Dynamic Routing Problems with Delayed Information

Esa Hyytiä[1(✉)] and Rhonda Righter[2]

[1] Department of Computer Science, University of Iceland, Reykjavik, Iceland
esa@hi.is
[2] Department of Industrial Engineering and Operations Research, University of California Berkeley, Berkeley, CA, USA
rrighter@berkeley.edu

Abstract. The problem of routing jobs to parallel servers is known as the dispatching problem. A typical objective is to minimize the mean response time, which according to Little's result is equivalent to minimizing the mean number in the system. Dynamic dispatching policies are based on information about the state of each server. In large or real-time systems, up-to-date and accurate system state may not be available to dispatcher. We consider, cases where state information at some time in the past is available, or completed jobs are acknowledged after some propagation delay, and give efficient dispatching policies based on the incomplete state information. The dynamic dispatching policies tailored to this setting are evaluated numerically.

Keywords: Job dispatching · Delay-aware policy · JSQ · SED

1 Introduction

We consider parallel server systems known as dispatching systems under a setting of imperfect state information. Dispatching systems comprise a dispatcher and a pool of parallel servers. New jobs arrive to the dispatchers, which then routes them to different servers so as to balance the load or to minimize the mean response time. Dispatching policies such as join-the-shortest-queue (JSQ) expect the exact and current number in queue from all servers [6], which we refer to as perfect state information. With perfect state information, JSQ is often the optimal policy with respect to mean response time, e.g., with homogeneous exponential service times [16].

In contrast, we study a scenario where the dispatcher has imperfect information about the current state of each server. For example, in large or real-time systems, it may not be feasible to query the state of each server prior too each routing decision. In our setting, servers inform the dispatcher about state changes. However, such reports are not immediately available to the dispatcher, but are delayed [7]. Note that applying the basic JSQ without taking the delays

Q. Zhao and L. Xia (Eds.): VALUETOOLS 2021, LNICST 404, pp. 171–184, 2021.
https://doi.org/10.1007/978-3-030-92511-6_11

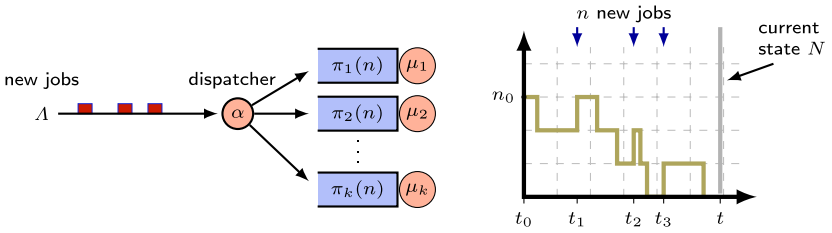


Fig. 1. Dispatching system with imperfect information (left) and evolution of the number in the server since the last status update at time t_0 (right).

into account can even lead to oscillations in the number at servers [12]. Finding optimal policies is intractable for the general model. Artiges studies properties of an optimal policy for two servers using MDP's [3]. Litvak and Yechiali [8] consider the trade-off between waiting for information before routing and routing without information, and Altman et al. [2] study a discrete-time version with deterministic service times. Most recent work on routing with delayed information has focused on asymptotic regimes [11,13,15]. We focus on analyzing heuristics.

In general, the dispatching decision can be based on different amounts of information:

1. **No information**. In this case, the server must implement a static Bernoulli split (RND) policy that may be based on the arrival rate Λ and service rates μ. In particular, in the heterogeneous case where $\mu_i \neq \mu_j$ for some i, j, one can, e.g., balance the load by routing a job to server i with probability $p_i = \mu_i / \sum_j \mu_j$. Optimal splitting probabilities can also be determined [4].
2. **Routing history** refers to the case where the dispatcher records past routing decisions, but gets no feedback from the servers. In this scenario one often implements the Round-Robin (RR) policy, or a weighted variant in the case of heterogeneous servers. RR is often the optimal policy in this type of setting [5, 9,10]. Consequently, whenever the state information is unclear, e.g., because the feedback loop is slow or acknowledgements are simply unreliable, RR can be expected to be near-optimal.
3. **Perfect information** means that the dispatcher knows the number at each server exactly. For example, the well-known JSQ and SED (Shortest Expected Delay) policies generally assume this (see Sect. 3.1).
4. **Delayed information** means that the dispatcher knows the routing history and the number at each server at some time instant in the past. Consequently, it can determine the distribution for the state at each server. This scenario is depicted in Fig. 1.

