To Split or not to Split: Selecting the Right Server with Batch Arrivals

Esa Hyytiä[1,*], Samuli Aalto

Department of Communications and Networking
Aalto University, Finland

Abstract

We consider a dispatching system, where jobs, arriving in batches, are assigned to single-server FCFS queues. Batches can be *split* to different queues on per job basis. However, the holding costs are batch-specific and incurred until the last member of the batch completes the service. By using the first policy improvement step of the MDP framework, we are able to derive robust dispatching policies, which split arriving batches only when deemed advantageous. The approach is also demonstrated in numerical examples.

Keywords: job dispatching; task assignment; batch arrivals; parallel processing

1. Introduction

The question of splitting a task or a job arises in many interesting situations. In the context of transportation of goods, one can ask when splitting some deliveries is advantageous [1]. In distributed and parallel computing the key idea is to split the tasks and process them concurrently whenever feasible. The final outcome becomes available only upon the completion of the last sub-task. Similarly, in a custom and immigration inspection, e.g., at an international airport, group arrivals can either be split to several counters or assigned to a single one. A group will wait until its last member is processed, i.e., the natural cost structure is a batch specific.

The system considered in this paper is illustrated in Fig. 1. Customers or tasks arrive according to a Poisson process in batches and are assigned to single-server queues immediately upon the arrival. It is possible to *split* a batch to several servers, or keep it together. Scheduling in each queue is the first-come-first-served (FCFS). We are interested in minimizing the sojourn time of a whole batch, possibly weighted with some batch-specific holding costs.

The described system is closely related to the basic dispatching or routing problem studied extensively in the literature [2, 3, 4, 5, 6]. In the basic problem, single customers arrive to a dispatcher and the typical task is to minimize the mean sojourn time (i.e., the mean response time). In particular, if one restricts oneself to batch-specific policies that assign the whole batch to the same server, each batch constitutes a macro-job and the system with batch arrivals reduces to the basic dispatching problem. However, in this paper, we do not make such a restriction but allow the splitting of the batches.

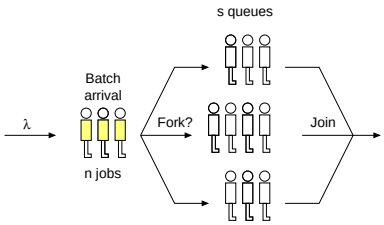


Figure 1: Customers arrive in batches and the costs are accrued until the last member of the batch has completed the service.

In the so-called fork-join queue, each incoming task is unconditionally split to s available servers, each having its own FCFS queue. This model was first considered by Flatto and Hahn in [7]. Exact and approximate results for the mean delay are available only for $s = 2$ servers [8]. We note that a fork-join queue is equivalent to our model when the batch size is constant and equal to the number of servers, and the dispatching policy assigns each sub-task to a different server.

2. Model

We assume that jobs arrive in batches according to a *Poisson process* with rate λ. Each batch arrival i comprises N_i jobs, where the *batch-sizes* N_i are assumed to be i.i.d. random variables $N_i \sim N$. Similarly, we let $\mathbf{X}_i = (X_{i,1}, \ldots, X_{i,N_i})$ denote the service requirements of the arriving batch i. Two batches are assumed to be i.i.d. so that the total offered load is

$$\rho_{\text{tot}} = \lambda \cdot \mathrm{E}[X_1 + \ldots + X_N].$$

If the service requirements are independent of the batch size N, but not necessarily pairwise independent within a batch, then

$$\rho_{\text{tot}} = \lambda \cdot \mathrm{E}[N] \cdot \mathrm{E}[X],$$

where X denotes the service requirement of a random job. Later, in the numerical examples, we assume the latter while the methodology is directly applicable also in the general case.

*Corresponding author
Email addresses: esa.hyytia@aalto.fi (Esa Hyytiä), samuli.aalto@aalto.fi (Samuli Aalto)
[1]Postal address: Aalto University, School of Electrical Engineering, Department of Communications and Networking, PO Box 13000, FI-00076 AALTO.

Jobs are served by a pool of s parallel single-server FCFS queues, with queue-specific service rates denoted by $\nu_1, \ldots, \nu_s$. We assume,

$$\rho_{\text{tot}} < \nu_1 + \ldots + \nu_s,$$

which is a necessary condition for a stable system. (additionally, the dispatching policy must balance the load in such a manner that no single queue is overloaded). We let random variables U_i denote the current waiting time (backlog in time units) in Queue i, $i = 1, \ldots, s$. With full state information, $U_i = u_i$, i.e., the exact (remaining) service times are available.

