IIE Transactions
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/uiie20

The "W" network and the dynamic control of unreliable flexible servers

Soroush Saghafian [a], Mark P. Van Oyen [a] & Bora Kolfal [b]

[a] Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 48109, USA

[b] School of Business, University of Alberta, Edmonton, AB, T6G2R6, Canada

Available online: 24 May 2011

To cite this article: Soroush Saghafian, Mark P. Van Oyen & Bora Kolfal (2011): The "W" network and the dynamic control of unreliable flexible servers, IIE Transactions, 43:12, 893-907

To link to this article: http://dx.doi.org/10.1080/0740817X.2011.575678

IIE Transactions (2011) **43**, 893–907

ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/0740817X.2011.575678

The "W" network and the dynamic control of unreliable flexible servers

SOROUSH SAGHAFIAN,[1] MARK P. VAN OYEN[1,*] and BORA KOLFAL[2]

[1]*Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109, USA*
E-mail: vanoyen@umich.edu
[2]*School of Business, University of Alberta, Edmonton, AB T6G2R6, Canada*

Received May 2010 and accepted March 2011

This article addresses the problem of effectively assigning partially flexible resources to various jobs in Markovian parallel queueing systems with heterogeneous and unreliable servers. Attention is focused on a structure forming a "W" and it is found that this design is highly efficient; it requires only a small amount of cross-training but often performs almost as well as a fully cross-trained system. It is shown that (even allowing disruptions) a version of the $c\mu$ rule, which prioritizes serving the "fixed task before the shared," is optimal under some conditions. Since the optimal policy is complex in general, a powerful and yet simple control policy is developed. This policy (which is implementable in any parallel queueing system) defines a simple measure of workload costs and assigns each server to the queue with the Largest Expected Workload Cost (LEWC). Thus, it effectively combines the intuition underlying two widely used policies: (i) the load-balancing objective in serving the Longest Queue (LQ); and (ii) the greedy cost minimization emphasis of the $c\mu$ rule. Extensive numerical tests show that LEWC performs well in comparison with four key policies: optimal, LQ, cμ, and generalized cμ (Gcμ). The stability of the LEWC, LQ, and Gcμ policies is proved.

[Supplementary materials are available for this article. Go to the publisher's online edition of *IIE Transactions* for additional appendices (detailed proofs, additional analyses, data sets, etc.).]

Keywords: Flexible servers, Markov decision process, control of queues, unreliable servers, stochastic resource allocation

1. Introduction

The use of cross-trained workers (or flexible machines) in manufacturing or service sectors provides flexibility by dynamically shifting workers (workloads) to respond to volatile demands, machine/worker availabilities, congestion, etc. Typically, agents/workers are *partially flexible*, in that they are trained to serve a limited number of different requests (task types) so as to achieve a cost-effective level of flexibility.

The literature on the modeling and analysis of flexibility includes the following three themes: (i) *system design* of specific paradigms for creating flexibility to maximize an objective; (ii) *server scheduling and control* policies to reap the benefits of flexibility; and (iii) *performance analysis* of specific systems and policies. Our work contributes to all three themes, especially the second theme.

System design, the first theme, motivates the development of methodology to determine which capabilities a server should be endowed with (see, for instance, Jordan and Graves (1995); Aksin and Karaesmen (2002); Hopp *et al.* (2004); Hopp and Van Oyen (2004); Iravani *et al.* (2005); Iravani *et al.* (2007); Bassamboo *et al.* (2009); Chou *et al.* (2010); Andradottir *et al.* (2010)). We first analyze parallel (Markovian) queueing systems with general structures (i.e., arbitrary number of queues and servers in parallel with general skill/capability sets for the servers) to prove properties such as stability. Then we focus on the "W" paradigm/structure for parallel operations that are "make-to-order."

To motivate the "W" paradigm, consider the small customer support center illustrated in Fig. 1. In this example, both agents can handle phone calls. However, only one of them is responsible for supporting customers through the Internet using chat and email. The other agent is provided the resources to handle postal mail and faxes. We refer to the queueing structure of Fig. 1 as a "W" queueing network (since it forms a "W" with respect to the server skills and workflow). For the manager of the system illustrated in Fig. 1, different request types have different response time urgencies. For instance, a quick reply to a chat or an email

*Corresponding author

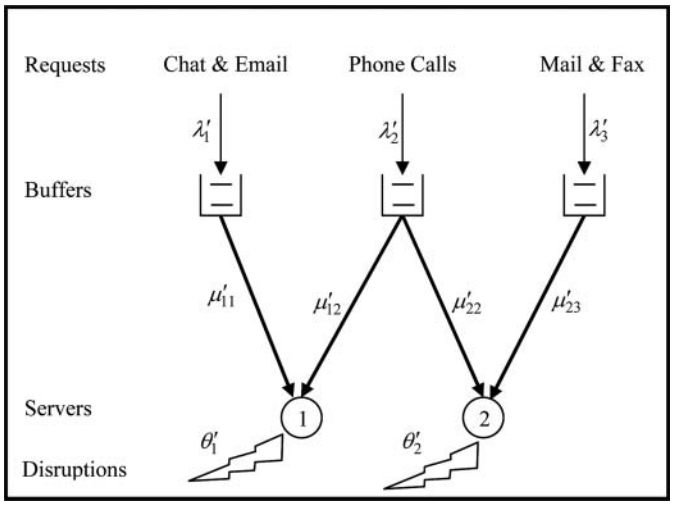


Fig. 1. An example of a small customer support center (the "W" structure).

request is often more important than a fast response to a postal mail or to a fax. Phone calls on hold (waiting in queue) are also usually more urgent than a mail or a fax request.

Generally, in such systems, a per unit of time cost (or relative weight) of h'_i can be assigned to holding a request of type i. Additionally, the servers are usually *heterogeneous*: they have different skill levels (service rates) in serving different job types. In general, one can model the service of a request of type i by server j as occurring with a rate of $\mu'_{ji} \geq 0$ (where zero indicates that server j lacks skill i). Moreover, servers might be subject to *stochastic disruptions* occurring with a rate of $\theta'_j \geq 0$ for server j, which represents the time lost due to an IT disruption, unplanned absences (e.g., unexpected meetings), etc. When disrupted, server j returns to a working state after an expected r'^{-1}_j units of time, which represents its average "repair" time. Assuming a type i request comes to the system at rate λ'_i, the manager of such a system needs to know how to assign the agents to different requests in *real time* to obtain good performance and extract the most benefit from the partial flexibility of the servers.

Conceptually, the "W" structure can be observed in many systems in practice. One of the situations where a "W" structure may naturally arise is where tasks performed by the servers have a wide variety and can be classified as tasks that are server specific and tasks that are shared between servers. Consider, for example, a small clinic with a physician and a nurse working together. There is a set of tasks that would be performed by the nurse (e.g., taking blood pressure and other diagnostic tests, administering medications, and basic treatments) and there is also a separate set of tasks that would be performed by the physician (e.g., diagnosis of diseases and injuries, prescribing medications and treatments, and performing higher skill medical procedures). Additionally, there is a set of tasks that could be performed by either the nurse or the physician, depending on the workloads of the nurse and the physician (e.g., diagnostic tests, bandaging, and giving home self-care or follow-up instructions).

The "W" structure may also arise from considerations of demand workload and service capacity. For example, the shared demand type may represent the demand class for which a server cannot provide enough capacity and, thus, capacity can be shifted via cross-training. Moreover, there are often tasks that are not cost-effective to cross-train. This may be caused by the training/certification expense of the skill, the difficulty in obtaining workers competent at that skill, or the infrastructure and layout that makes the cross-functionality ineffective.

We contribute to the first theme of flexibility research, system design, by showing that the "W" structure achieves most of the potential performance with two servers and three job types, supporting the notion that *a little flexibility goes a long way* (which has been a theme of several papers such as Jordan and Graves (1995) and Bassamboo *et al.* (2009) for some different structures). Considering the expense of cross-training servers and, more important, application-specific obstacles to cross-train certain task types, the frugality of the "W" design makes it widely useful in application.

We contribute to the second theme of flexibility research, control, by generating insights into effective mechanisms for the control of servers in the "W" design as well as systems with any general structure. Specifically, for the "W" structure, we rigorously establish a partial characterization of the $c\mu$ rule (i.e., the weighted shortest expected processing policy) as an optimal policy under certain operating conditions. We also develop a high-performance heuristic index policy, termed *Largest Expected Workload Cost* (LEWC), and benchmark it relative to the optimal policy for a large test suite. The proposed LEWC index policy, however, is not specific to the "W" design and can be implemented in any parallel queueing system.

Even after ignoring possible disruptions, the control problem that we consider in this article is a difficult and still an open area of research. For instance, Bell and Williams (2001, p. 615) considered the "N" structure (with reliable servers), a special case of the "W" with the third demand stream removed, and noted that even for the "N" "the problem of finding a control policy that minimizes a cost associated with holding jobs in the system is notoriously difficult." The "W" model is a significant departure from the "N," because it has two partially flexible servers, whereas the "N" has two extremes: one inflexible and one fully flexible server. Server disruptions further complicate the problem. In addition to identifying sufficient conditions under which the well-known greedy $c\mu$ policy is optimal, our numerical analysis provides further insights for situations where those conditions do not hold. Particularly, the optimal policy is a *state-dependent threshold-type*

policy characterized by four switching surfaces in the cases studied.

Addressing the system design agenda of the first theme, Iravani *et al.* (2005); and Iravani *et al.* (2007, 2011) have developed methodologies such as structural flexibility and capability flexibility for estimating the better of alternative cross-training architectures with respect to mean waiting time. To test these methods, the above papers primarily used the Longest Queue (LQ) as the control policy. In this article, we propose LEWC as a more effective policy. It should be noted that even for a particular structure such as "W," performance analysis (the third theme) under the optimal policy is difficult. Thus, we provide a careful Markov Decision Process (MDP)-based numerical benchmarking study that gives insights into the optimal policy as well as LEWC, LQ, $c\mu$, and Generalized $c\mu$ ($Gc\mu$) with quadratic holding cost (also referred to as *max-weight*). We find that not only does the LEWC heuristic clearly outperform LQ, $c\mu$, and $Gc\mu$, but it is also a near-optimal policy with a relatively small optimality gap. Moreover, we establish its stability. Since LEWC can be used for the control of servers in systems with any flexibility structure, the obtained results introduce LEWC as a promising policy for future research into the design of flexible structures with arbitrary topologies. This is particularly useful because the comparison of alternative flexibility/queueing designs under their optimal control policies is computationally intractable for large systems.