Our focus in this paper is the last scenario 4). In scenarios 1) and 2), less information is available, and thus the corresponding policies can be readily applied also in our setting. In contrast, dispatching policies based on perfect information, as assumed in scenario 3), serve as lower bounds.

2 Server State Known in the Past

Let us first consider a single server and its state distribution at some time t. Suppose that the available information is the exact number of jobs at the server at time t_0, $t_0 \leq t$, after which n jobs have been routed to the server, at time instants $t_1, \ldots, t_n$, such that $t_0 < t_1 < \ldots < t_n \leq t$. The unknown current number at the server is denoted by N. The situation is depicted in Fig. 1. The thick yellow curve depicts *one* possible sample path. Clearly, $0 \leq N \leq n_0 + n$.

To summarize, we assume that the dispatcher knows the following:

1. At time t_0 server had exactly n_0 active jobs. This is the most recent status update. Note that, because of the exponential services and Poisson arrivals, earlier updates can be ignored.
2. Since then, n new jobs have been routed to the server at time instants $t_1, \ldots, t_n$ (routing history is known)
3. Service times are i.i.d., $X \sim \text{Exp}(\mu)$ (μ is assumed to be known/learned)

First we make an observation regarding the exponential service times.

Remark 1: *The G/M/1 queue can be seen as a system where the server runs an exponential timer that triggers at rate μ. Every time the timer goes off one job is released if present. If the system was empty, the server "shoots a blank".*

The information about the state at time t_0 could also be a distribution, but for now we assume that status updates have no errors. The first task is to determine the state probability distribution at time $t = t_{n+1}$ based on this information. Let $\Delta_i = t_i - t_{i-1}$ and $D_i \sim \text{Poisson}(\mu\Delta_i)$, where D_i denotes the number of potential departures during the time interval Δ_i. Let N_1 denote the number in the system immediately after time t_1, for which it holds that

$$N_1 = 1 + (n_0 - D_1)^+, \quad \text{where } 0 < N_1 \leq n_0 + 1,$$

and $(x)^+ := \max\{0, x\}$. Similarly, immediately after time t_j, $j = 2, \ldots, n$,

$$N_j = 1 + (N_{j-1} - D_j)^+, \quad \text{where } 0 < N_j \leq n_0 + j.$$

No job arrives after the final time interval (t_n, t), and thus the number in the system at time t is

$$N = (N_n - D_{n+1})^+, \quad \text{where } 0 \leq N \leq n_0 + n.$$

Note that N depends on the service rate μ, but not on parameters of the arrival process because we know the actual arrival pattern during (t_0, t). That is, we assume exponential service times, but the arrival process can be arbitrary, i.e., the so-called G/M/1 model. Putting these together yields Algorithm 1 that computes the current state distribution $\boldsymbol{\pi} = (\pi_0, \ldots, \pi_{n_0+n})$ of the random variable N, where $\pi_i = \mathbb{P}\{N = i\}$.

procedure FINDN($n_0, t_0, \ldots, t_n, t$)
  $\boldsymbol{\pi} = (0, \ldots, 0, 1)$ ▷ $\pi_{n_0} = 1$
  for $i = 1, \ldots, n$ **do**
    $\boldsymbol{\pi} \leftarrow$ UPDATE($\boldsymbol{\pi}, t_i - t_{i-1}, 1$)
  end for
  $\boldsymbol{\pi} \leftarrow$ UPDATE($\boldsymbol{\pi}, t - t_n, 0$)
  return $\boldsymbol{\pi}$
end procedure

Algorithm 1: Computation of the state distribution based on the routing history and the state of the server at the start.

procedure UPDATE($\boldsymbol{\pi}, \Delta, a$)
  $k \leftarrow |\boldsymbol{\pi}|$ ▷ initially at most $k-1$ jobs
  $\boldsymbol{\pi}^* \leftarrow (0, \ldots, 0)$ ▷ $|\boldsymbol{\pi}^*| = k - 1 + a$
  for $i = 0, \ldots, k-1$ **do** ▷ condition on i jobs initially
    $s \leftarrow 1$ ▷ find $s = \mathrm{P}\{\text{empty system}\}$
    for $j = 0, \ldots, i-1$ **do** ▷ condition on j jobs departing
      $p \leftarrow (\mu\Delta)^j / j! \cdot e^{-\mu\Delta}$ ▷ pr. that j jobs completed
      $\pi^*_{i-j+a} \leftarrow \pi^*_{i-j+a} + p\pi_i$
      $s \leftarrow s - p$
    end for
    $\pi^*_a \leftarrow \pi^*_a + s\pi_i$ ▷ a jobs to an empty system
  end for
  return $\boldsymbol{\pi}^*$
end procedure