The *cost structure* is batch specific, i.e., batch i accrues costs at rate b_i until all jobs of the batch have been served. We note that the processing order within the same batch is thus irrelevant as the costs are incurred until the last member of the batch has completed the service. Therefore, the dispatching decision can be characterized by an n-tuple $\mathbf{d} = (d_1, \ldots, d_n)$ with $d_j \in \{1, \ldots, s\}$ defining the queue for task j with $n = N_i$.

With unit holding cost, $b_i = 1$ for all batches i, the optimal policy minimizes the mean sojourn time of a batch, which according to Little's result is equivalent to minimizing the mean number of batches in the system (not jobs). For minimizing the mean sojourn time of jobs in the system, the holding cost is equal to the batch size, i.e., $b_i = n$, where we assume that jobs also depart the system in batches (the join operation). Additionally, one can envision situations where some batches are more important, which translates to a higher holding cost (rate) in our model.

3. Greedy basic policy

A greedy dispatching policy (i.e., a batch of selfish customers) minimizes the expected sojourn time of their own batch. With a given decision $\mathbf{d}$, let $I(\mathbf{d})$ denote the set of queues receiving at least one new customer,

$$I(\mathbf{d}) \triangleq (i \,:\, d_j = i \text{ for some } j),$$

and $A_i = A_i(\mathbf{d})$ the amount of work Queue i receives,

$$A_i \triangleq \frac{1}{\nu_i} \sum_{j=1}^{n} 1(d_j = i) \cdot X_j.$$

Then the sojourn time of the batch is given by

$$S(\mathbf{X}; \mathbf{d}) \triangleq \max_{i \in I(\mathbf{d})} (U_i + A_i).$$

The greedy action is the one minimizing the expectation,

$$\operatorname*{argmin}_{\mathbf{d}} \mathrm{E}\left[\max_{i \in I(\mathbf{d})} (U_i + A_i)\right],$$

Thus, in general one finds the optimal greedy decision by enumerating the n^s possible assignments and choosing the one with the smallest expected sojourn time (which is independent of the batch specific holding cost). Unfortunately, the general expression is not amenable for straightforward approaches due to the expectation of the maximum. With the full state information, the job sizes and waiting times become deterministic and the task reduces to the basic enumeration. This facilitates also efficient pruning of the search space.

4. Towards the Optimal Policy

In this section, we assume a central dispatcher which seeks to minimize the long-term cumulative costs. In general, we assume a size-aware system, where (remaining) service requirements are available. For each batch, the dispatcher has to weigh between the "immediate costs" related to the sojourn time of the given batch (known exactly) and the "future costs" related to the later arriving batches (stochastic component).

Consider first dispatching policies where the same decision applies to the whole batch, i.e., each customer of the batch is assigned with the same queue. All these policies effectively combine the jobs of the batch to a single macro-job, and the system reduces to an ordinary dispatching system. We refer to such policies as *the batch-specific policies*.

Another important class of policies are the so-called *state-independent policies*, where the dispatching decision is independent of the past decisions and the queue states. It can, however, depend on the new batch including its jobs, their sizes and the holding cost b. The key property of state-independent policies is that the arrival process of macro-jobs to each queue is a Poisson process.

Several well-studied examples of state-independent batch-specific policies exists. First, one can carry out a Bernoulli-split, i.e., independently of the other arrivals, earlier decisions and queue states, the whole batch is assigned to Queue i with probability of p_i. Alternatively, the dispatching probabilities can depend on customer or batch classes or such useful information as the (expected) service requirements.

4.1. Policy Improvement

With state-independent batch-specific policies, the arriving process to each Queue i is a Poisson process with some rate λ_i, and the mean sojourn times, or the mean cost accruing rates with arbitrary holding cost rates, follow from the Pollaczek-Khinchine mean value formula for the whole batch. For such policies, similarly as in our past work [9, 10], it is possible to carry out the first policy iteration (FPI) step, yielding a new improved state-dependent policy, which also splits batches to several servers when deemed advantageous. To this end, one needs the so-called value function for the whole system. However, when the basic policy is state independent and batch specific, the system separates and it is sufficient to know the value functions of the corresponding individual M/G/1-FCFS queues, and consider the macro-jobs comprising of the jobs of a batch assigned to the same server.