The rest of this article is organized as follows. Section 2 briefly reviews some related studies. Section 3 formulates the problem using an MDP framework and identifies some attributes applicable to a parallel queueing system with a general flexibility structure. Section 4 presents the results on the "W" structure, describes the proposed LEWC heuristic, and extensively tests its performance.

2. Literature survey

When there is a single server in the system that is fully flexible and has memoryless service times, Buyukkoc *et al.* (1985) and Walrand (1988) show that the well-known $c\mu$ policy is optimal. The $c\mu$ rule is a very intuitive and easy control policy to implement; however, it may perform poorly when partial flexibility is introduced, as is the case with the "W." It remains, however, optimal under some conditions. For instance, Down and Lewis (2010) prove the optimality of the $c\mu$ rule for an "N" structure under some special conditions. Veatch (2010) shows the optimality of $c\mu$ for systems without disruptions where servers collaborate on jobs and special conditions are satisfied.

In parallel systems, which is our focus, the literature mainly considers the control problem in the heavy-traffic regime (see, for instance, Van Mieghem (1995); Harrison (1998); Harrison and López (1999); Bramson and Williams (2000); Bell and Williams (2001, 2005); Meyn (2003); Mandelbaum and Stolyar (2004)). The literature, however, lacks policies that are effective for a *wide range of utilizations*. Our target in this article is on systems in the utilization range of 70% to 90%.

The problem of dynamically assigning servers to jobs has also been studied under the throughput maximization objective (see, for instance, Andradóttir *et al.* (2001, 2003, 2007); Armony and Bambos (2003); Dai and Lin (2005)). Among these papers, Andradottir *et al.* (2007) is most closely related to our work since it also allows for disruptions. However, throughput maximization is appropriate only for systems in which delay is not a major concern, and in most cases it is an easier problem to analyze.

Work on the benefit of flexibility to compensate for the risk of disruptions is also related to our work. For this stream of research, we refer interested readers to Andradottir *et al.* (2007), Saghafian and Van Oyen (2011), and the references therein.

3. General characteristics

This section addresses general Markovian parallel queueing structures with partially flexible and possibly unreliable servers. That is, we consider Markovian parallel queueing systems with an arbitrary number of servers, arbitrary number of customer classes, and arbitrary flexibility structures. We allow even more generality by allowing for a stochastic disruption/repair process unique to each server. We first describe our model and then formulate it using an MDP framework.

3.1. *The model*

Consider a queueing system represented by a bipartite graph $G = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is partitioned to two finite sets: $\mathcal{N}_c = \{1, \ldots, m\}$ for customer/jobs classes, and $\mathcal{N}_s = \{m+1, \ldots, m+n\}$ for servers/machines (see Fig. 2; A labeling $\{1, \ldots, n\}$ might be used for servers when it does not generate any confusion.) Arrivals of customers of class $i \in \mathcal{N}_c$ follow a Poisson process with rate $\lambda'_i \in \mathbb{R}^+$, and server $j \in \mathcal{N}_s$ can serve a customer of class $i \in \mathcal{N}_c$

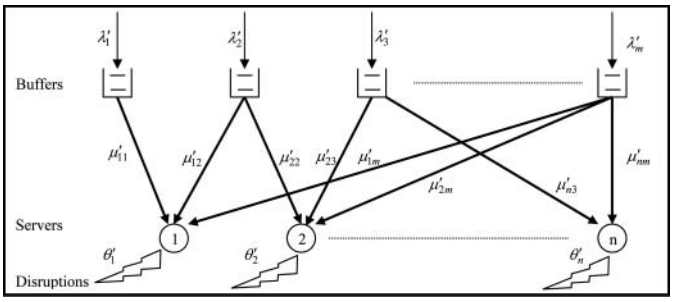


Fig. 2. A general parallel queueing system with server disruptions and arbitrary flexibility structure.

with an exponentially distributed amount of time with rate $\mu'_{ji} \in \mathbb{R}^+$. In the graph $G, (i, j) \in \mathcal{E} \subseteq \mathcal{N}_c \times \mathcal{N}_s$ if, and only if, $\mu'_{ji} > 0$. We let $\mathcal{S}_j = \{i : (i, j) \in \mathcal{E}\}$ denote the skill set or "capabilities" of server j and $\mathcal{S}_i^{-1} = \{j : (i, j) \in \mathcal{E}\}$ denote the servers capable of serving class i. To allow for server unreliability, disruptions to server $j \in \mathcal{N}_s$ occur according to a Poisson process with rate $\theta'_j \geq 0$ (equality holds if server j is completely reliable). Note that we focus on systems for which disruptions occur at the same rate whether or not the server is in use. For example, unplanned employee absence, a power outage, or an economic disruption may happen independently of server idleness. Once a server is disrupted, it immediately undergoes a repair process that takes an exponentially distributed amount of time with rate $r'_j > \theta'_j$ for server j. All above-mentioned stochastic processes are considered to be independent of each other.

Let $\mathbf{h}' = (h'_1, \ldots, h'_m)$, where h'_i denotes the per unit time (inventory) holding cost associated with holding a customer of class i. The objective is to find an optimal resource allocation (or server assignment) policy to minimize the average holding cost of the system assuming that the servers cannot collaborate on the same job (unless otherwise mentioned), but service preemption is permitted. To achieve this goal, let $\mathbf{X}^\pi(t) = (X_1^\pi(t), \ldots, X_m^\pi(t))$ where $X_i^\pi(t)$ denotes the number of class i customers in the system at time t under policy π. A policy is then optimal if it achieves the following optimal cost:

$$Z^* = \inf_{\pi \in \Pi} Z^\pi = \inf_{\pi \in \Pi} \left\{ \sum_{i \in \mathcal{N}_c} h'_i L_i^\pi \right\}, \tag{1}$$

where Π is the set of all admissible policies, and L_i^π denotes the long-run average number of class i customers in the system under policy π. This latter measure can be computed as

$$L_i^\pi = \limsup_{T \to \infty} \frac{1}{T} \int_0^T E\big[X_i^\pi(s)\big]\, \mathrm{d}s. \tag{2}$$

3.2. *Formulation of the MDP*

For $j \in \mathcal{N}_s$, let $a_j(t) = 1$ denote that server j is available (i.e., not disrupted) at time t and let $a_j(t) = 0$ otherwise. The state of the system is then a vector $\tilde{\mathbf{X}}(t) = (\mathbf{X}(t), \mathbf{a}(t))$ with state space $\mathcal{S} = \mathbb{Z}^{+m} \times \{0, 1\}^n$, where $\tilde{\mathbf{X}}_i(t) = X_i(t) \in \mathbb{Z}^+$ for $i \in \mathcal{N}_c$ and $\tilde{\mathbf{X}}_j(t) = a_j(t) \in \{0, 1\}$ for $j \in \mathcal{N}_s$. We use uniformization (see Lippman (1975)) to formulate the discrete time equivalent of the problem. Since $\theta'_j < r'_j$, we use the uniformization factor $\psi = \sum_{i \in \mathcal{N}_c} \lambda'_i + \sum_{j \in \mathcal{N}_s} r'_j + \sum_{j \in \mathcal{N}_s} \max_{i \in \mathcal{N}_c}\{\mu'_{ji}\}$ (where $0 < \psi < \infty$). Let $\lambda_i = \lambda'_i/\psi$, $\mu_{ji} = \mu'_{ji}/\psi$, $\theta_j = \theta'_j/\psi$ and $r_j = r'_j/\psi$ denote the parameters after uniformization corresponding to the transition probabilities in the underlying discrete Markov chain. Also, let α be a continuous-time discount rate and ξ be an exponential random variable with rate ψ denoting the length of one unit of time in the corresponding discrete Markov chain. The equivalent discount factor in discrete time is then:

$$\beta = E[e^{-\alpha \xi}] = \int_0^\infty (e^{-\alpha t})\,(\psi e^{-\psi t})\mathrm{d}t = \frac{\psi}{\alpha + \psi}. \tag{3}$$

Also, since the state of the system does not change in one period of the discrete time version, the equivalent instantaneous one period cost is

$$\mathbf{h}\mathbf{X}^{\mathrm{T}} = E\left[\int_0^\xi \mathbf{h}'\mathbf{X}^{\mathrm{T}} e^{-\alpha t}\, \mathrm{d}t\right] = \frac{1-\beta}{\alpha}\, \mathbf{h}'\mathbf{X}^{\mathrm{T}} = \frac{\mathbf{h}'}{\psi + \alpha}\mathbf{X}^{\mathrm{T}}, \tag{4}$$

and so $\mathbf{h} = \mathbf{h}'/(\psi + \alpha)$. The finite-horizon optimal expected discounted cost can then be computed using the following optimality equation defined for every $\tilde{\mathbf{X}} \in \mathcal{S}$ and $n \in \mathbb{Z}^+$:

$$\begin{aligned}
&V_{n+1,\beta}(\tilde{\mathbf{X}}) \\
&= \mathbf{h}\mathbf{X}^{\mathrm{T}} + \beta\Bigg[\sum_{i \in \mathcal{N}_c} \lambda_i\, V_{n,\beta}(A_i\tilde{\mathbf{X}}) + \sum_{j \in \mathcal{N}_s}[\theta_j\, a_j\, V_{n,\beta}(B_j\tilde{\mathbf{X}}) \\
&\quad + r_j\,(1 - a_j)\, V_{n,\beta}(R_j\tilde{\mathbf{X}})] \\
&\quad + \min_{\mathbf{u} \in \mathcal{U}(\tilde{\mathbf{X}})} \Bigg\{ \sum_{i \in \mathcal{N}_c} \sum_{j \in \mathcal{N}_s} \mathbb{1}\{u_j = i\}\, \mu_{ji}\, V_{n,\beta}(D_i\tilde{\mathbf{X}}) \\
&\quad + \Bigg(1 - \sum_{i \in \mathcal{N}_c} \lambda_i - \sum_{i \in \mathcal{N}_c} \sum_{j \in \mathcal{N}_s} \mathbb{1}\{u_j = i\}\, \mu_{ji} \\
&\quad - \sum_{j \in \mathcal{N}_s}[\theta_j\, a_j + r_j\,(1 - a_j)]\Bigg) V_{n,\beta}(\tilde{\mathbf{X}})\Bigg\}\Bigg],
\end{aligned} \tag{5}$$