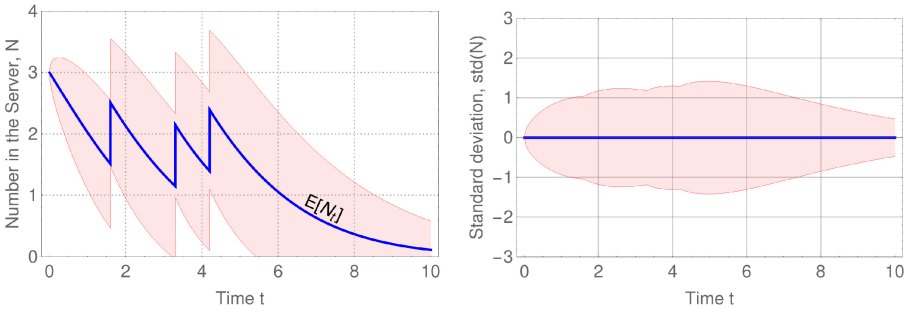
Update of the state distribution after time Δ.

Example 1. Let us consider the arrival pattern depicted in Fig. 1. At time $t_0 = 0$, the system has $n_0 = 3$ jobs (latest status update received). New jobs arrive at time instants $\{1.6, 3.3, 4.2\}$, and the service rate is $\mu = 1$. Figure 2 illustrates how our belief on the number N_t behaves as a function of time t. After the last arrival, at time $t = 4.2$, both the mean $\mathbb{E}[N_t]$ and the standard deviation σ_t gradually decrease to zero. Eventually we can be fairly sure that the system is empty even though no status report has been received since time $t_0 = 0$. A naïve assumption that the service time of all jobs is exactly the mean $1/\mu = 1$ would imply that the server is idle already at time $t = 6$. This is far from the reality, as we can see from the figure!

3 Dispatching with Known Past State

Suppose next that a dispatcher is routing jobs to k servers. The dispatcher keeps record of jobs it has routed to each server (say a job id and time stamp) that allows it to deduce which jobs have not been acknowledged yet by each server.

If we assume (i) first-come-first-served (FCFS) scheduling, and (ii) that there is no reordering (jobs arrive "immediately" to the server), a message about completion of job i acknowledges also all earlier jobs. Consequently, the dispatcher



(a) Mean number, $\mathbb{E}[N_t] \pm \sigma_t$ (b) Uncertainty, $\pm\sigma_t$

Fig. 2. Evolution of the mean number in the system with confidence intervals.

knows the exact number at the server at time t_0 when the acknowledgement was generated. We note that this resembles how (basic) TCP works: cumulative acknowledgments inform the sender that all octets up to a given point have reached the destination.

In general, we can assume any work-conserving scheduling discipline because (i) each departing job is (eventually) acknowledged, (ii) our performance metric is the mean response time (cf. Little's formula), (iii) service is exponential. Hence, our results hold also for processor sharing (PS) and last-come-first-served (LCFS).

In summary, our model for the dispatching system is as follows:

1. Jobs arrive to the dispatcher at inter-arrival times[1] A_i.
2. Job sizes are independent and exponentially distributed.
3. The server pool comprises k servers with service rates $\mu_1, \ldots, \mu_k$. Thus the service time at server j is $X_j \sim \text{Exp}(\mu_j)$.
4. The dispatcher routes jobs immediately upon arrival to the servers. The routing decision is irrevocable.
5. The dispatcher is aware of (e.g., it has learned) the server-specific service rates μ_j. Moreover, it is aware of the state of each server at some time in the past, $(n_0^{(j)}, t_0^{(j)})$, where $(n_0^{(j)}, t_0^{(j)})$ gets updated by acknowledgements as the process continues.

3.1 Dispatching Policies

Let us next introduce the dispatching policies considered in this paper.

P1 **Round-Robin (RR)** assigns jobs sequentially to all servers, $1, 2, \ldots, k, 1, 2, \ldots$. We note that RR is the optimal policy with respect to the mean

[1] The results for the distribution of N, and therefore also the corresponding policies, hold for any arrival pattern, even non-renewal, though in our numerical examples we consider only Poisson arrival processes.

response time in this setting with identical servers and no acknowledgements are available. Moreover, given RR ignores any acknowledgements, it is also agnostic to any delays in them.

P2 **Join-the-Shortest-Queue (JSQ)** chooses the queue with the least number of jobs,

$$\alpha_{\text{JSQ}} := \underset{j}{\text{argmin}}\, N_j.$$

Ties are resolved randomly. Note that this policy requires perfect information, i.e., that the N_j are known.

