gap between the two. Since the line heuristic approximates step-wise threshold structure of optimal DJD policy with a horizontal line, it is able to simplify the solution structure without compromising on the performance. It helps decision makers to use only the total number of jobs in the system to make prediction or wait decision. The solution resembles with N-Policy for a single server queue where a server waits until queue builds up to N before it starts the service (Yadin and Naor (1963)). Whereas, our LH policy suggests waiting to predict a job until the queue builds up to the specified LH threshold. The total number of jobs in the system becomes an important lever to balance the speed and accuracy trade-off in complex fork-join queues.

Performance Metric	AW	AP	Static	LH
Average % Gap (for non-zeros)	106.8	125.5	6.33	3.1
Maximum % Gap	340.1	1259.2	38.7	22.6
# of Examples with $< 1\%$ Gap	9	47	47	48

Table 5: Summary of 72 examples with High Prediction Costs (i.e., $c_1/c_0 \in \{50, 500\}$) and high utilization (i.e., $> 90\%$)

Observation 2: *Our LH policy always performance better than all other benchmark policies, even for its worst performance under the high prediction costs and high utilization.*

Table 4 shows that the LH policy performs better than the other benchmark policies since their gap from optimal MDP solution is larger than 1.03%. In fact, LH performance is better than other benchmark policies for every scenario in all three performance metrics. Although, as in Table 5, with high prediction costs and high utilization, Line heuristic is relatively away from the optimal solution with 3.1% average error gap higher than its overall average of 1.03%. Also, the maximum error gap of 22.6% is realized in the case of high prediction costs and high utilization. But LH still stays best among all other policies. If we look closely, LH policy's gap from the optimal MDP solution increase for the cases when Type 1 jobs have unequal distribution of $p = 0.25$. Due to high prediction costs, LH tries to wait for additional evaluations when the aggregated evaluation with join server suggests Type 1 prediction decision. Since utilization is also high, it incurs huge costs. Given that, LH still performs better than all other alternatives. We relate this to social media platform's practice of employing AW policy in decision making when more than 1 billion stories are posted by their users every day and they sometimes employ only around 100 fact-checkers (Halimeh

et al. (2017)), i.e., a high utilization system for fact-checking process. AP policy may not be advisable over current AW policy since the cost of calling a false news as not-false incurs high costs for society. Thus implementation of LH policy can help accelerate the fact-checking process.

Performance Metric	AW	AP	Static	LH
Average % Gap (for non-zeros)	42.18	2.9	2.92	0.62
Maximum % Gap	135.1	19.6	13.9	7.4
# of Examples with $< 1\%$ Gap	12	40	40	82

Table 6: Summary of 96 examples with high prediction costs and low utilization levels

Observation 3: *There is a benefit to use dynamic join policy as compared to a static join policy.*

From Table 4, Table 5 and Table 6, the results suggest that there is an at least 2% error gap improvement in going from a static to a dynamic policy which is our line heuristic. Moreover, the maximum error gaps realised, in Table 5 and 6, is significantly better for the line heuristic policy. These results suggest that under high prediction costs examples, a line heuristic based dynamic policy to control join decisions in fork-join queues provides significant benefits to its performance as compared to standard static policies. The service systems under high risk decisions, where an incorrect prediction incurs high cost (for e.g., in fact-checking or ED patient treatment), we note the importance of using dynamic policy like Line heuristic to improve on the speed as well as the accuracy.

Performance Metric	AW	AP	Static	LH
Average % Gap (for non-zeros)	161.7	0.15	0.15	0.07
Maximum % Gap	321.7	0.5	0.5	0.13
# of Examples with $< 1\%$ Gap	0	216	216	216

Table 7: Summary of 216 examples with Low Prediction Costs and low utilization levels

Observation 4: *Always Wait policy is not that bad under high utilization scenario. But, even under high prediction cost and low utilization scenario, Always Wait is not the best policy.*

From Table 5, Table 6 and Table 7, we note that AW policy performs better than its overall performance in Table 4. Particularly in Table 5, we might expect for policy like AW which builds up queues to perform worse but high prediction costs demanded more accurate decisions thus making AW perform better than its overall performance. It also performs better than the AP policy. Whereas, for Table 6, with high prediction costs and low utilization scenario, AW policy performs its best compared to other scenarios, since it doesn't get holding cost penalty due to low utilization and higher costs prefers more evaluations for the join decision. Although even for high costs and low utilization examples, AW is not the best among the other heuristics. This makes AW policy relevant for high risk settings like ED patient diagnosis for further treatments given the ED goes through light traffic as suggested in Carmeli et al. (2023) within an Israeli ED. Even for U.S. hospital emergency times, we find a ED resource utilization level of 85% during COVID pandemic and lower

levels in-general (*US Hospitals* (2022)).