where $V_{n,\beta}(\tilde{\mathbf{X}})$ represents the optimal cost of an n-period problem starting at state $\tilde{\mathbf{X}}$, $\mathbb{1}\{\cdot\}$ is the indicator function, and the initial condition is $V_{0,\beta}(\tilde{\mathbf{X}}) = 0$ for every $\tilde{\mathbf{X}} \in \mathcal{S}$. In this optimality equation, the arrival, departure, repair, and breakdown state transition operators for $i \in \mathcal{N}_c$ and $j \in \mathcal{N}_s$ are denoted by $A_i\tilde{\mathbf{X}} = \tilde{\mathbf{X}} + \mathbf{e}_i$, $D_i\tilde{\mathbf{X}} = \tilde{\mathbf{X}} - \mathbf{e}_i$, $R_j\tilde{\mathbf{X}} = \tilde{\mathbf{X}} + \mathbf{e}_j$, and $B_j\tilde{\mathbf{X}} = \tilde{\mathbf{X}} - \mathbf{e}_j$, respectively, where $\mathbf{e}_i$ ($\mathbf{e}_j$) is a vector with the same dimension as $\mathcal{S}$ with a one in ith (jth) position and zeros elsewhere. Moreover, the control action is the vector $\mathbf{u} = (u_j \in \mathcal{N}_c \cup \{0\}, \forall j \in \mathcal{N}_s)$ where $u_j = i \in \mathcal{N}_c$ if server j is assigned to serve class i, and $u_j = 0$ if it is not assigned to any class. The set of admissible control actions at state $\tilde{\mathbf{X}}$ is denoted by a set of vectors $\mathcal{U}(\tilde{\mathbf{X}})$, where:

$$\begin{aligned}
\mathcal{U}(\tilde{\mathbf{X}}) = \Bigg\{ &\mathbf{u} = (u_j \in \mathcal{N}_c \cup \{0\} \text{ s.t. } \forall i \in \mathcal{N}_c : \mathbb{1}\{u_j = i\} \\
&\leq a_j\, \mathbb{1}\{i \in \mathcal{S}_j\},\ \sum_{j \in \mathcal{N}_s} \mathbb{1}\{u_j = i\} \leq X_i)\Bigg\}.
\end{aligned} \tag{6}$$

That is, server j cannot be assigned to class i if it is disrupted, if it lacks skill i, or if the number of class i jobs is insufficient.

Similar to Equation (5), the optimal average inventory holding cost can be computed using an MDP with the following average-cost optimality equation:

$$J(\tilde{\mathbf{X}}) + Z_{\mathrm{U}}^{*} = \frac{\mathbf{h}'}{\psi}\mathbf{X}^{\mathrm{T}} + \sum_{i\in\mathcal{N}_{\mathrm{c}}}\lambda_i J(A_i\tilde{\mathbf{X}}) + \sum_{j\in\mathcal{N}_{\mathrm{s}}}[\theta_j a_j J(B_j\tilde{\mathbf{X}}) + r_j(1-a_j)J(R_j\tilde{\mathbf{X}})] + \min_{\mathbf{u}\in\mathcal{U}(\tilde{\mathbf{X}})}\left\{\sum_{i\in\mathcal{N}_{\mathrm{c}}}\sum_{j\in\mathcal{N}_{\mathrm{s}}}\mathbb{1}\{u_j=i\}\mu_{ji}J(D_i\tilde{\mathbf{X}}) + \left(1-\sum_{i\in\mathcal{N}_{\mathrm{c}}}\lambda_i - \sum_{i\in\mathcal{N}_{\mathrm{c}}}\sum_{j\in\mathcal{N}_{\mathrm{s}}}\mathbb{1}\{u_j=i\}\mu_{ji} - \sum_{j\in\mathcal{N}_{\mathrm{s}}}[\theta_j a_j + r_j(1-a_j)]\right)J(\tilde{\mathbf{X}})\right\}, \quad (7)$$

where $J(\tilde{\mathbf{X}})$ is a relative cost function, Z_{U}^{*} denotes the optimal per period average cost in the uniformized problem, and $Z^{*} = \psi\, Z_{\mathrm{U}}^{*}$ is the optimal per period average cost of the original problem. The next section first analyzes the stability conditions of the the general queueing system under consideration. Then it provides another method to compute the optimal average cost and the relative function $J(\tilde{\mathbf{X}})$ using the finite-horizon version of the problem (i.e., value iteration).

3.3. *Stability*

It is important first to identify the stability region of the system for several reasons. In addition to several interesting theoretical considerations, it provides an important practical design guideline. We define the general queueing network under consideration to be stabilizable if, and only if, there exists a policy $\pi \in \Pi$ such that $Z^{\pi} = \sum_{i\in\mathcal{N}_{\mathrm{c}}} h_i' L_i^{\pi} < \infty$. This is equivalent to the existence of a *finite* mean equilibrium distribution of the underlying stochastic process $\{\mathbf{X}(t), t \geq 0\}$. To check the stability of the underlying system with partially flexible and unreliable servers, we develop and implement the following Linear Program (LP; in the spirit of Harrison and López (1999) and Andradottir *et al.* (2007)). Our LP maximizes the minimum *excess service capacity*, τ, that can be provided for *all* customer classes.

LP 1:

$$\text{Max } \tau, \quad (8)$$

subject to:

$$\sum_{j\in\mathcal{S}_i^{-1}} y_{ji}\left(\frac{r_j}{\theta_j + r_j}\right)\mu_{ji} \geq \lambda_i + \tau, \quad \forall i \in \mathcal{N}_{\mathrm{c}}, \quad (9)$$

$$\sum_{i\in\mathcal{S}_j} y_{ji} \leq 1, \quad \forall j \in \mathcal{N}_{\mathrm{s}}, \quad (10)$$

$$y_{ji} \geq 0, \quad \forall j \in \mathcal{N}_{\mathrm{s}}, \forall i \in \mathcal{S}_j. \quad (11)$$

In this LP, we introduce the decision variable y_{ji} ($j \in \mathcal{N}_{\mathrm{s}}, i \in \mathcal{S}_j$) to denote the long-run proportion of time that server j is "assigned" to work on class i (including the times during which server j is disrupted) when the arrival rate of class i is $\lambda_i + \tau$. Notice that (using either renewal theory or a two-state Markov chain model of disruption and repair process) server j in steady state is available $r_j/(\theta_j + r_j)$ percent of the time. Thus, $y_{ji}\,(r_j/(\theta_j + r_j))$ represents the long-run proportion of the time that server j is available and working on class i (when the arrival rate of class i is $\lambda_i + \tau$); and $y_{ji}\,(r_j/(\theta_j + r_j))\mu_{ji}$ is the corresponding long-run average capacity offered to class i by server j given y_{ji}. Hence, from constraint (9) we see that objective function (8) maximizes the minimum excess capacity among all classes. Constraint (10) (together with constant (11)) sets an upper bound for the total fraction of time that a server can be assigned to a specific class. The following theorem, based on fluid model analysis (see, for instance, Dai (1999)) and similar to some results presented in the literature (see for instance, Andradottir *et al.* (2007)), relates the above LP to the stabilizability of the system. This theorem provides a tool to ensure that the class of finite cost policies is not empty, and hence the optimization in Equation (1) is of interest. See Online Appendix A for all of the proofs.

Theorem 1. *(*stability*). Let τ^* be the optimal objective value of LP 1. Then:*

(i) the system is stabilizable (i.e., $\exists\pi \in \Pi$ s.t. $Z^{\pi} < \infty$) if $\tau^* > 0$;

(ii) the system is not stabilizable (i.e., $\forall\pi \in \Pi : Z^{\pi} = \infty$) if $\tau^* < 0$.

Now that we have a tool to check stabilizability, we can take one step further and (i) guarantee the existence of an optimal *stationary* policy; and (ii) establish the convergence of the finite-horizon problem to the average-cost case (both in the cost and in the policy). Indeed, we can establish a convenient alternative approach to find an optimal average-cost policy by stating that (i) it is sufficient to restrict attention to the class of stationary policies; and (ii) solving the finite-horizon version of the problem defined in Equation (5) can provide both the average-cost optimal value Z^* and the average-cost optimal policy π^*.

Theorem 2. *(*stationary policy and convergence*). If $\tau^* > 0$, then:*

(i) there exists an average-cost optimal stationary policy;

(ii) the optimal average cost can be computed by: $Z^* = \inf_{\pi\in\Pi}\{\sum_{i\in\mathcal{N}_{\mathrm{c}}} h_i' L_i^{\pi}\} = \lim_{\beta\to 1^-}\lim_{n\to\infty}\psi(1-\beta)V_{n,\beta}(\tilde{\mathbf{X}})$;

(iii) the relative cost function $J(\tilde{\mathbf{X}})$ defined in Equation (7) satisfies: $J(\tilde{\mathbf{X}}) = \lim_{\beta\to 1^-}\lim_{n\to\infty}[V_{n,\beta}(\tilde{\mathbf{X}}) - V_{n,\beta}(\mathbf{0})]$;

(iv) Let $\pi_{n,\beta}$ denote an optimal policy for the n-period (discounted cost) problem. Then any limit point π_{β} of the sequence $\{\pi_{n,\beta}\}_{n\geq 1}$ (as $n \to \infty$) is optimal