P3 **Naïve JSQ$_0$:** We can adapt JSQ to our setting by falsely assuming that the available information describes the current state accurately, i.e., that $N_j \approx n_0^{(j)} + n^{(j)}$, and

$$\alpha_{\text{JSQ}_0} := \underset{j}{\text{argmin}}\, n_0^{(j)} + n^{(j)}.$$

In other words, we apply JSQ without being aware of the delays in acknowledgements. Note that $n_0^{(j)} + n^{(j)}$ corresponds to the maximum number of jobs server j may have at time t. This policy will be equivalent to RR if we start empty and servers are homogeneous.

P4 **Time-aware JSQ$_\mathbf{e}$:** Given we can compute the distribution of N_j for all j, JSQ can be generalized to the case of delayed information by choosing the queue which is expected to have the least number of jobs by defining[2]

$$\alpha_{\text{JSQ}_\text{e}} := \underset{j}{\text{argmin}}\, \mathbb{E}[N_j].$$

3.2 Numerical Example 1

The first example system consists of 4 identical servers with service rates $\mu_i = 1$ for all i. Jobs arrive according to a Poisson process with rate $\lambda = 3$ to a dispatcher. Initially, at time $t = 0$, the servers have $\mathbf{n_0} = (1, 2, 3, 4)$ jobs, which is the available information to dispatching policies JSQ$_0$ and JSQ$_e$ throughout the time horizon (so there is no feedback from the servers). We assume RR will assign the next job to server 1. (This would be consistent with partial utilization of the available information about the initial state.) In contrast, RND is static, whereas JSQ utilizes the exact state information for every action.

The numerical results with 10,000 simulation runs are depicted in Fig. 3. On the x-axis is the time t, and the y-axis corresponds to the average costs incurred during $(0, t)$,

[2] In principle, it is possible that no job has departed from a busy server since the start. That is, the number of possible states increases without bound as the process continues, and evaluating the distribution and its mean eventually becomes cumbersome. As a workaround, our implementation of JSQ$_e$ truncates the state-space to $k_{\max} = 20$ jobs (per server), and updates the state probabilities accordingly whenever a new job arrives. Given the load is reasonable, the effect of this modification will be negligible.

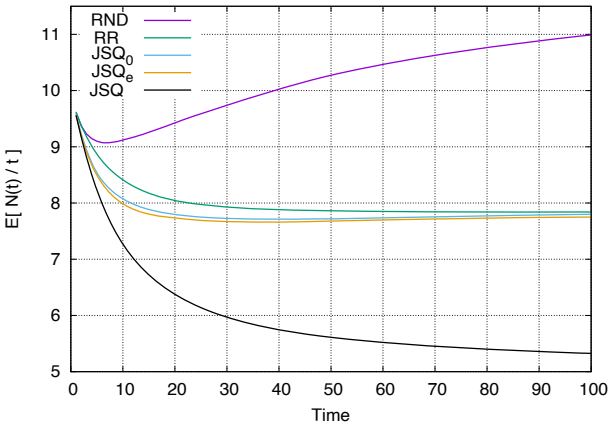


Fig. 3. The evolution of the mean cost from the known initial state $\mathbf{n_0} = (1, 2, 3, 4)$ as a function of time with four identical servers.

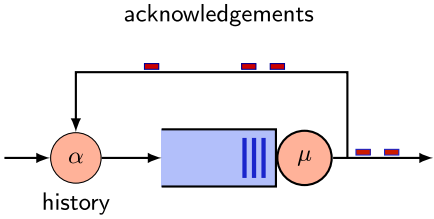


Fig. 4. Propagation delay influences the optimal dispatching.

$$C(t) := \frac{1}{t} \int_0^t \mathbb{E}[N_{\text{sys}}(h)]\, dh,$$

where $N_{\text{sys}}(h)$ denotes the total number of jobs in the whole system at time h. Obviously $C(t) \to \mathbb{E}[N_{\text{sys}}]$ as $t \to \infty$, and we can observe that the chosen dispatching policies fall into three groups: RND has the worst performance, {RR, JSQ_0, JSQ_e}, all reduce to RR as the time increases and the pure JSQ is the best (thanks to the exact state information). A closer look at the middle group shows that RR is initially worse than JSQ_0 and JSQ_e, of which the latter is marginally better. That is, JSQ_e utilizes the available information best.

3.3 Numerical Example 2

Let us now consider a scenario where the dispatcher receives acknowledgements from job completions after a fixed delay of d (see Fig. 4), and where t_0 is set to be the time the last acknowledgement was received.