We numerically find close approximation of line heuristic to the optimal MDP solution. LH policy is helpful since it is based on the total number of jobs in the system and easy to compute. We now show how a decision maker can use line heuristic based control policies to achieve a desired speed and accuracy trade-off.

7.2 Speed-Accuracy Trade-off Curve

Until now we used a weighted combination of total number of jobs waiting in the system objective (i.e., speed) and expected prediction errors objective (i.e., accuracy). The weighted objective function problem helped to solve for the structure of optimal join decision control policy. To balance the speed and accuracy trade-off, some managers would use a different approach to optimize speed (or, accuracy) given a service level on accuracy (or, speed). For e.g., in data-centers with fork-join processes, decision makers employ service levels like 99.9th percentile's response time of 200 ms, i.e., only one out of 1000 requests should experience a response time greater than 200 ms (Nguyen et al. (2020)), or in fake news detection for social media platforms, with a service level on misinformation to balance the engagement and misinformation trade-off (Candogan and Drakopoulos (2020)). The decision makers want to align their operational decisions to achieve these service levels.

Using notations from section 4.5, we define two decision problems, each optimizing for one of the speed or accuracy objectives. Decision problem 1 corresponds to the case where the manager looks for optimal control policy $\bar{\pi}$ that minimizes the long-run average waiting time of the jobs subject to the constraint on long-run expected number of prediction errors ($\mathcal{E}^{\alpha}$). The service level on accuracy is denoted by $\mathcal{E}^{\alpha}$.

$$\min_{\bar{\pi}\in\Pi} \liminf_{t\to\infty} \frac{1}{t\lambda}\left[\int_0^t [N_i^{\bar{\pi}}(u) + N_i^{s\bar{\pi}}(u)]\mathrm{du}\right] \tag{12}$$

$$\text{s.t. } \liminf_{t\to\infty} \frac{1}{t}\mathbb{E}\left[\mathcal{E}_0^{\bar{\pi}}(t) + \mathcal{E}_1^{\bar{\pi}}(t)\right] \leq \mathcal{E}^{\alpha}$$

For the objective value, we use Little's law to obtain long-run average waiting time from the average total number of jobs in the system by using the arrival rate λ in the denominator.

The decision problem 2 corresponds to the other case where the manager minimizes the long-run expected number of prediction errors subject to the constraint on long-run average waiting time of jobs (W^{α}). The service level on speed is denoted by W^{α}. We note that it is equivalent to having a service level on the long-run average total number of jobs in the system, using Little's law we can scale the service level with arrival rate λ.

$$\min_{\bar{\pi}\in\Pi} \liminf_{t\to\infty} \frac{1}{t}\mathbb{E}\left[\mathcal{E}_0^{\bar{\pi}}(t) + \mathcal{E}_1^{\bar{\pi}}(t)\right] \tag{13}$$

$$\text{s.t. } \liminf_{t\to\infty} \frac{1}{t\lambda}\left[\int_0^t [N_i^{\bar{\pi}}(u) + N_i^{s\bar{\pi}}(u)]\text{du}\right] \leq W^{\alpha}$$

To solve the two decision problems, it is computationally very challenging to look for an optimal MDP solution which satisfies the service level constraints. Moreover, in certain service systems, the accurate estimates of weights may not be available. It sometimes difficult to exactly quantify the costs of delaying the prediction decision, i.e., delaying the treatment of patient while waiting for diagnosis results, or even, the cost of making an in-accurate decision, i.e., cost of calling a low risk patient as high risk in emergency units. Thus, we present a cost independent analysis through trade-off curve based analysis.

In practice, many decision makers use trade-off curves to achieve service level goals. Since we have shown that our LH policy performs very close to the optimal MDP solution. We now propose to use our line heuristic policy to design join decision control policies under performance service levels. The design of LH policy enables an easy computation of a trade-off curve.

For the given knowledge distributions, service times of servers and the line heuristic policy, we can quantify the average waiting time of a job in the system and also, the expected percent errors. We get the speed and accuracy trade-off curve using the pareto optimal frontier of plotting all line heuristic policies with average waiting time of jobs on one axis and the expected percent errors on the other. Since, the line heuristic is general case of previously proposed benchmark policies, the pareto optimal curve includes Always Wait (WH) heuristic and Always Predict (AP) heuristic as its two extreme points.