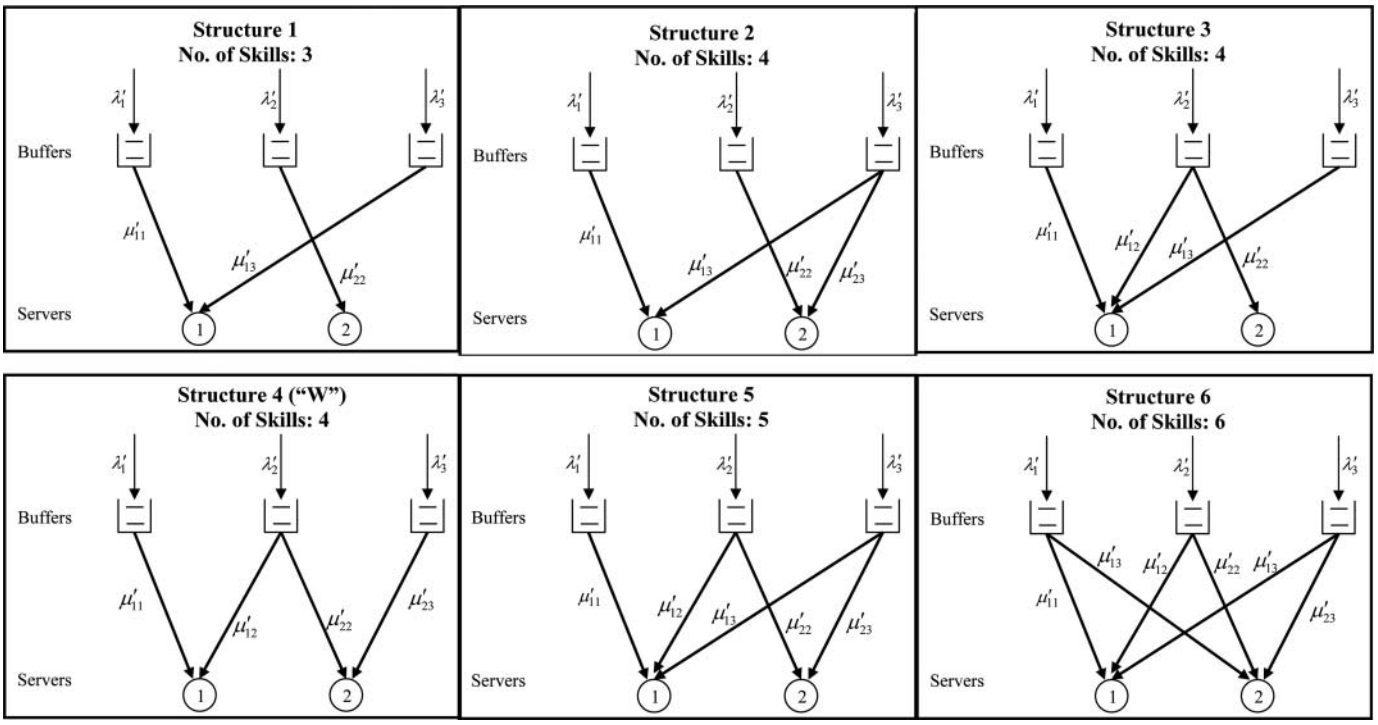


Fig. 3. Various possible structures with $|\mathcal{N}_c| = 3$ and $|\mathcal{N}_s| = 2$.

for the infinite-horizon discounted cost. Moreover, any limit point of the sequence $\{\pi_\beta\}_{\beta\in(0,1)}$ *(as* $\beta \to 1^-$*) is average-cost optimal.*

In the search for effective mechanisms to control the servers, we are able to restrict our attention to the class of policies that do not allow for unforced idling. This is shown in Appendix A (see Lemma 1 and Proposition 1), where we establish this result based on a proof of the monotonicity of the value function.

4. The "W" structure

In the previous section, we presented some characteristics applicable to any Markovian parallel queueing system with an arbitrary server flexibility structure. In this section, to develop more insights, we consider a special structure forming a "W" (see Fig. 1 or Structure 4 in Fig. 3). This structure is an especially effective paradigm for systems with three demand types and two servers. It should be noted that the "N" structure, widely studied in the literature, is a special case of a "W" with $\mu'_{23} = \lambda'_3 = 0$.

The next section shows that the "W" is an efficient design that requires only a little cross-training to achieve a performance almost as good as any design with two servers and three job types. Since cross-training the servers is costly (and sometimes infeasible) in practice, this observation shows that for systems with three demand types and two servers, instead of fully cross-training every server, it is sufficient to make them capable to serve a shared task in addition to their dedicated/fixed one and form a "W" structure.

4.1. *The "W" structure: an efficient system design*

From a system design perspective, it is crucial to understand the effective ways of cross-training servers. Please note that in this section we focus on congestion (and mean wait), so all holding costs are set to one. To understand the design problem, consider the various possible designs with three customer classes and two servers illustrated in Fig. 3. These six structures progressively add skills, except Structures 2, 3, and 4 (the "W"), which have the same number of skills. Thus, Structures 2, 3, and 4 also allow us to explore the sensitivity with respect to *where* the fourth skill is added. In Structures 3 and 4 the class with the highest arrival rate is the shared one, but it is not the case in Structure 2. Structure 2 is indeed a "W," where the shared task is not the one with the highest arrival rate (i.e., the middle class). The goal is to find an *efficient design* among these six structures. In other words, to improve the design of Structure 1, we address two questions: (i) *Where* should one implement flexibility/cross-training? and (ii) *How many* additional skills are adequate to get a reasonably good performance?

To answer these questions, we compare the performance of the above-mentioned structures under their optimal policies in various test suites (parameter settings) as presented in Table 1.C (see Online Appendix C). Notice that,

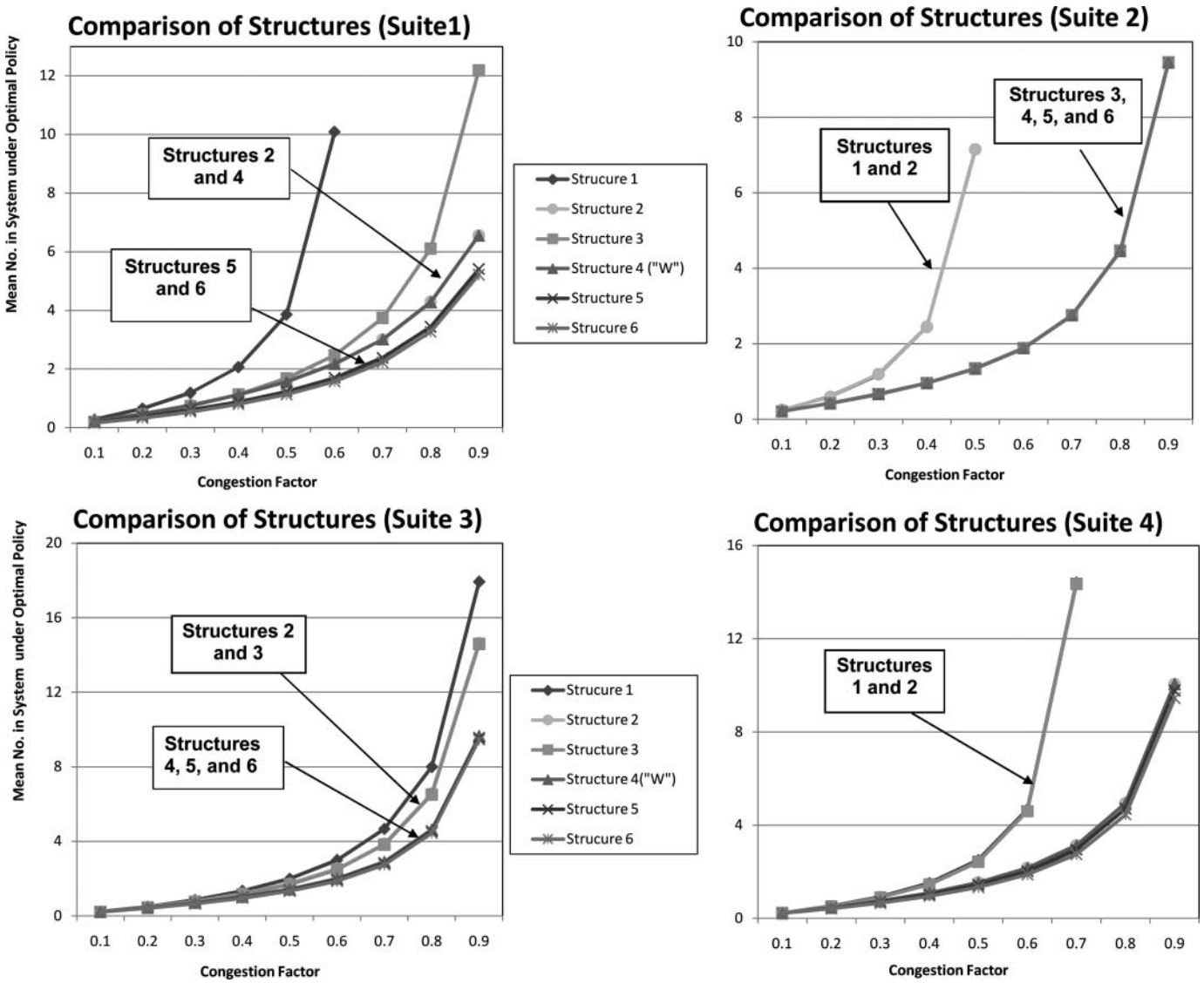


Fig. 4. Comparison of possible structures under four suites of parameters using the optimal policies.

considering the built-in symmetry in our test suites (symmetry between classes 1 and 3 as well as the symmetry in the speed of a server in serving different classes), the six structures considered in Fig. 3 cover all possible designs; any other (stabilizable) structure is homomorphic to one of these six structures. Figure 4 summarizes our computational results by depicting the optimal long-run average number of customers in each of these six structure for our test suite and under various congestion factors (ρ in Table 1.C). The mean (i.e., long-run average) number of customers (or jobs) in the system under the optimal policy is computed by numerically solving the average-cost MDP optimality Equation (7) with $h'_i = 1\ (\forall i \in \mathcal{N}_c)$.

The results depicted in Fig. 4, which is a summary of optimally solving 6 (structures) $\times$ 4 (suites) $\times$ 9 (congestion factors) $=$ 216 problem instances, confirm that (i) flexibility usually has a diminishing rate of return; (ii) a little flexibility can go a long way; and (iii) it usually matters where we add the additional flexibility, which has been elaborated on in studies such as Jordan and Graves (1995), Hopp *et al.* (2004), Iravani *et al.* (2005), and Bassamboo *et al.* (2009). The primary intent of this section is, however, to reveal the following insight about the "W."

Insight. Structure 4, the "W" (or to be precise the "W" with the proper task being shared), is an efficient design where a little cross-training can achieve most of the flexibility of a fully flexible network (i.e., Structure 6). In test suites 2, 3, and 4, the "W" is almost as good as Structure 6 and in test suite 1, the "W" is still an efficient architecture. This observation is especially important considering the expense of cross-training servers in most practical situations and reveals the benefit of implementing a "W" structure.

It should be noted that similar characteristics have been shown in the literature for chaining (see, for instance, Jordan and Graves (1995) and Hopp *et al.* (2004)) and tailored pairing (Bassamboo *et al.* (2009)), but the "W" is not a special case of those structures. Concurrent research in Andradottir *et al.* (2010) takes an alternate approach to analyzing the "W" and other structures with respect to throughput.

4.2. *Dynamic control of servers in the "W" structure*

The previous section examined the benefit of implementing a "W" structure; however, this benefit cannot be fully achieved without efficiently assigning servers to jobs in real-time. Hence, the remaining question is *what control policy should be used in real-time to extract the most benefit from the limited flexibility of servers in this design?* The answer to this question will also provide insight into the control of more complex queueing structures with partially flexible servers. We first state a corollary of Theorem 1 to partially characterize the stability region of a "W" design in more insightful expressions.