Figure 5 depicts the mean response time with RR, JSQ, JSQ_0 and JSQ_e, as a function of the ACK delay d that is varied from zero to 4. JSQ_0 behaves as if d were zero, i.e., the decision is based on estimating the number at the server

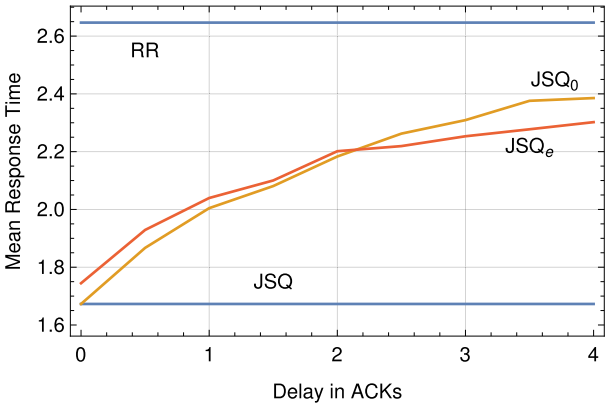
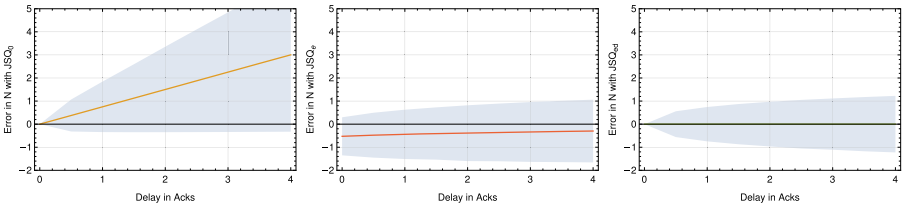


Fig. 5. Simulation results with 4 identical servers as a function of ACK delay d.



(a) JSQ_0 (b) JSQ_e (c) JSQ_ed

Fig. 6. Mean and standard deviation of the bias in the server state estimates, Q, with JSQ_0, JSQ_e and JSQ_{ed} as a function of ACK delay d.

simply by $\hat{N} = n_0 + n$ (which is a strict upper bound). On the other hand, JSQ_e is based on the mean of the distribution computed using Algorithm 1 with t_0, the time of the most recent acknowledgement. Initially, when $d = 0$, JSQ_0 reduces to JSQ and it is thus optimal in our setting. In contrast, the performance with JSQ_e is significantly worse. However, when d increases JSQ_e eventually becomes better than the elementary JSQ_0, and $d = \infty$ corresponds to example 1.

While RR, JSQ_0 and JSQ_e are all realistic choices in the given scenario, the pure JSQ cannot be implemented because it requires the knowledge about the exact number at each server. However, the gap between it and the realistic policies corresponds to the *price of information delay* quantifying the increase in the mean response time due to the delay d. As RR is agnostic and does not even try to utilize the incomplete information about the servers' states, it serves as an upper bound for the price of information delay in our setting.

Next we take a closer look at the quantities JSQ_0 and JSQ_e policies are based on, i.e., the expected number in each server upon dispatching, denoted by $\hat{N}_j$. We can omit the subscript j as this quantity is statistically the same for each server. Figure 6(a) and (b) depict the difference between the state estimates of

JSQ_0 and JSQ_e, and the actual number in the server upon dispatching,

$$Q := \hat{N} - N,$$

We observe that JSQ_0 overestimates the situation, which is explained by the fact that it is a strict upper bound for N because it includes also all the unacknowledged completed jobs into the estimate. Interestingly, JSQ_e underestimates the situation, and thus both get it quite wrong!

4 Delayed Completion Acknowledgement

In the last example, we made an interesting observation. Even though JSQ_e was supposed to have a better understanding about the current state of each server, this was not necessarily the case assuming delayed acknowledgements of service completions. Moreover, its performance can be worse than that of JSQ_0, which is easier to implement. This is somewhat unexpected and calls for a closer look.

Note that our implementation of JSQ_e is based on the assumption that the absence of more recent ACKs conveys no information. Similarly, JSQ_0 assumes that all unacknowledged jobs are still being processed. In our example scenario with a fixed delay on acknowledgements, both assumptions are wrong! We do know that job completions that we are not yet aware of may occur only during the time interval $I_d = (t - d, t)$. Thus instead of considering time intervals $I_i = (t_i, t_{i-1})$, we need to consider their intersection with I_d, i.e. the effective time interval for unknown departures is $I_i \cap I_d$.