Let $\mathbf{z} = ((\Delta_1, b_1), \ldots, (\Delta_m, b_m))$ denote a state of an ordinary M/G/1-FCFS queue with m jobs, where Δ_k is the (remaining) service time of job k, and b_k is its holding cost rate (per unit time). Job m is currently receiving service and job 1 is the most recent arrival. The value function of an M/G/1-FCFS with respect to arbitrary job-specific holding costs is [9, 10],

$$v_{\mathbf{z}} - v_0 = \sum_{j=1}^{m} \left(b_j \sum_{k=1}^{j} \Delta_k \right) + \frac{\lambda \cdot \mathrm{E}[B]}{2\,(1-\rho)} u_{\mathbf{z}}^2, \tag{1}$$

where $u_{\mathbf{z}} = \Delta_1 + \ldots + \Delta_m$ denotes the backlog in state $\mathbf{z}$, $\mathrm{E}[B]$ the mean holding cost rate of later arriving jobs, and ρ the proportion of time that the server is busy (in the long run).

In our case, we need to adjust (1) slightly in order to take into account the batch-specific cost structure. We assume that the current state is arbitrary, i.e., batches may have been split among several servers, but all the future arrivals will be assigned according to a state-independent batch-specific policy. In particular, we can consider macro-jobs, i.e., the jobs of a batch assigned to the same queue, and let Δ_k denote the service time of macro-job k. For each batch, we set $b_k = 0$ for all macro-jobs k except for the one that is scheduled to depart last from the given batch (note that the departure order gets fixed upon dispatching). Then (1) holds for each queue.

Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_s)$ denote the state of the whole system comprising s parallel queues. Due to Poisson arrivals, the system separates and the value function is

$$\begin{aligned} v_{(\mathbf{z}_1,\ldots,\mathbf{z}_s)} - v_0 &= \sum_{i=1}^{s} v_{\mathbf{z}_i} - v_0 \\ &= \sum_{i=1}^{s} \left(\sum_{j=1}^{m_i} \left(b_{i,j} \sum_{k=1}^{j} \Delta_{i,k} \right) + \frac{\lambda_i \cdot \mathrm{E}[B_i]}{2\,(1-\rho_i)} u_i^2 \right). \end{aligned} \tag{2}$$

Let a_i denote the amount of work added to Queue i with a given decision $\mathbf{d}$, and b the holding cost of the new batch. Then define

$$i^* \triangleq \operatorname*{argmax}_{i \in I(\mathbf{d})} (a_i + u_i),$$

i.e., $i^* = i^*(\mathbf{d})$ denotes the queue from which the last job of the new batch will depart. Let $\mathbf{Z} \oplus \mathbf{d}$ denote the resulting state after dispatching the new batch according to decision $\mathbf{d}$. From (2), it follows that the *admittance cost*, $c(\mathbf{d}) = v_{\mathbf{Z}\oplus\mathbf{d}} - v_{\mathbf{Z}}$, is given by

$$c(\mathbf{d}) = \overbrace{b(a_{i^*} + u_{i^*})}^{=\text{immediate}} + \overbrace{\sum_{i \in I(\mathbf{d})} \frac{\lambda_i \mathrm{E}[B_i]}{2(1-\rho_i)} \left(2u_i a_i + a_i^2 \right)}^{=\text{future}}, \tag{3}$$

where $(\lambda_i, \mathrm{E}[B_i], \rho_i)$ denote the arrival rate, the average holding cost, and the offered load of Queue i, respectively, all according to the chosen state-independent batch-specific basic policy. The latter sum term, indicated with "future", corresponds to the expected increase in cumulative costs the later arriving batches experience with the given basic policy. According to FPI, one chooses $\mathbf{d}$ that minimizes $c(\mathbf{d})$,

$$\operatorname*{argmin}_{\mathbf{d}} c(\mathbf{d}).$$

Note that starting FPI with a batch-specific basic policy is not such a big restriction as it may first seem. Especially, if there is uncertainty about the backlogs, choosing more queues simply increases the chances that one of them turns out to progress slower than expected.

Here we assume a size-aware system with known (remaining) service times. In passing, we note that by conditioning the obtained admittance cost can be generalized to situations where backlogs and service requirements are random variables [11].