Corollary 1. *(*stability of "W"*). Consider a "W" structure under stochastic disruptions (or without them as a degenerate case). Let*

$$\rho_1 = \lambda_1 \Big/ \left(\mu_{11}\frac{r_1}{r_1+\theta_1}\right) \text{ and } \rho_3 = \lambda_3 \Big/ \left(\mu_{23}\frac{r_2}{r_2+\theta_2}\right).$$

Also, define effective service rates of the shared task as

$$\mu_{12}^{\text{eff}} = \mu_{12}\frac{r_1}{r_1+\theta_1} \text{ and } \mu_{22}^{\text{eff}} = \mu_{22}\frac{r_2}{r_2+\theta_2}.$$

The system is not stabilizable if $\max\{\rho_1, \rho_3\} > 1$. *On the other hand, if* $\max\{\rho_1, \rho_3\} < 1$, *the system is stabilizable if* $(1-\rho_1)\,\mu_{12}^{\text{eff}} + (1-\rho_3)\,\mu_{22}^{\text{eff}} > \lambda_2$.

Now we characterize the optimal control policy. Hereafter, we assume that the system under consideration is stabilizable. The following theorem shows the optimality of prioritizing the *fixed task before the shared* for every server under certain conditions. This policy is analogous to the well-known $c\mu$ ($h\mu$ in our notation) rule as a strict priority ordering for every server.

Theorem 3. *(*optimality of the $c\mu$ strict priority: fixed before shared*). For a "W" structure with stochastic disruptions (or without them as a degenerate case), if* $h'_1\,\mu'_{11} \geq h'_2\,\mu'_{12}$, $h'_3\,\mu'_{23} \geq h'_2\,\mu'_{22}$ *and either (i) server collaboration is allowed; or (ii) server collaboration is disallowed but* $\mu'_{12} \geq \mu'_{11}$ *and* $\mu'_{22} \geq \mu'_{23}$ *hold, then the* $c\mu$ *priority rule is optimal for each server. That is, there exists an optimal policy under which every server, when not disrupted and regardless of the other server's allocation or disruption state, prioritizes its fixed task before the shared task whenever its fixed queue is not empty.*

The *fixed before shared* policy described in Theorem 3 can be viewed as an extension of the $c\mu$ rule for systems with partially flexible and unreliable servers. Indeed, Theorem 3 shows that this extension of the $c\mu$ policy is optimal for the "W" when the $c\mu$ index ($h\mu$ in our terminology) gives priority to the fixed task for each server (even when servers are unreliable). Under the conditions specified in Theorem 3, using the $c\mu$ strict priority rule for a server cannot result in the poor side effect of underutilizing the other server because of the specific flexibility structure. In other words, under these conditions, the $c\mu$ policy is *starvation free*; it maximizes the amount of job available to the other server and, hence, remains optimal. This insight might also hold for larger systems where $c\mu$ priorities are toward the fixed tasks for all servers. One nice feature of the above policy (i.e., fixed before shared) is that it defines a prescriptive rule for each server regardless of the other server's allocation or disruption state. This feature removes the need for servers to communicate in real time and provides a static (i.e., state-independent) rule that is easy to implement.

Our extensive MDP-based numerical computations show that the optimal policy is complex in general when $c\mu$ priorities are not toward the fixed tasks. Relaxing all such assumptions, we observe from our extensive numerical examples that the optimal policy for a general "W" structure with server disruptions is a *state-dependent threshold-type policy* that can be defined by four switching surfaces. See Online Appendix B for a detailed discussion on this observation and for numerical examples supporting it.

4.3. *An efficient heuristic policy: LEWC*

In practice, to be implementable, a policy must be easy enough to use. In the experience of the authors, managers and researchers working with on-demand service centers usually believe that a simple policy such as LQ is preferable to the cost/effort of implementing a complex policy in real-time (see Hopp and Van Oyen (2004) and Iravani *et al.* (2005)). However, our investigation has revealed that the popular LQ policy does not perform well in many situations. Moreover, as the previous section revealed, the optimal policy is complex and hard to implement in real time in practice. Therefore, in this section, we develop a heuristic policy that is both easy to implement and highly effective.

This policy balances the expected workload cost of queues. Indeed, this heuristic prescribes that every server (whenever not disrupted) in every decision epoch should prioritize serving the queue with the LEWC, regardless of the allocation or availability (i.e., disruption state) of other servers. Under this policy, a server does not need to know all of the queue lengths. Rather, each server needs visibility only of her/his duty area (skill set) to decide which queue to serve. Moreover, this policy eliminates the need for communication between servers, since each server can perform her/his job without the knowledge about the

other servers' allocations, availabilities, or workloads. As a result, a manager can prescribe a rule to each server *in advance* and ensure good overall performance. In large networks more general than the "W," this is a significant advantage. However, this policy is still dynamic and requires different actions for each server depending on the real-time length of the queues within the server's skill set.

To develop this policy, we first slightly modify LP 1 presented in Section 3.3; we call the new program LP 2. The objective of this LP (applicable to any general network and not only the "W") is to find allocations y_{ji} that maximize the minimum *percentage* excess capacity, $\tilde{\tau}$, among all queues.

LP 2:

$$\text{Max } \tilde{\tau}$$

subject to:

$$\sum_{j\in\mathcal{S}_i^{-1}} y_{ji}\left(\frac{r_j}{\theta_j+r_j}\right)\mu_{ji} \geq \lambda_i(1+\tilde{\tau}), \quad \forall i \in \mathcal{N}_c, \tag{12}$$

$$\sum_{i\in\mathcal{S}_j} y_{ji} \leq 1, \qquad \forall_j \in \mathcal{N}_s, \tag{13}$$

$$y_{ji} \geq 0, \qquad \forall j \in \mathcal{N}_s, \forall i \in \mathcal{S}_j. \tag{14}$$

Next, for each queue i (with queue length x_i), we develop an index $\mathcal{J}_i(x_i)$ to approximate the expected workload cost of that queue. We call this the *LEWC index* and define it as:

$$\mathcal{J}_i(x_i) = \frac{h_i \times x_i}{\sum_{j\in\mathcal{S}_i^{-1}} y^*_{ji}\,(r_j/(\theta_j+r_j))\,\mu_{ji}}, \tag{15}$$

where y^*_{ji} are the solution to LP 2, and $\mathcal{S}_i^{-1}$ represents the set of servers able to serve queue i. In fact, if all servers that can work on queue i are assigned to work there based on the steady-state allocations obtained from LP 2, a single job in the first position of queue i will take $[\sum_{j\in\mathcal{S}_i^{-1}} y^*_{ji}\,(r_j/(\theta_j+r_j))\,\mu_{ji}]^{-1}$ units of time to be served (assuming work sharing is permitted). Since x_i jobs are in queue i, it will take approximately (ignoring the waiting times) $x_i \times [\sum_{j\in\mathcal{S}_i^{-1}} y^*_{ji}\,(r_j/(\theta_j+r_j))\,\mu_{ji}]^{-1}$ units of time to serve all the jobs in queue i. This generates a workload cost of $\mathcal{J}_i(x_i)$ for queue i. It should be clear that the LEWC index also accounts for other system parameters, such as arrival rates, disruption rates, and repair rates, through the optimal solutions y^*_{ji}. Therefore, LEWC incorporates not only the load-balancing logic of LQ and the greedy cost minimization of $c\mu$ but also considers utilizations via solutions y^*_{ji}. The LEWC heuristic policy follows.

LEWC Algorithm:

Step 1. Solve LP 2 to obtain the optimal allocations y^*_{ji}.

Step 2. At the current state, $\tilde{\mathbf{X}}$, use Equation (15) to compute indexes $\mathcal{J}_i(x_i)$ for all queues (i.e., $i \in \mathcal{N}_c$). Then assign each available server j to the queue $i^*_j = \text{argmax}_{i\in\mathcal{S}_j}\,\mathcal{J}_i(x_i)$; i.e., to the queue with the largest LEWC index among the queues that it can serve. If two or more queues have the same index, break the tie by assigning the server to the queue with the smallest label (i.e., the left most queue in our diagrams).

The following theorem states that our proposed policy stabilizes the system, if the system is stabilizable (i.e., if there exists a policy under which the average holding cost is finite). The ability to stabilize the system is another obvious benefit of using LEWC instead of strict priority policies, such as $c\mu$, which do not belong to the class of stabilizing policies (i.e., policies that always result in a finite cost if the underlying system is stabilizable).

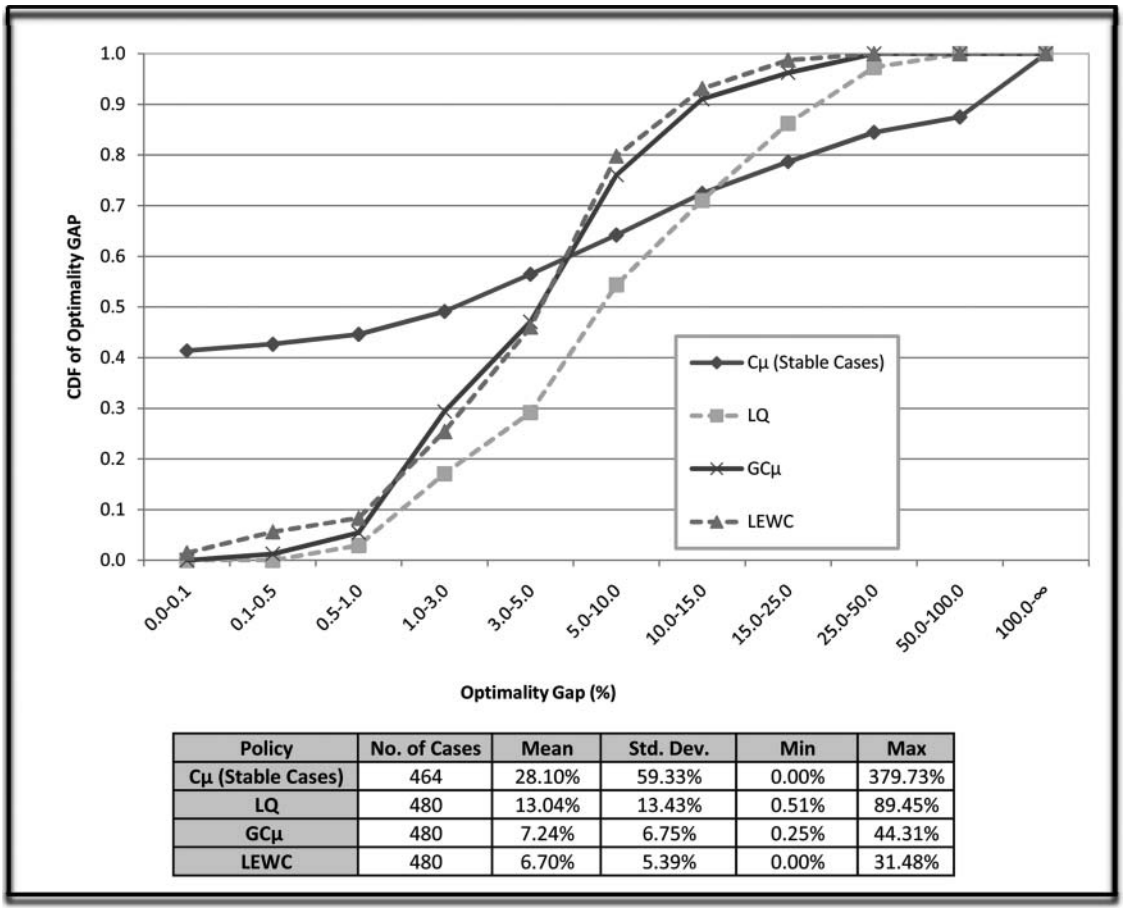
Theorem 4. *(*stability under LEWC*). If the condition of Theorem 1 or Corollary 1 is satisfied (and, hence, the system is stabilizable), then implementing the LEWC policy stabilizes the "W" system. That is, if $Z^* = \inf_{\pi\in\Pi} Z^\pi < \infty$ and Γ denotes the LEWC policy, then $Z^\Gamma < \infty$.*