Figure 9 shows the plot for an example with $N = 2, \lambda = 0.12, \mu_1 = 0.14, \mu_2 = 0.2$ and $p = 0.25$. The blue dots represent different pareto optimal LH policies whereas the grey dots corresponds to LH policies for all possible thresholds. The purple and orange dot show the two extreme special cases of line heuristics, i.e., always predict policy and always wait policy, respectively. Therefore, the red line symbolizes the pareto optimal frontier of the LH policies. We note that the accuracy decreases when we lower the speed and vice-versa. Also, closer the curve to the origin, the better it performs on both speed and accuracy parameter.

We illustrate the use of the pareto optimal trade-off curve to solve the two decision problems. Under problem 1, the decision maker looks for optimal join decision control policy that minimizes long-run average waiting time given the service level on percent errors of $\mathcal{E}^{\alpha}$ and vice-versa for problem 2. In figure 10, the left hand side plot solves for the optimal LH policy that achieves a service level on percent errors, shown in green triangle. To solve problem 2, the right hand side plot looks for optimal join decision policy that achieves service level on average waiting time while minimizing the percent errors.

In Figure 10, the red star shows the optimal MDP solution based on using cost parameters with holding cost $h = 1$ (related to waiting) and prediction error costs $c_1, c_0 = 33.05$ (related to accuracy). If a decision maker were to achieve an average waiting time of less than 9 then its computationally very hard to find

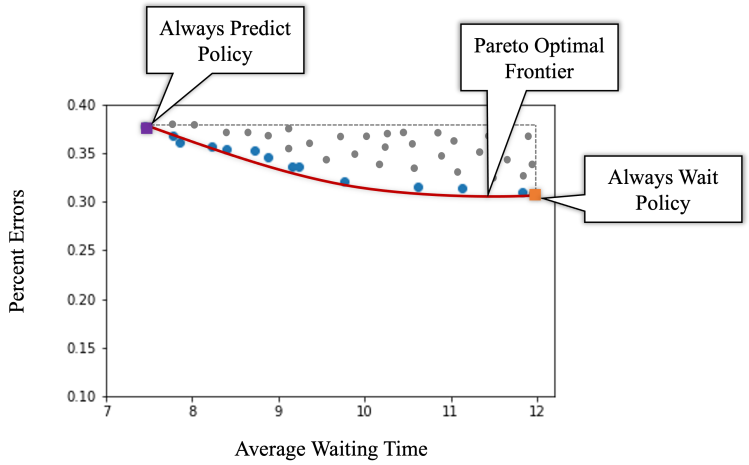

Figure 9: Pareto Optimal Frontier out of all Line Heuristics

corresponding holding and prediction error costs to find a percent error minimizing decision policy. Whereas, a decision maker can easily find a LH heuristic based decision policy for the average waiting time of jobs service level.

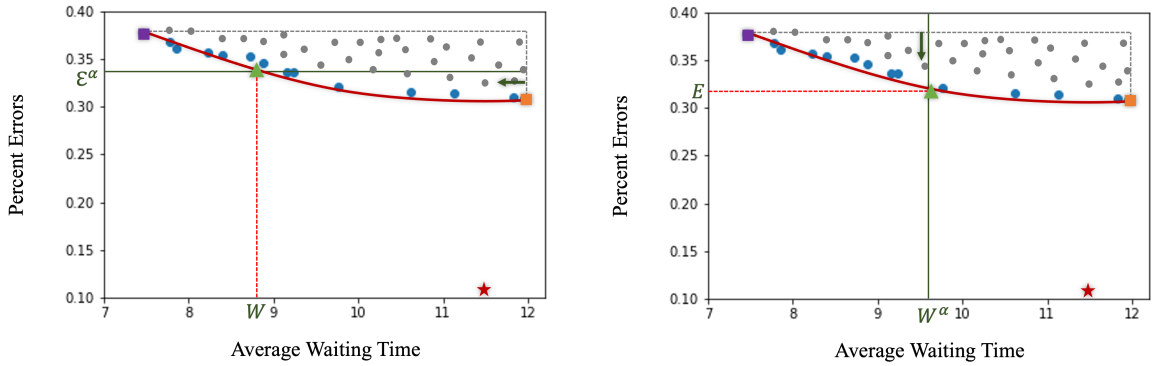

Figure 10: Finding control policy for service level based decision making

The pareto trade-off curve helps manager's to choose a decision policy based on the different service levels on speed and accuracy for the system. This gives a simple rule for managers to act in a seemingly complex fork-join queueing system. The manager can decide to choose a service level on average errors (or average waiting time) between the two extremes of the pareto curve, i.e., between the two benchmark AW and AP policies and hence, look for a line heuristic policy that minimizes the other objective.