That is, when the constant delay parameter d is known (or its distribution), we can utilize it to obtain a better understanding of N in each server. Otherwise the procedure works similarly as before.

Let e_i denote the length of the subset of time interval (t_{i-1}, t_i) during which jobs may have departed without our knowledge, $e_i := |I_i \cap I_d|$. We have three cases for $I_i \cap I_d$,

$$\begin{cases} (t_{i-1},\, t_i) & \text{when } t - d \le t_{i-1}, \\ (t - d,\, t_i) & \text{when } t_{i-1} < t - d \le t_i, \\ \emptyset, & \text{when } t_i < t - d. \end{cases} \tag{1}$$

Now it is easy to deduce that

$$e_i = \min\{t_i - t_{i-1},\ \max\{0,\ t_i - t + d\}\}.$$

The maximum number of departures, denoted by D_i, during time interval (t_{i-1}, t_i) has a Poisson distribution with parameter μe_i, and the corresponding update rule for determining N_i is

$$N_i = (N_{i-1} - D_i)^+ + a_i,$$

where $N_0 = n_0$, and $a_i = 1$ if a job arrives at time t_i and otherwise $a_i = 0$.

Including the knowledge about the fixed delay on ACKs thus leads to a minor modification to Algorithm 1. In `FindN`, the calls to `Update` must be modified so that the second parameter is e_i.

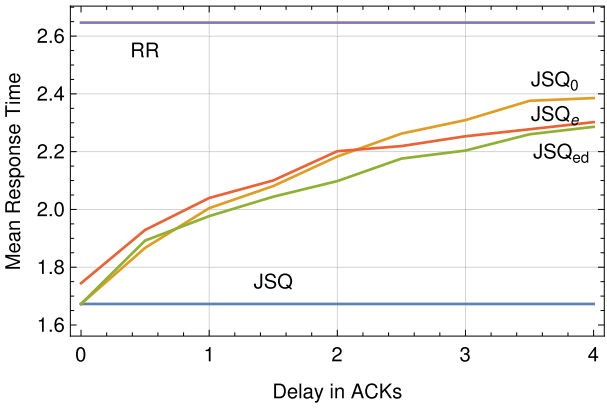


Fig. 7. Simulation results with 4 identical servers as a function of ACK delay d.

P5 **Delay-aware JSQ$_{\mathbf{ed}}$:** Taking into account the known fixed delay d when determining the server-specific state distribution, and then computing the expected number in servers leads to delay-aware JSQ.

4.1 Numerical Example 3

Let us next consider the same setting as in the last example, i.e., four identical servers and the constant delay d of the ACKs is varied from zero to 4. Figure 7 depicts the numerical results, including for the new policy JSQ$_{ed}$. We can observe that JSQ$_{ed}$ yields the shortest mean response time among all policies that can be realized (i.e., excluding the basic JSQ that violates our assumptions on the available information). It thus gives the best estimate for to the price of information delay in our scenario.

Figure 6(c) depicts the error in JSQ$_{ed}$'s estimate on the server state. We can see that it is based on the unbiased estimate (the mean is correct), however, the standard deviation is non-negligible.

4.2 Numerical Example 4: Dependency on Load

Next we study how the offered load ρ affects the performance. We fix the delay in acknowledgements to $d = 2$ and vary the arrival rate λ. Otherwise the example system is kept the same, i.e., we have 4 identical servers with $\mu_i = 1$.

Numerical results are depicted in Fig. 8. The x-axis corresponds to the offered load, $\rho = \lambda / \sum_j \mu_j$, and the y-axis to the mean response time scaled by $(1 - \rho)$. Hence, with a single fast server with $\mu' = \sum_j \mu_j$, one would obtain the mean response time of $\mathbb{E}[T'] = 0.25$. This is depicted with the dashed constant line in the figure.

It is interesting to note that RR can be clearly better than the (naïve) JSQ$_0$. In our example case, this happens when the offered load is moderate.

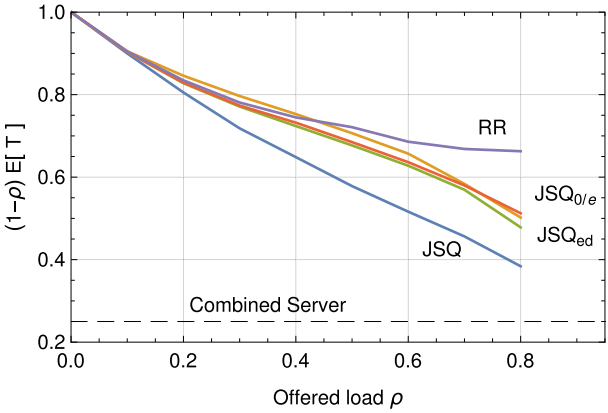


Fig. 8. Mean response time as a function of the offered load ρ with four identical servers and ACK delay of $d = 2$.