The following theorem presents the same property for the LQ policy as well as the $Gc\mu$ rule with quadratic holding cost.

Theorem 5. *(*stability under LQ and $Gc\mu$*). Suppose the condition of Theorem 1 or Corollary 1 is satisfied (and, hence, the system is stabilizable). Then implementing either the LQ policy or the $Gc\mu$ rule with quadratic holding costs stabilizes the "W" system. That is, if $Z^* = \inf_{\pi\in\Pi} Z^\pi < \infty$, and ν denotes either of these policies policy, then $Z^\nu < \infty$.*

4.4. *Computational results*

This section compares the performance of our proposed heuristic with (i) the optimal policy; (ii) the widely used LQ policy; (3) the well-known $c\mu$ rule; and (4) the $Gc\mu$ rule for quadratic holding costs. Under LQ, each server prioritizes serving the queue (among its skill set) with the highest queue length. The $c\mu$ rule, as mentioned before, prescribes server j to serve the queue $k = \text{argmax}_i\, c_i\,\mu_{ji}$, where c_i is the holding cost of a customer in class i. Under the $Gc\mu$, the class to be served by server j is $k = \text{argmax}_i\,\mu_{ji}\,C'_i(x_i(t))$, where $x_i(t)$ is the queue length of class i at time t and $C'_i(\cdot)$ is the derivative of the holding cost function with respect to x_i. As is prevalent in the literature, we use this policy for the case of a quadratic holding cost ($C_i(x_i) = c_i\,x_i^2$). Thus, the implemented version of the $Gc\mu$ (also referred to as max-weight) prescribes server j to serve class $k = \text{argmax}_i\, c_i\,\mu_{ji}\,x_i$. When there is only one job in the shared queue and none in other queues, under all policies we assume that the server that is (among available servers) faster in serving the shared task serves the only job in the system.



Policy	No. of Cases	Mean	Std. Dev.	Min	Max
Cμ (Stable Cases)	464	28.10%	59.33%	0.00%	379.73%
LQ	480	13.04%	13.43%	0.51%	89.45%
GCμ	480	7.24%	6.75%	0.25%	44.31%
LEWC	480	6.70%	5.39%	0.00%	31.48%

Fig. 5. Performance of $c\mu$, LQ, $Gc\mu$ and LEWC relative to the optimal policy.

To perform the comparisons, we developed an extensive test suite of problem instances that covers various combinations of holding costs, disruption rates, service rates, arrival rates, workload distribution among the queues, and system congestions around 70% and 90% (which are common in small service centers and make-to-order manufacturing systems). Part II of Online Appendix C presents this test suite and the methods used to cover wide ranges of parameter combinations. This test suite generates 480 problem instances for the "W" network and builds a fairly large test suite (given the computational effort for these models).

To benchmark the "W," we employed the MDP of Section 3.2 to compute the optimal cost for each of our problem instances. A similar computational framework is used for *policy evaluation* to benchmark the performance of the LEWC, LQ, $c\mu$, and $Gc\mu$ policies. We used the value-iteration algorithm to solve MDPs numerically and we truncated the state space so that even for the cases with high utilization the probability of reaching the truncation limit was insignificant. Figure 5 summarizes our computational results over the test suite by depicting the empirical Cumulative Distribution Function (CDF) for the percentage optimality gap (i.e., the CDF of the percentage increase over the optimal cost) of each heuristic policy (i.e., LQ, $c\mu$, $Gc\mu$, and LEWC). Specifically, this figure summarizes the result of our $480 \times 5 = 2400$ MDP-based runs. Figure 5 also presents key statistics of the obtained optimality gaps: mean, standard deviation, minimum, and maximum.

Even though the system can be stabilized (by Corollary 1) for each problem instance, we observed that the greedy $c\mu$ policy is unstable in 16 out of 480 problem instances (i.e., 3.33% of cases) within the test suite. Hence, we considered the remaining 464 cases as the basis for computing the statistics on the $c\mu$ rule. However, as Theorems 4 and 5 indicate, LEWC, LQ, and $Gc\mu$ always stabilize the "W." Figure 5 illustrates that the proposed heuristic, LEWC, outperforms the other policies. The mean optimality gap for LEWC is 6.70% in contrast to 13.04% for LQ, 28.10% for $c\mu$ (among stable cases), and 7.24% for $Gc\mu$. That is, the mean optimality gaps of LQ, $c\mu$, and $Gc\mu$ are 195%, 420%, and 108% of that of LEWC, respectively. These results suggest that LEWC (as the first best) and $Gc\mu$ (as the second best) are *nearly optimal* policies considering that the problem instances include *wide variations* on disruption rates, repair rates, arrival rates, costs, traffics, etc. This observation is especially important in light of the following points.

1. The optimal policy is too complex for practical application in many settings.
2. Even for small systems with few servers and task types, obtaining the optimal policy becomes quickly intractable, especially when disruptions are allowed. Therefore, when the size of the systems increases, the optimality gap of a heuristic quickly becomes intractable, so comparisons to the performance of other available heuristics are appropriate.

The standard deviation column in Fig. 5 shows that LEWC is considerably more *robust* than other policies in the sense that it is more predictably effective over a wide range of model parameters. For any test case, the heuristics employ the true parameters. Thus, robustness for us is not associated with model uncertainty; rather, it is the range of parameters over which a policy is effective. Indeed, as the figure shows, the standard deviations of LQ, $c\mu$, and $Gc\mu$ are 249%, 1100%, and 125% of that of LEWC, respectively. From Fig. 5 we also observe that the CDF of the optimality gap of LEWC is closer to that of $Gc\mu$ compared to LQ and $c\mu$. However, LEWC outperforms all of the policies including $Gc\mu$ in all four metrics (mean, standard deviation, minimum, and maximum). Moreover, the obtained CDF for the optimality gap of LEWC is *always* above that of LQ, highlighting the clear advantage of using LEWC over LQ. However, the CDF for the optimality gap of $c\mu$ is *initially* above LEWC, because for about 40% of the test problems within our test suite the $c\mu$ rule obtains the optimal cost (since its optimality conditions presented in Theorem 3 are met). Of course, one can revise the LEWC policy so that it implements the $c\mu$ rule when its optimality conditions are met. We did not implement this obvious improvement, because the LEWC policy as stated can be applied in any general network structure for which the optimality conditions of $c\mu$ may not be known. This way, we gain more confidence that LEWC is suitable for a wide range of applications.

Although the CDF for the optimality gap of $c\mu$ is initially above LEWC, it should be noted that the $c\mu$ rule is a greedy policy and is very risky to implement unless the system's manager can ensure that its optimality conditions are not violated in advance. For instance, even under a heavy-traffic regime, Mandelbaum and Stolyar (2004) discussed that although the $Gc\mu$ rule is asymptotically optimal when holding costs are convex, its special case, $c\mu$, may not be optimal when the holding costs are linear. Our results for systems with moderate traffic and linear holding costs show that the $c\mu$ rule performs poorly on average. Moreover, as Fig. 5 shows, $c\mu$ is unstable in 3.33% of cases, and among the stable cases, $c\mu$ shows a large standard deviation of 59.33% (and a maximum of 379.73%) in its optimality gap. Our proposed algorithm, similar to $Gc\mu$, combines the cost minimization intuition behind the $c\mu$ and the load balancing idea of LQ. However, unlike $Gc\mu$, LEWC uses an LP (LP 2) to approximate the effort levels (y_{ji}). This use of an LP permits a more accurate estimatation of the workload and allows LEWC to dynamically balance the *workload costs*. This fact makes LEWC not only a more effective policy in terms of the mean optimality gap but also a considerably more *robust* policy with a relatively small standard deviation of 5.39%. This small standard deviation suggests another advantage of LEWC: using LEWC for comparing various queueing designs (where the optimal policy is computationally intractable) can be more reliable than implementing other policies (see, for instance, Iravani *et al.* (2005) and Iravani *et al.* (2007) where LQ is used for strategic design comparisons).

Another observation from Fig. 5 is that the widely used LQ and $Gc\mu$ policies are never optimal within our test suite, showing a minimum optimality gap of 0.51 and 0.25%, respectively. However, our proposed LEWC algorithm achieves the optimal cost in a few cases and, like $c\mu$, has a minimum gap of 0%. Moreover, LEWC, unlike $c\mu$ and LQ, rarely results in an optimality gap of above 15%. Indeed, under LEWC the chance of obtaining a performance that is 15% worse than optimal is only 6.9% (within our test suite), but under LQ and $c\mu$ the chances are 29.0% and 27.6%, respectively.