5. Numerical Examples

Next we illustrate the framework by numerical examples. For comparison, we consider the following heuristic policies:

1. Random (RND) chooses the queue uniformly in random (thus balancing the load).
2. Size-Interval-Task-Assignment (SITA) [12, 13, 14] uses $s + 1$ thresholds ξ_i, $0 = \xi_0 < \xi_1 < \ldots < \xi_s = \infty$, and chooses Queue i if the size x of a job/batch belongs to the ith interval, $\xi_{i-1} \le x < \xi_i$. The ξ_i can be optimized with aid of Pollaczek-Khinchine formula [5, 6]. For simplicity, we consider SITA-E, where the ξ_i are chosen in such a way that each queue receives an equal load [13].
3. Join-the-shortest-queue (JSQ) chooses the queue with the smallest number of jobs, $\operatorname{argmin}_i m_i$.
4. Least-work-left (LWL) chooses the queue with the smallest backlog, $\operatorname{argmin}_i u_i$.

With JSQ and LWL, the possible ties are broken in favor of the queue with a smaller id. For all above, we apply both (i) batch-wise and (ii) job-wise assignment, yielding 4 batch- and 4 job-specific policies.

The RND and SITA policies are state-independent, i.e., their decisions are independent of the states of the queues. Another state-independent policy is the Number-In-Batch-Assignment (NIBA) given in Algorithm 1. As the name suggests, it utilizes the batch-size information (i.e., the number of jobs in the new batch), and assigns the batches with the *same number of jobs* to the same servers while balancing the load at the same time. The rationale is the same as with SITA, i.e., it aims to seggregate short batches from the long ones thus decreasing the chances of a short batch waiting behind a long one. Algorithm 1 assumes s identical servers, but it is straightforward to generalize NIBA also for a heterogeneous case.

We consider an elementary system with four identical servers ($s = 4$) each processing jobs with unit service rate, $\nu_i = 1$, for $i = 1, \ldots, s$. Batches arrive according to a Poisson process with rate λ. The batch size N is either 1, 2 or 3 with probabilities of $1/4$, $1/2$ and $1/4$, respectively. The service requirements X are also independent and identically distributed with mean 1:

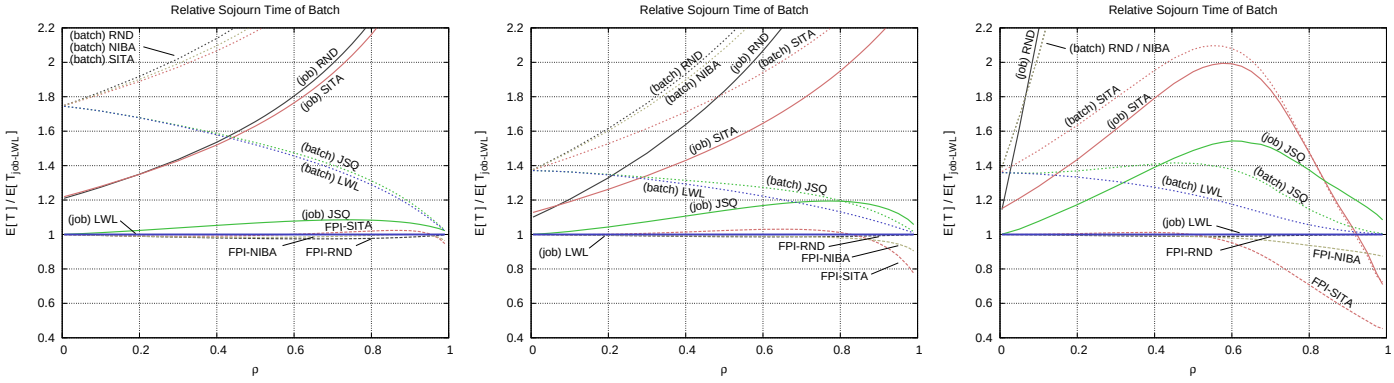
1. U(0.5, 1.5): uniform distribution on interval (0.5, 1.5).
2. Exp(1): exponential distribution with mean 1.
3. Bounded Pareto distribution with parameters $(a, b, \alpha) = (0.33959, 1000, 1.5)$, where (a, b) is the interval and α is the shape parameter. Thus, CDF for $a \le x \le b$ is

$$\mathrm{P}\{X < x\} = \frac{1 - (a/x)^\alpha}{1 - (a/b)^\alpha}.$$

The offered load per server is $\rho = \rho_{\text{tot}}/s = \lambda\,\mathrm{E}[N]\,\mathrm{E}[X]/s = \lambda/2$. A load-balancing routing matrix for NIBA is