8 Conclusion

We modelled the join decision for fork-join knowledge-based service systems to capture the speed and accuracy trade-off. We particularly looked at fork-join queues with parallel knowledge servers who provide

evaluation for copies of same job and the join server making the join decision for a job based on evaluations from *all* or *some* of the servers. To control the join decision policy, we answer two important questions for the join server in fork-join queues: (1) how many evaluations are optimal for the join decision? and (2) what should be the prediction decision for the job which can be of either Type 1 or Type 0? We use MDP-based framework to find optimal dynamic join decision policy. The solution minimizes the waiting cost of any job in the system and also penalizes the in-accurate final prediction decisions. We note that the true type of the jobs are unknown, thus its imperative to have a descriptive model of quantifying accuracy. We model the knowledge distributions of each server with a beta distribution. A series of analytical results completely characterises the structure of optimal DJD policy for join server. Finally, we generate insights into the solution with the help of our line heuristic. This line heuristic is shown to perform very close to optimal solution and yet, easy to compute and understand.

The optimal DJD policy is characterised by a double-threshold policy on aggregated evaluation score from the servers and a step-wise threshold policy on total number of jobs as well as on the number of copies in the output buffers of each server. Further, the prediction decision follows a newsvendor like critical fractile threshold policy to predict between Type 1 or Type 0 for a job with aggregated evaluation score with the join server. The line heuristic policy simplifies the structure of step-wise threshold structure of optimal DJD policy. It suggests the join server to wait for additional evaluation only when there are less total number of jobs in the system than a LH policy threshold. This threshold is calculated based on a simpler queueing model.

This proposed optimal dynamic join decision policy can help accelerate the third-party fact-checking for social media platforms, help in early diagnosis of patients in emergency unit and hence, decrease the load for testing units etc. The decision makers can employ service level based decision making using our LH policy. LH based pareto optimal curve helps to balance the speed and accuracy trade-off and find a control policy to achieve a desired trade-off.

Fork-join queues can further be looked from the point of view "fork" node, where the allocation of a incoming jobs to the parallel server is decided. It can be useful to investigate sending jobs to few of the servers to avoid over-redundancy in the system. Also, it would interesting to model strategic actions of servers when it is beneficial for them to hide the true type of the job. In that setting, to achieve similar level of accuracy might require additional evaluations to remove "non-alignment" bias. Particularly, one can explore learning of type of a job in the system using reinforcement learning techniques and evaluate the performance of heuristic policies.

References

Alizamir, Saed, Francis De Véricourt, and Peng Sun. 2013. "Diagnostic accuracy under congestion." *Management Science* 59 (1): 157–171.

Alizamir, Saed, Francis de Véricourt, and Peng Sun. 2022. "Search under accumulated pressure." *Operations Research* 70 (3): 1393–1409.

Anand, Krishnan S, M Fazıl Paç, and Senthil Veeraraghavan. 2011. "Quality–speed conundrum: Trade-offs in customer-intensive services." *Management Science* 57 (1): 40–56.

Atar, Rami, Avishai Mandelbaum, and Asaf Zviran. 2012. "Control of fork-join networks in heavy traffic." In *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton),* 823–830. IEEE.

Avaaz. 2020. *How Facebook can flatten the curve of the coronavirus infodemic.*

Canale, Damiano. 2021. "The Opacity of Law: On the Hidden Impact of Experts' Opinion on Legal Decision-making." *Law and Philosophy* 40 (5): 509–543.

Candogan, Ozan, and Kimon Drakopoulos. 2020. "Optimal signaling of content accuracy: Engagement vs. misinformation." *Operations Research* 68 (2): 497–515.

Carmeli, Nitzan, Galit B Yom-Tov, and Onno J Boxma. 2023. "State-dependent estimation of delay distributions in fork-join networks." *Manufacturing & Service Operations Management* 25 (3): 1081–1098.

Chan, Carri W, Galit Yom-Tov, and Gabriel Escobar. 2014. "When to use speedup: An examination of service systems with returns." *Operations Research* 62 (2): 462–482.

Delasay, Mohammad, Armann Ingolfsson, Bora Kolfal, and Kenneth Schultz. 2019. "Load effect on service times." *European Journal of Operational Research* 279 (3): 673–686.

Feizi, Arshya, Agni Orfanoudaki, Soroush Saghafian, and Nicole Hudgson. 2023. "Vertical Patient Streaming in Emergency Departments." *Available at SSRN 4465161.*

Gardner, Kristen, Mor Harchol-Balter, Alan Scheller-Wolf, Mark Velednitsky, and Samuel Zbarsky. 2017. "Redundancy-d: The power of d choices for redundancy." *Operations Research* 65 (4): 1078–1094.