Another point of interest is to look at the performances of the policies in detail from the perspectives of disruption, congestion, and cost. Table 1 presents the detailed comparisons based on Settings I to IV (see Table 4.C in Online Appendix C). These four settings represent various combinations of disruption and system congestion. Setting I represents a system with no disruption and relatively high traffic. The scope of this research is not the heavy-traffic regime; therefore, here high traffic means relatively high congestion of around 90% and low represents 70%. In Setting II the servers are reliable, but the system congestion is relatively low. Settings III and IV represent scenarios where the system is under relatively lower traffic; in Setting III, servers are completely reliable, but they are under stochastic disruptions in Setting IV. The results in Table 1 suggest the following observations.

1. Interestingly, $c\mu$ outperforms LQ on average under relatively low traffic (see the mean optimality gaps under Settings II and IV). Under relatively higher traffic, however, LQ is better than $c\mu$. This observation may suggest that the load balancing of LQ becomes more important than the greedy cost minimization of $c\mu$ when traffic is moderate to relatively high.
2. LQ is always worse than $c\mu$ with respect to the minimum optimality gap criterion and always better with respect to the maximum optimality gap. This result is intuitive since $c\mu$, unlike LQ, is an extreme (and a greedy) policy. Additionally, LEWC is almost as good as $c\mu$ (which itself outperforms LQ) under the minimum optimality gap criterion and always better than LQ (which itself outperforms $c\mu$) under the maximum optimality gap criterion. Moreover, in all of these four settings, LEWC outperforms LQ, $c\mu$, and $Gc\mu$ with respect to the mean optimality gap and, therefore, presents the best policy

Table 1. Comparison of policies based on the combinations of disruption and the system congestion using the percentage optimality gaps

Setting	*Disruption*	*Traffic*	*Policy*	*Number of cases*	*Mean (%)*	*Min. (%)*	*Max. (%)*
(I)	No	High	$c\mu$ (stable cases)	112	69.52	0.00	379.73
			LQ	120	22.12	1.25	89.45
			$Gc\mu$	120	12.47	0.87	44.31
			LEWC	120	11.92	0.31	31.48
(II)	No	Low	$c\mu$ (stable cases)	120	7.01	0.00	36.75
			LQ	120	10.45	0.56	43.56
			$Gc\mu$	120	4.86	0.25	16.51
			LEWC	120	4.57	0.00	12.06
(III)	Yes	High	$c\mu$ (stable cases)	112	32.75	0.00	352.13
			LQ	120	12.47	1.23	62.44
			$Gc\mu$	120	7.77	0.71	29.12
			LEWC	120	7.06	0.01	16.19
(IV)	Yes	Low	$c\mu$ (stable cases)	120	6.36	0.00	36.95
			LQ	120	7.11	0.51	34.53
			$Gc\mu$	120	3.84	0.33	12.67
			LEWC	120	3.25	0.00	8.03

under various settings. This strength of LEWC derives from the way it accounts for different parameters of the system through the proposed LP 2 incorporated in the LEWC index.

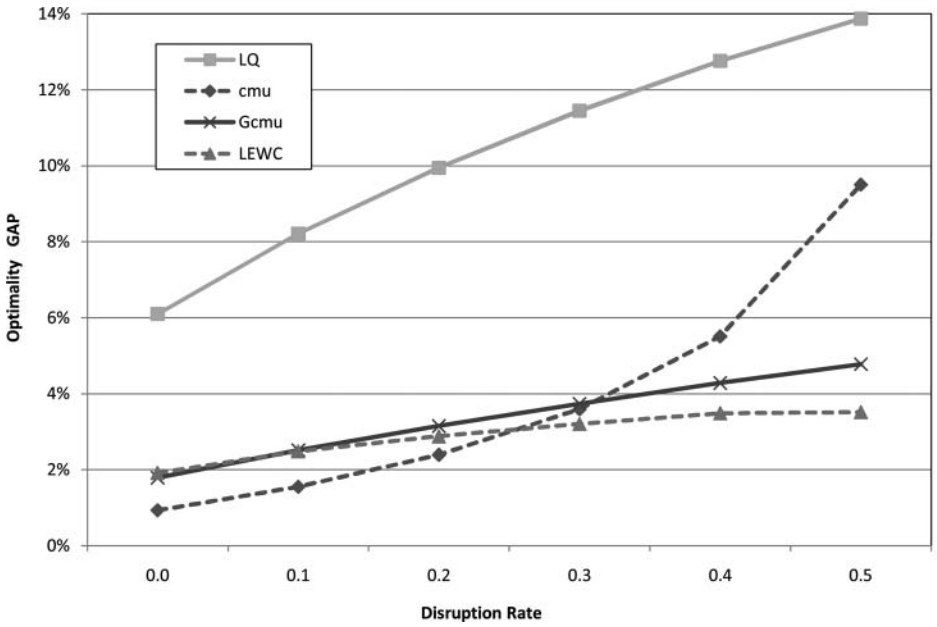
3. All of the policies show a smaller average optimality gap under lower congestion (compare Setting I with II, and Setting III with IV). This observation may suggest that it is better to implement these policies for systems with low to moderate congestion rather than systems with relatively high traffic.

Table 2 compares the policies based on the various holding cost settings defined in Table 5.C in Online Appendix C. In Setting A, all holding costs equal one, representing a symmetric situation. Settings B to D represent situations with asymmetric holding costs among customer classes where the degree of asymmetry develops from a low degree in B to a high degree in D. A closer look at Table 2 provides the following observations.

1. All of the policies perform their best (based on the mean criterion) when there is no cost asymmetry. Moreover, the performance of both LQ and LEWC deteriorates as the level of asymmetry increases. However, this deterioration does not occur for $c\mu$ or $Gc\mu$. In fact, both $c\mu$ and $Gc\mu$ perform their worst when the level of asymmetry in holding costs is moderate (Setting C). Although the performance of LEWC, unlike $c\mu$ and $Gc\mu$, deteriorates as the level of asymmetry increases, as Table 1 show, LEWC still outperforms both $c\mu$ and $Gc\mu$.
2. The proposed heuristic (LEWC) is much more robust to changes in holding costs than other policies. For instance, the mean optimality gap of LQ changes from 5.03% to 20.67% (a more than 410% change) by moving from no asymmetry in costs to high asymmetry in costs whereas the mean optimality gap of LEWC only changes from 4.41% to 8.43% (less than a 183% change). These results show that the performance of the widely used $c\mu$ and LQ policies, unlike LEWC and $Gc\mu$, is very sensitive to the holding costs. This observation is intuitive, because LQ does not consider holding costs and $c\mu$ depends on them in a relatively extreme way.
3. To complete the previous observation, we should note that LEWC, similar to $c\mu$ and $Gc\mu$, uses holding costs to determine the switching curves, but LEWC, unlike

Table 2. Comparison of policies based on the holding cost settings (level of cost asymmetry among different classes: (A) zero, (B) low, (C) moderate, (D) high

	$c\mu$ (stable cases)			*LQ*			*$Gc\mu$*			*LEWC*		
	Optimality gap (%)			*Optimality gap (%)*			*Optimality gap (%)*			*Optimality gap (%)*		
Setting	*Mean*	*Min.*	*Max.*	*Mean*	*Min.*	*Max.*	*Mean*	*Min.*	*Max.*	*Mean*	*Min.*	*Max.*
(A)	0.00	0.00	0.00	5.03	1.08	11.39	4.28	0.64	10.99	4.41	0.00	14.52
(B)	29.51	0.00	207.18	8.34	0.56	31.30	7.63	0.44	33.19	5.67	0.14	23.81
(C)	40.75	0.00	379.73	12.77	0.51	51.10	8.69	0.61	44.31	7.17	0.11	29.10
(D)	23.33	0.00	182.31	20.67	1.24	89.45	6.37	0.25	23.39	8.03	0.04	31.48
Total	28.10	0.00	379.73	13.04	0.51	89.45	7.24	0.25	44.31	6.70	0.00	31.48



		Opt. Gap	
Policy	No. of Cases	Mean	Std. Dev.
Cμ	288	3.93%	12.00%
LQ	288	10.29%	7.69%
GCμ	288	3.34%	2.35%
LEWC	288	2.92%	2.10%

Fig. 6. Sensitivity of $c\mu$, LQ, $Gc\mu$, and LEWC to variations in disruption risks.

$c\mu$ and $Gc\mu$, uses holding cost together with other system parameters. This fact makes LEWC less sensitive to changes in the system parameters (including holding costs and service rates). Therefore, our results recommend the use of LEWC rather than $c\mu$ and $Gc\mu$ when the system parameters vary over time. A similar comparison indicates that LEWC is also preferable to LQ. However, it should be noted that LQ is the rational choice in the absence of any information on the system parameters (which is perhaps its best feature).

Finally, to explore the effect of server disruptions on the performance of LEWC, LQ, $c\mu$, and $Gc\mu$, we consider the test suite presented in Tables 6.C, 7.C, and 8.C (Online Appendix C) and depict in Fig. 6 the average optimality gap (over all problem instances) of each of these policies for different levels of disruptions. As the results confirm, LEWC is the least sensitive policy to variations in the disruption risks as it explicitly incorporates disruption rates. This robustness to disruptions provides another benefit of the proposed policy, LEWC. Among other policies, $Gc\mu$ and LQ are more robust to disruptions compared to $c\mu$, since they implicitly incorporate the effect of disruptions through a consideration of queue lengths.

5. Conclusions

This article considered the problem of assigning servers to various jobs in real time to obtain good performance and thereby extract the most benefit from the flexibility of the servers. We first developed an MDP modeling framework for parallel queueing systems with arbitrary number of job types, arbitrary number of servers, arbitrary flexibility structures, and heterogeneous servers subject to stochastic disruptions. We implemented an LP to investigate the stability of such a queueing system, provided that some convergence and monotonicity results, and showed that it is sufficient to restrict attention to the class of non-idling stationary policies.

To gain more insights into the characteristics of effective real-time server assignment mechanisms, we then considered the "W" design in which servers are trained to work on a shared task in addition to their fixed task. As a three-class network with two partially flexible servers, the "W" generalizes the "N" structure, which has received considerable attention (mainly due to the intrinsic difficulties of the underlying control problem). Next, comparing the "W" design with other possible structures, we showed that the "W" queueing network is an efficient paradigm; with a little investment in flexibility, it provides a performance almost as good as a fully flexible network. This article provided specific observations into how the "W" design is the preferred structure for many systems with three demand types and two servers. Given the obstacles of fully cross-training servers in practice, our models, analyses, and numerical studies provide designers with a better understanding of *how to effectively introduce limited flexibility to a system.*

We next provided insights for a system manager on *how to benefit from the limited flexibility of servers in real time*. We showed that, for the "W," a version of the greedy $c\mu$ policy that prescribes every server (whenever not disrupted) to work on *the fixed task before the shared task* is optimal under some conditions. In general, we observed that the optimal policy is of a state-dependent threshold type and can be characterized by four threshold surfaces. However,

our findings confirmed that the optimal policy is complex in general, which makes it less attractive for use in practice.