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 \\ 0 & 0 & 1/3 & 2/3 \end{pmatrix},$$

i.e., batches with one job are assigned to Queue 1, batches with two jobs to Queues 1, 2 and 3, and batches with three jobs to Queues 3 and 4. As a result, all queues receive the same load, $\rho = \lambda/2$ (as $\mathrm{E}[X] = 1$).



$X \sim \mathrm{U}(0.5, 1.5)$ $X \sim \mathrm{Exp}(1)$ $X \sim$ Bounded Pareto

Figure 2: The mean relative sojourn time of a batch with different dispatching policies against the job-specific LWL.

As for the cost structure, we consider three scenarios: (i) minimization of the mean number of batches ($B = 1$), (ii) minimization of the mean number of jobs ($B = N$) assuming jobs also depart in batches (join operation), and (iii) minimization of the mean cost rate with two priority classes.

Algorithm 1 Number-In-Batch-Assignment (NIBA).

$p_{j,i} = 0 \; \forall \, i, j$ {Routing matrix: i=queue, j=batch size}
$i = 1$; $a = \mathrm{E}[N]/s$
$j = 1$; $b = j \cdot \mathrm{P}\{N = j\}$; $q = 1$
while $i \leq s$ **do**
  if $a \leq q \cdot b$ **then**
    $p_{j,i} = a/b$; $q = q - a/b$
    i++; $a = \mathrm{E}[N]/s$ {next queue}
  else
    $p_{j,i} = q$; $a = a - q \cdot b$;
    j++; $b = j \cdot \mathrm{P}\{N = j\}$; $q = 1$ {next batch size}
  end if
end while

5.1. Example: Sojourn time

Let us first consider the mean sojourn time (mean delay) of batches, i.e., $B = 1$ for all batches. As just explained, we consider nine heuristic policies: the job- and batch-specific RND, SITA, JSQ and LWL, and the batch-specific NIBA. The batch-specific RND, SITA and NIBA are also state-independent, and thus they can serve as basic policies for the FPI step. To this end, we need to determine the queue-specific batch arrival rates λ_i, the mean holding cost rates $\mathrm{E}[B_i]$ and the offered loads ρ_i for each basic policy, in accordance with (3). As here $B = 1$, we trivially have $\mathrm{E}[B_i] = 1$. Similarly, as we have chosen to consider only load-balancing basic policies, $\rho_i = \rho = \rho_{\mathrm{tot}}/s$, $\forall \, i$. The arrival rates λ_i, however, depend on the basic policy. For RND, we have $\lambda_i = \lambda/s$, and for NIBA, $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (9, 6, 5, 4) \cdot \lambda/24$. For SITA the queue-specific arrival rates λ_i depend on the job size distribution and are determined numerically at the same time as the SITA thresholds ξ_i. As the workload in a batch is a random sum, $X_1 + \ldots + X_N$, no elementary expression exists for the (load-balancing) thresholds with the chosen example job size distributions.

The numerical results are depicted in Fig. 2. Each graph shows the performance of different policies against the job-specific LWL as a function of the offered load ρ. First we observe that it is advantageous to process batches in parallel whenever the offered load is sufficiently low. However, the situation gets more interesting when the load increases.

The optimal decision naturally depends on the available information. JSQ is aware of the number of jobs in each server but not on their (remaining) service times. Consider, for example, the case with exponentially distributed job sizes (middle figure). Initially, when ρ is small, the job-specific JSQ is clearly a better option. A switch-over point is about at $\rho \approx 0.8$, after which one should no longer split a batch to several servers.