5 Heterogeneous Servers

Often servers are not identical. For example, the server pool may comprise two types of hardware due to an upgrade. In this section, we generalize the heuristic policies to the case of heterogenenous service rates.

We adapt the previously discussed dispatching policies accordingly. Instead of RR, we can use the so-called generalized Round-Robin (GRR) [1]. With GRR, faster servers appear more frequently in the number sequence defining the assignment pattern.

Similarly, instead of JSQ we consider the *shortest-expected-delay* (SED) [14],

$$\alpha_{\text{SED}} := \operatorname*{argmin}_j \frac{n_j + 1}{\mu_j}.$$

where μ_j is the service rate of server j, and n_j is the current number in server j. Given the exact state information is not available, we resort to estimating the mean $\mathbb{E}[N_j]$ the same way as before. Similarly as with JSQ, the subscript indicates how $\mathbb{E}[N_j]$ is estimated (based on delayed acknowledgements).

5.1 Numerical Example 5

As an example scenario, suppose we have three servers with service rates $\boldsymbol{\mu} = \{2, 1, 1\}$, so that server 1 is twice as fast as the other two servers. As in our earlier numerical examples, jobs arrive according to Poisson process. The propagation delay of the acknowledgements is constant, either small $d = 0.5$, or large $d = 3$.

As mentioned, the standard round robin sequence would be ill-fitted given one server is significantly faster than other, and we will resort to the generalized Round-Robin (GRR). In our example case the optimal sequence is trivially $1, 2, 1, 3, 1, 2, \ldots$.

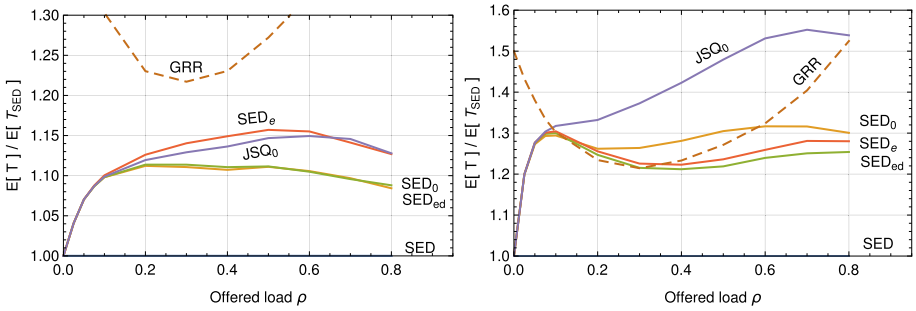


Fig. 9. Mean response time as a function of the offered load ρ with three heterogeneous servers and ACK delay of $d = 0.5$ and $d = 3$.

Numerical results are depicted in Fig. 9. The y-axis corresponds to the mean response time relative to SED with perfect state information. We also included the (naïve) JSQ, resolving ties in favor of the fast server, for comparison.

When the propagation delay of ACKs is small (left figure), we can see that both SED_0 and SED_{ed} work well, whereas SED_e and JSQ_0 have similar performance. GRR utilizes the slower servers unnecessarily, especially at low loads, and it is no match to these dynamic policies.

However, when the propagation delay increases to $d = 3$ things change dramatically. First, SED_{ed} yields the shortest mean response time, followed by SED_e, as expected. These are the policies that take into account the delay in acknowledgements, and its role is now sufficiently large that even SED_e does a good job. JSQ_0 shows significantly worse performance than SED policies. At the same time, GRR starts to move up in the ranks. This is expected as GRR is insensitive to delay in acknowledgements. In fact, it is also insensitive to delay from dispatcher to server, should there be such.

To summarize, our numerical experiments suggest RR and its generalized version (GRR) are good robust options when the state information available to the dispatcher is vague and/or noisy, especially when load is relatively high. Moreover, false assumptions on the process describing the delivery of acknowledgements can severely degrade the performance of the system when a JSQ- or SED-type of policy is applied. With accurate modelling under the right assumptions, JSQ- and SED-policies can work well even under uncertainties about the server states. However, when delays are small, the standard JSQ and SED remain near-optimal and should be favored for their simplicity.

6 Conclusions

In this paper, we considered elementary yet tractable models for dispatching systems subject to varying information and (random) delays in servers' status updates. Such settings arise, e.g., in large or real-time systems, where it is not practical to query the state of each server for every job.