Therefore, we introduced a new, powerful, and implementable policy to control the servers. Squillante *et al.* (2001, p. 2992) states that

> A fundamental understanding of the scheduling tradeoff [between achieving server load balancing and scheduling jobs where they are processed most efficiently] is of great theoretical interest.

Our heuristic policy, LEWC, considers this trade-off and balances the *workload cost* of queues (using the traditional notion of instantaneous workload). This balance is achieved by combining the *load balancing* intuition that is effectively captured in the *queue length dependence* of the LQ policy and the greedy *cost rate minimization* concept embodied in the $c\mu$ rule. LEWC dynamically measures the expected workload costs of the queues and then assigns each server to the queue with LEWC among its skill set. We first established the stability of LEWC (as well as LQ and $Gc\mu$) and then performed an extensive MDP-based numerical test to gain insights into the performance of LEWC as well as three widely used policies: the LQ policy and the $c\mu$ and the $Gc\mu$ rules. Our results particularly suggested the following two conclusions: (i) LEWC is a *near-optimal* policy outperforming the other policies; and (ii) LEWC is more *robust* than LQ, $c\mu$, and $Gc\mu$ over a wide range of operating environments (holding costs, service rates, disruptions, etc.). This latter observation is an important property in practice, since system parameters often vary over time.

This robustness may also suggest that the proposed LEWC heuristic is a more reliable mechanism than the other policies for applications to strategic design, where a designer needs a fair control policy to compare the performance of alternate designs. However, future work could explore if LEWC is also robust across different cross-training designs. If so, it would also be a useful policy for the strategic design of general flexible/queueing systems. This can greatly benefit research targeting strategic design of queueing systems in the vein of Iravani *et al.* (2005) and Iravani *et al.* (2007).

Acknowledgement

The work of the first two authors was partially supported by NSF grant DMI-0542063.

References

Aksin, O.Z. and Karaesmen, F. (2002) Designing flexibility: characterizing the value of cross-training practices. Working paper, INSEAD, Fontainebleau Cedex, France.

Andradóttir, S., Ayhan, H. and Down, D.G. (2001) Server assignment policies for maximizing the steady-state throughput of finite queueing systems. *Management Science*, **47**(10), 1421–1439.

Andradóttir, S., Ayhan, H. and Down, D.G. (2003) Dynamic server allocation for queueing networks with flexible servers. *Operations Research,* **51**(6) 952–968.

Andradottir, S., Ayhan, H. and Down, D.G. (2007) Compensating for failures with flexible servers. *Operations Research*, **55**(4), 753–768.

Andradottir, S., Ayhan, H. and Down, D.G. (2010) Design principles for fexible systems. Working paper. School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, CA.

Armony, M. and Bambos, N. (2003) Queueing dynamics and maximal throughput scheduling in switched processing systems. *Queueing Systems: Theory and Application*, **44**(3), 209–252.

Bassamboo, A., Randhawa, R.S. and Van Mieghem, J.A. (2009) A little flexibility is all you need: asymptotic optimality of tailored chaining and pairing in queuing systems. Working paper, Kellog School of Business, Northwestern University, Evanston, IL.

Bell, S.L. and Williams, R.J. (2001) Dynamic scheduling of a system with two parallel servers in heavy traffic with resource pooling: asymptotic optimality of a threshold policy. *Annals of Applied Probabilty,* **11**(3), 608–649.

Bell, S.L. and Williams, R.J. (2005) Dynamic scheduling of a parallel server system in heavy traffic with complete resource pooling: asymptotic optimality of a threshold policy. *Electronic Journal on Probability*, **10,** 1044–1115.

Bramson, M. and Williams, R.J. (2000) On dynamic scheduling of stochastic networks in heavy traffic and some new results for the workload process, in *Proceedings of the 39th IEEE Conference on Decision and Control*, IEEE, Piscataway, NJ, pp. 516–521.

Buyukkoc, C., Varaiya, P. and Walrand, J. (1985) The $c\mu$ rule revisited. *Advances in Applied Probability*, **17**, 237–238.

Chou, M.C., Chua, G.A., Zheng, H. and Teo, C.-P. (2010) Design for process flexibility: efficiency of the long chain and sparse structure. *Operations Research,* **58**(1), 43–58.

Dai, J.G. (1999) *Stability of Fluid and Stochastic Processing Networks*, Publication 9, Centre for Mathematical Physics and Stochastics. Department of Mathematical Sciences, University of Aarhus, Denmark.

Dai, J.G. and Lin, W. (2005) Maximum pressure policies in stochastic processing networks. *Operations Research*, **53**(2), 197–218.

Down, D.G. and Lewis, M.E. (2010) The *N*-network model with upgrades. *Probability in the Engineering and Informational Sciences*, **24**, 171–200.

Harrison, J.M. (1998) Heavy traffic analysis of a system with parallel servers: asymptotic analysis of discrete-review policies. *Annals of Applied Probability*, **8**(3), 822–848.

Harrison, J.M. and López, M.J. (1999) Heavy traffic resource pooling in parallel-server systems. *Queueing Systems*, **33**, 339–368.

Hopp, W.J., Tekin, E. and Van Oyen, M.P. (2004) Benefits of skill chaining in production lines with cross-trained workers. *Management Science*, **50**(1), 83–98.

Hopp, W.J. and Van Oyen, M.P. (2004) Agile workforce evaluation: a framework for cross-training and coordination. *IIE Transactions*, **36**(10), 919–940.

Iravani, S.M., Kolfal, B. and Van Oyen, M.P. (2011) Capability flexibility: a decision support methodology for parallel service and manufacturing systems with flexible servers. *IIE Transactions*, **43**(5), 363–382.

Iravani, S.M.R., Van Oyen, M.P. and Sims, K.T. (2005) Structural flexibility: a new perspective on the design of manufacturing and service operations. *Management Science*, **51**(2), 151–166.

Iravani, S.M.R., Kolfal, B. and Van Oyen, M.P. (2007) Call center labor cross-training: it's a small world after all. *Management Science*, **53**(7), 1102–1112.

Jordan, W.J. and Graves, S.C. (1995) Principles on the benefits of manufacturing process flexibility. *Management Science*, **41**(4), 577–594.

Lippman, S. (1975) Applying a new device in the optimization of exponential queueing system. *Operations Research*, **23**(4), 687–710.

Mandelbaum, A. and Stolyar, A.L. (2004) Scheduling flexible servers with convex delay costs: heavy-traffic optimality of the generalaized $c\mu$-rule. *Operations Research*, **52**(6), 836–855.

Meyn, S.P. (2003) Sequencing and routing in multiclass queueing networks part II: workload relaxations. *SIAM Journal on Control and Optimization*, **42**(1), 178–217.

Saghafian, S. and Van Oyen, M.P. (2011) The value of flexible backup suppliers and disruption risk information: newsvendor analyses with recourse. Working paper, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI.

Squillante, M.S., Xia, C.H. Yao, D.D. and Zhang, L. (2001) Threshold-based priority policies for parallel-server systems with affinity scheduling, in *Proceeding of the 2001 American Control Conference*, pp. 2992–2999.

Van Mieghem, J.A. (1995) Dynamic scheduling with convex delay costs: the generalized $c\mu$ rule. *Annals of Applied Probability*, **5**(3), 809–833.

Veatch, M.H. (2010) A $c\mu$ rule for parallel servers with two tiered $c\mu$ preference. Working paper, Mathematics Department, Gordon College, Wenham, MA.

Walrand, J. (1988) *An Introduction to Queueing Networks*, Prentice-Hall, Englewood Cliffs, NJ.

Biographies

Soroush Saghafian is currently a Ph.D. candidate in Industrial and Operations Engineering (IOE) at the University of Michigan. His research focus is on the application and development of operations research methods in modeling and control of stochastic systems with specific applications in (i) control of flexible queuing systems; (ii) healthcare operations; and (iii) supply chain and operations management. He has been awarded the 2010 INFORMS Pierskalla Award for the best research paper in Healthcare (from the Healthcare Applications Section of INFORMS), the 2010 Murty Prize for best research paper in Optimization, and the 2007 IOE Bonder Fellowship award for applied Operations Research. He has also been a finalist for the best student paper award of the Production and Operations Management Society (POMS) in 2009 (Supply Chain) and 2011 (Healthcare). At the University of Michigan, he taught IOE 440 (Operations Analyses and Management) as the primary instructor in both 2009 and 2010. Prior to joining the University of Michigan, he taught courses in Applied Probability Theory and Plant Layout as a primary instructor. He has served as a referee for various journals including *Operations Research*, *Operations Research Letters*, *Naval Research Logistics*, *IIE Transactions*, *IEEE Transactions on Evolutionary Computation*, and *Production and Operations Management*.

Mark P. Van Oyen has served as an Associate Professor of Industrial and Operations Engineering (IOE) at the University of Michigan since 2005. His core interests focus on the analysis, design, control, and management of operations systems, with emphasis on healthcare, service operations, and supply chains and how they can be designed for greater performance, flexibility, and resilience. His research also contributes to applied probability and the control and performance analysis of queuing networks. His awards include IOE Department Faculty of the Year for 2008–2009, ALCOA Manufacturing Systems Faculty Fellow, a best paper award from *IIE Transactions*, and Researcher of the Year from Loyola University Chicago's School of Business. He has served as Associate Editor for *Operations Research*, *Naval Research Logistics*, and *IIE Transactions* and Senior Editor for *Flexible Services & Manufacturing*. He was a faculty member of the Northwestern University School of Engineering (1993–2005) and Loyola Univ. of Chicago's School of Business Administration (1999–2005). He has received grant funding from the National Science Foundation (NSF), Office of Naval Research (ONR), EPRI, ALCOA, General Motors, and the VA. In industry, he was a researcher with GE Corporate R&D as well as an analysis and simulation engineer with Lear Siegler's Instrument & Avionic Systems Division.

Bora Kolfal is an Assistant Professor of Management Sciences at the School of Business, University of Alberta. His research interests are centered around design and control of service operations, manufacturing systems, and supply chains, with emphasis on improving their flexibility and performance. He mainly focuses on analysis of queueing networks, healthcare applications of OR, and game theory.