Halimeh, Ahmed Abu, Pardis Pourghomi, and Fadi Safieddine. 2017. "The Impact of Facebook's News Fact-Checking on Information Quality (IQ) Shared on Social Media." In *ICIQ.*

Hopp, Wallace J, Seyed MR Iravani, and Gigi Y Yuen. 2007. "Operations systems with discretionary task completion." *Management Science* 53 (1): 61–77.

Isola, Justine. 2018. "Symposium Cybersecurity, Fake News & Policy: Dis-and Mis-Information." *HASTINGS LAW JOURNAL* 69 (5): 1333–1338.

Joshi, Gauri, Yanpei Liu, and Emina Soljanin. 2012. "Coding for fast content download." In *2012 50th Annual allerton conference on communication, control, and computing (Allerton),* 326–333. IEEE.

Joshi, Gauri, Emina Soljanin, and Gregory Wornell. 2015. "Queues with redundancy: Latency-cost analysis." *ACM SIGMETRICS Performance Evaluation Review* 43 (2): 54–56.

———. 2017. "Efficient redundancy techniques for latency reduction in cloud systems." *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 2 (2): 1–30.

Kelly, T.C. 2022. "Can Jurors Evaluate the Reliability of Expert Evidence?" *Expert Pages.*

Liu, Zhonghao, Chaithanya Bandi, and Seyed Iravani. 2022. "Robust Fork Join Queue and Application in Fact Checking."

Lyons, Tessa. 2018. "Hard questions: What's Facebook's strategy for stopping false news." *Facebook newsroom* 23:2018.

Merion, Robert M, Mary K Guidinger, John M Newmann, Mary D Ellison, Friedrich K Port, and Robert A Wolfe. 2004. "Prevalence and outcomes of multiple-listing for cadaveric kidney and liver transplantation." *American Journal of Transplantation* 4 (1): 94–100.

Nguyen, Minh, Sami Alesawi, Ning Li, Hao Che, and Hong Jiang. 2020. "A black-box fork-join latency prediction model for data-intensive applications." *IEEE Transactions on Parallel and Distributed Systems* 31 (9): 1983–2000.

O'Sullivan, Jack W, Ali Albasri, Brian D Nicholson, Rafael Perera, Jeffrey K Aronson, Nia Roberts, and Carl Heneghan. 2018. "Overtesting and undertesting in primary care: a systematic review and meta-analysis." *BMJ open* 8 (2): e018557.

Özkan, Erhun. 2022. "Control of Fork-Join Processing Networks with Multiple Job Types and Parallel Shared Resources." *Mathematics of Operations Research* 47 (2): 1310–1334.

Özkan, Erhun, and Amy R Ward. 2019. "On the control of fork-join networks." *Mathematics of Operations Research* 44 (2): 532–564.

Raaijmakers, Youri, Sem Borst, and Onno Boxma. 2023. "Fork–join and redundancy systems with heavy-tailed job sizes." *Queueing Systems* 103 (1-2): 131–159.

Ross, Sheldon M. 1995. *Stochastic processes.* John Wiley & Sons.

Saghafian, Soroush, Wallace J Hopp, Seyed MR Iravani, Yao Cheng, and Daniel Diermeier. 2018. "Workload management in telemedical physician triage and other knowledge-based service systems." *Management Science* 64 (11): 5180–5197.

Song, Jing-Sheng, Susan H Xu, and Bin Liu. 1999. "Order-fulfillment performance measures in an assemble-to-order system with stochastic leadtimes." *Operations Research* 47 (1): 131–149.

Sun, Zhankun, Nilay Tanık Argon, and Serhan Ziya. 2018. "Patient triage and prioritization under austere conditions." *Management Science* 64 (10): 4471–4489.

U.S. Hospitals Under Strain as ER Wait Times Lengthen. 2022, October. Accessed February 23, 2023.

Varki, Elizabeth, Arif Merchant, and Hui Chen. 2008. "The M/M/1 Fork-Join Queue with Variable Sub-Tasks" (January 1, 2008).

Varma, Subir, and Armand M Makowski. 1994. "Interpolation approximations for symmetric fork-join queues." *Performance Evaluation* 20 (1-3): 245–265.

Wang, Xiaofang, Laurens G Debo, Alan Scheller-Wolf, and Stephen F Smith. 2010. "Design and analysis of diagnostic service centers." *Management Science* 56 (11): 1873–1890.

Yadin, Micha, and Pinhas Naor. 1963. "Queueing systems with a removable service station." *Journal of the Operational Research Society* 14:393–405.