The main contributions of this paper are:

1. We give computationally straightforward algorithms to determine distributions for the server states based on incomplete information. The arrival process can be arbitrary, but exponential service times are assumed, which is a reasonable assumption, e.g., with a large number of servers.
2. The (naïve) JSQ, referred to as JSQ_0, turned out to be worse than RR in some cases. This is due to the fact that its estimate on server states has bias. In other words, before applying JSQ, one should ensure that the state information is timely and accurate [11].
3. Moreover, we observed that it is also important to take into account the delay pattern appropriately. Our first trial, JSQ_e turned out to have defiencies especially when completed jobs are acknowledged and ACK delays are short.
4. Our delay-aware policy, JSQ_{ed}, takes the delay structure appropriately into account. In the example cases, its performance was better than with any other comparable policy based on the same amount of information.
5. With heterogeneous servers, JSQ is often replaced by SED, which takes into account the service rates. Similarly as with JSQ, SED can also be adapted to the setting of uncertain state information. The numerical results suggest similar observations; with a short delay the standard SED works well, but as the delay increases it must be taken into account. SED_{ed} turned out to be superior across different example scenarios.
6. Generalized round robin (GRR) is well-suited for heterogenerous servers. It is a robust policy that works well when state information is vague or contains unknown errors (e.g. wrong assumptions about the ACK propagation delay processs). However, with sufficiently accurate state information even the basic JSQ_0 was better than GRR.

The dynamic decision making scenario studied in this paper can be approached in the framework of partially observable MDPs. Similarly, one can develop "black-box" machine learning models that can be expected to perform well after a sufficiently long training period. However, these approaches are less insightful on how the system works, and the learning phase in real operation can be disruptive when the operational parameters change. In contrast, our approach can be *immediately* adapted to a new scenario, e.g., if the arrival rate Λ or the size of the server pool changes. This also allows dynamic dimensioning of the server pool without the long experiments that machine learning approaches would require.

References

1. Altman, E., Gaujal, B., Hordijk, A.: Balanced sequences and optimal routing. J. ACM **47**(4), 752–775 (2000)
2. Altman, E., Kofman, D., Yechiali, U.: Discrete time queues with delayed information. Queueing Syst. **19**, 361–376 (1995)
3. Artiges, D.: Optimal routing into two heterogeneous information. In: Proceedings of the 32nd Conference on Decision and Control, San Antonio, Texas (1993)

4. Buzen, J.P., Chen, P.P.: Optimal load balancing in memory hierarchies. In: Proceedings of the 6th IFIP Congress, pp. 271–275, Stockholm, Sweden, August 1974
5. Ephremides, A., Varaiya, P., Walrand, J.: A simple dynamic routing problem. IEEE Trans. Automatic Control **25**(4), 690–693 (1980)
6. Haight, F.A.: Two queues in parallel. Biometrika **45**(3–4), 401–410 (1958)
7. Lipschutz, D.: Open problem-load balancing using delayed information. Stochas. Syst. **9**(3), 305–306 (2019)
8. Litvak, N., Yechiali, U.: Routing in queues with delayed information. Queueing Syst. **43**, 147–165 (2003)
9. Liu, Z., Righter, R.: Optimal load balancing on distributed homogeneous unreliable processors. Oper. Res. **46**(4), 563–573 (1998)
10. Liu, Z., Towsley, D.: Optimality of the round-robin routing policy. J. Appl. Probab. **31**(2), 466–475 (1994)
11. Mitzenmacher, M.: How useful is old information? IEEE Trans. Parallel Distrib. Syst. **11**(1), 6–20 (2000)
12. Novitzky, S., Pender, J., Rand, R.H., Wesson, E.: Nonlinear dynamics in queueing theory: determining the size of oscillations in queues with delay. SIAM J. Appl. Dyn. Syst. **18**(1), 279–311 (2019)
13. Pender, J., Rand, R., Wesson, E.: A stochastic analysis of queues with customer choice and delayed information. Math. Oper. Res. **45**(3), 1104–1126 (2020)
14. Selen, J., Adan, I., Kapodistria, S., van Leeuwaarden, J.: Steady-state analysis of shortest expected delay routing. Queueing Syst. **84**(3), 309–354 (2016)
15. Whitt, W.: On the many-server fluid limit for a service system with routing based on delayed information. Oper. Res. Lett. **49**, 316–319 (2021)
16. Winston, W.: Optimality of the shortest line discipline. J. Appl. Probab. **14**, 181–189 (1977)