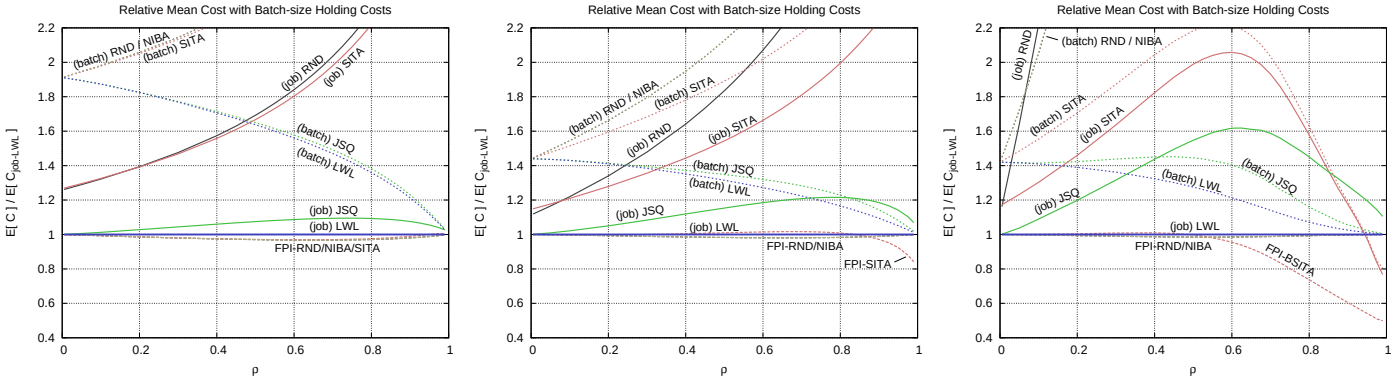
SITA policies are aware of the exact service time(s) of the new job/batch, but not the state of the queues. They are known to (typically) perform well when service times vary a lot, which can be observed also here. LWL and FPI are aware of the actual backlogs in each queue and achieve clearly the lowest mean sojourn times. In particular, with exponential and bounded Pareto distributed job sizes, FPI-SITA appears to be a good choice especially under a heavy load.

5.2. Example: Holding cost according to the size of the batch

Next, the holding cost is equal to the size of the batch, $B = N$. This type of a cost structure arises, e.g., in the immigration inspection when minimizing the individual sojourn time with group arrivals. It turns out that in this case FPI-NIBA and FPI-RND are equivalent due to the fact that (cf. (3)),

$$\frac{\lambda_i \, \mathrm{E}[B_i]}{2(1-\rho_i)} = \frac{\rho}{2(s-\rho)\,\mathrm{E}[X]},$$

for both. That is, from the view point of FPI, all queues receive equivalent streams of batches. With RND, the arrival processes to all queues are statistically identical, while with NIBA this obviously is not the case, but, e.g., the mean batch size varies. With SITA, the situation is again somewhat more complicated and both λ_i and $\mathrm{E}[B_i]$ are determined numerically as a function of the SITA thresholds ξ_j.



$X \sim \mathrm{U}(0.5, 1.5)$ $X \sim \mathrm{Exp}(1)$ $X \sim$ Bounded Pareto

Figure 3: The mean holding cost of a batch with different dispatching policies compared against the job-specific LWL, when holding cost is equal to the batch size.

Simulation results are similar as in the previous example and depicted in Fig. 3. For example, with exponential and bounded Pareto distributed job sizes, the batch-specific JSQ and LWL become better than the job-specific JSQ when the offered load exceeds a certain threshold. Similarly, we observe that FPI-SITA is again significantly better than the job-specific LWL when service times vary a lot. As $\rho \to 1$, FPI-SITA outperforms the job-specific LWL by a factor of 2 with the bounded Pareto distributed job sizes.

5.3. Example: High priority batches

Finally, we consider a case with two classes: a high priority Class 1 having a holding cost rate of $b = 4$, and a low priority Class 2 with $b = 1$. The batch sizes and service requirements are i.i.d. and obey the same distributions for both classes. Fraction of high priority Class 1 batches is 25% and the remaining 75% of the batches are Class 2. As the local scheduling discipline in each queue is FCFS, the only place where the high priority customers can be favored is at the dispatcher. All heuristic policies, whether batch-specific or not, are *blind* to the holding costs and behave the same way as in the previous examples.

Additionally, we define a specific state-independent *class-aware* policy, referred to as the HP25 policy, which seggregates the high priority requests from the other by assigning all high priority batches to Queue 1, and the rest in random to Queues 2-4. As a result, the arrival process to each queue is a Poisson process with rate $\lambda/4$ and load $\rho = \rho_{\mathrm{tot}}/4 = \lambda/2$. Therefore, the performance of HP25 and RND is actually equal. While dedicating more capacity to the high priority class would give us a better class-aware policy, HP25 is already sufficient for our purposes. With HP25, the mean holding cost for Queue 1 is trivially $\mathrm{E}[B_1] = 4$ and for the other queues $\mathrm{E}[B_i] = 1$. Therefore, when HP25 is used as the basic policy for FPI, yielding FPI-HP25, Queue 1 has a special role via the mean holding cost. We note that in addition to HP25, all FPI policies also take the holding cost structure into account.

As the other basic policies, i.e., the batch-specific RND, NIBA and SITA, are blind to holding costs, we have $\mathrm{E}[B_i] = \mathrm{E}[B]$ for them. Also the queue-specific arrival rates λ_i and offered loads ρ_i are determined for each basic policy similarly as earlier in order to enable the policy improvement step.

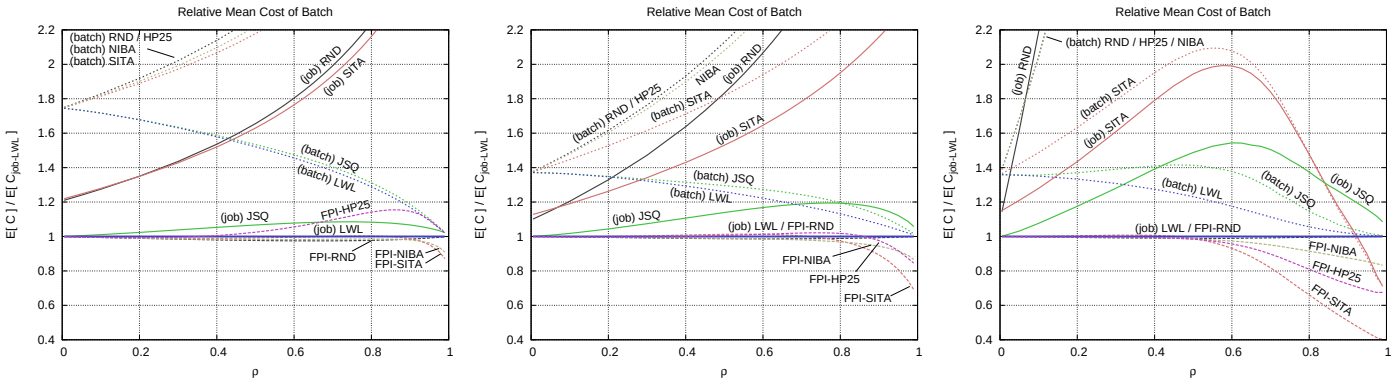
The simulation results are depicted in Fig. 4. Again, with the uniformly distributed job sizes, the FPI policies are only marginally better than the job-specific LWL. With exponentially distributed job sizes, FPI-NIBA, FPI-HP25 and FPI-SITA outperform LWL when $\rho \to 1$. As the job sizes vary more (bounded Pareto distribution), the FPI policies start to shine. In particular, FPI-SITA achieves over 50% reduction in the mean holding cost under a heavy load.

Fig. 5 illustrates the fraction of batches split to several servers with the exponentially distributed job sizes (batch-specific policies are omitted for the obvious reason). The behavior with other job size distributions was similar. When the offered load is very low, all batches with more than one task (i.e. 75% of the batches) are split and processed in parallel, as expected. As the offered load increases, all policies, and the FPI policies in particular, start to avoid splitting more and more. We believe that also the optimal policy behaves in a similar way, i.e., the fraction of batches split decreases as the offered load increases.

6. Conclusions

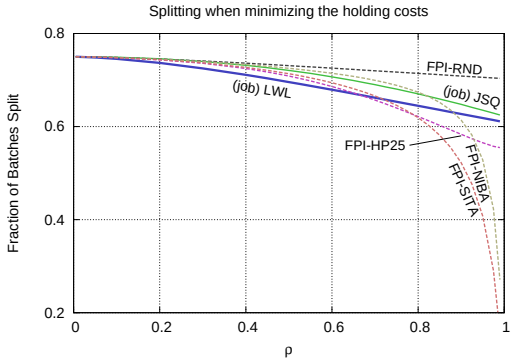
Traditional dispatching problem addresses the problem of server selection for arriving tasks. In this paper, we have addressed the joint-problem of splitting the tasks and choosing the server for each sub-task, i.e., the task assignment problem with batch arrivals. The possibility to split some tasks enables parallel processing. The server system is assumed to comprise s single-server FCFS queues. Each batch has some *batch-specific holding cost rate* and the task is to minimize the long-term mean cost rate. The model is related to the fork-join queues, where the splitting is not optional but the default action. This type of dynamic decision problems arise frequently, e.g., in the context of parallel and distributed computing [8].

We approached the dispatching problem in the MDP framework, which allows one to improve any state-independent batch-specific basic policy in a straightforward manner. The choice of a good basic policy (with respect to policy improvement) depends on the particular setting (cost structure, arrival



$X \sim \mathrm{U}(0.5, 1.5)$ $X \sim \mathrm{Exp}(1)$ $X \sim$ Bounded Pareto

Figure 4: The mean holding cost of a batch with different dispatching policies compared against the job-specific LWL. Here 25% of the batches are high priority and have a four times higher holding cost than the rest.



$X \sim \mathrm{Exp}(1)$

Figure 5: The fraction of batches split to several servers when minimizing the mean holding cost per batch in the example with high- and low priority classes.

patterns and the pool of parallel servers). It appears to be beneficial for FPI, if the basic policy associates each queue with a certain *role*, e.g., one queue is expected to receive mainly short jobs and another the longer ones.

Assigning a batch to servers with a given cost structure is a very challenging and important problem, e.g., for the performance analysis and optimization of parallel and distributed systems. Only few optimality results exist for the traditional dispatching problem, and the opportunity to split tasks to parallel servers simply complicates the situation. The FPI step tells us to which direction a given policy should be developed to, while finding the optimal policy would require consecutive policy iteration rounds. Unfortunately, it is more difficult to determine the value function once the dispatching policy no longer is batch specific. On the other hand, the FPI step often yields the largest improvement towards the optimal solution.

[1] C. Archetti, M. Savelsbergh, M. Speranza, To split or not to split: that is the question, Transportation Research - Part E 44 (1) (2008) 114–123.

[2] A. Ephremides, P. Varaiya, J. Walrand, A simple dynamic routing problem, IEEE Transactions on Automatic Control 25 (4) (1980) 690–693.

[3] W. Whitt, Deciding which queue to join: Some counterexamples, Operations Research 34 (1) (1986) 55–62.

[4] Z. Liu, D. Towsley, Optimality of the round-robin routing policy, Journal of Applied Probability 31 (2) (1994) 466–475.

[5] H. Feng, V. Misra, D. Rubenstein, Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems, Performance Evaluation 62 (1-4) (2005) 475–492.

[6] E. Bachmat, H. Sarfati Analysis of SITA policies, Performance Evaluation 67 (2) (2010) 102–120.

[7] L. Flatto, S. Hahn, Two parallel queues created by arrivals with two demands, SIAM Journal on Applied Mathematics 44 (1984) 1041–1053.

[8] O. Boxma, G. Koole, Z. Liu, Queueing-theoretic solution methods for models of parallel and distributed systems, in: Performance Evaluation of Parallel and Distributed Systems Solution Methods, 1994, pp. 1–24.

[9] E. Hyytiä, A. Penttinen, S. Aalto, Size- and state-aware dispatching problem with queue-specific job sizes, European Journal of Operational Research 217 (2) (2012) 357–370.

[10] E. Hyytiä, S. Aalto, A. Penttinen, Minimizing slowdown in heterogeneous size-aware dispatching systems, ACM SIGMETRICS Performance Evaluation Review 40 (2012) 29–40.

[11] E. Hyytiä, S. Aalto, A. Penttinen, J. Virtamo, On the value function of the M/G/1 FCFS and LCFS queues, Journal of Applied Probability 49 (4) (2012).

[12] M. E. Crovella, M. Harchol-Balter, C. D. Murta, Task assignment in a distributed system: Improving performance by unbalancing load, in: Proc. of SIGMETRICS '98, Madison, Wisconsin, USA, 1998, pp. 268–269.

[13] M. Harchol-Balter, M. E. Crovella, C. D. Murta, On choosing a task assignment policy for a distributed server system, Journal of Parallel and Distributed Computing 59 (1999) 204–228.

[14] M. Harchol-Balter, A. Scheller-Wolf, A. R. Young, Surprising results on task assignment in server farms with high-variability workloads, in: Proc. of SIGMETRICS, ACM, New York, NY, USA, 2009, pp. 287–298.