Queues with Skill Based Parallel Servers and a FCFS Infinite Matching Model
Ivo Adan, Marko Boon, Ana Bušić, Jean Mairesse, Gideon Weiss

To cite this version:
Ivo Adan, Marko Boon, Ana Bušić, Jean Mairesse, Gideon Weiss. Queues with Skill Based Parallel Servers and a FCFS Infinite Matching Model. MAMA 2013 workshop of ACM Sigmetrics, 2013, Pittsburgh, United States. pp.22-24, 10.1145/2567529.2567536. hal-01273894

HAL Id: hal-01273894
https://inria.hal.science/hal-01273894
Submitted on 14 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Queues with Skill Based Parallel Servers and a FCFS Infinite Matching Model

Ivo Adan † Marko Boon † Ana Busic ‡ Jean Mairesse § Gideon Weiss †

1. A QUEUEING SYSTEM

We consider the following skill based parallel service queueing system: Customers are of types \( C = \{c_1, \ldots, c_I\} \), servers are of types \( S = \{s_1, \ldots, s_j\} \), and there is a bipartite graph \( G \) of compatibilities between \( C, S \). The graph has an arc \( (i,j) \in G \) if server type \( s_j \) has the skill to serve customer type \( c_i \). Customers arrive in independent Poisson streams of rates \( \lambda_i \), and have absolutely continuous patience distributions \( F_i \). There are \( n_j \) servers of type \( s_j \), and the service times are customer-server-type dependent, the service of a customer of type \( c_i \) by a server of type \( s_j \) has a random duration distributed as \( G_{ij} \), with average \( m_{ij} \). We use the terminology of queuing theory throughout, but this type of system, with minor modifications, is useful in modeling call centers, manufacturing systems, organ transplants, multimedia servers, and cloud computing [8].

Performance of such systems is highly dependent on the operating policy. We focus here on first come first served (FCFS), where a server is assigned to the longest waiting compatible customer, coupled with assign longest idle server (ALIS), where a customer is assigned to the compatible server that has been idle for the longest time. FCFS-ALIS is widely used, because it is fair to both customers and servers, it is simple to implement, it requires little information about the parameters and the current state of the system, and it is robust under time varying conditions.

Our goal here is to develop a structured method to support the design and efficient operation of skill based parallel service systems under FCFS. At this level of generality such systems are highly intractable, no analytic results are expected, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scaling are considered, and asymptotics, e.g. using many server scal...
Let $M$ be the location of the first unmatched customer, and $\overline{M}$ be the location of the last matched customer, which was exchanged by a server. Consider the ordered sequence of servers and customers, $\gamma = \gamma_1, \gamma_2, \ldots, \gamma_{\overline{M}}$ with $\gamma_0 = 0$ if $N = \overline{M} = M - 1$ (see Figure 1 where $N = 4, M = 3, \overline{M} = 6$ and $\gamma = (c^3, s^4, c^5, s^6)$). We define a Markov chain $Z_N$ with state $\gamma$, where $Z_{N+1}$ will be obtained when $s^{N+1}$ is matched and exchanged.

$$C \xrightarrow{c} \{\emptyset, C\} \xrightarrow{c} \{\emptyset, C\} \xrightarrow{c} \{\emptyset, C\} \xrightarrow{c} \{\emptyset, C\} \xrightarrow{c} \{\emptyset, C\} \xrightarrow{c} \{\emptyset, C\} \xrightarrow{c} \{\emptyset, C\}$$

$$S \xrightarrow{s} \{\emptyset, S\} \xrightarrow{s} \{\emptyset, S\} \xrightarrow{s} \{\emptyset, S\} \xrightarrow{s} \{\emptyset, S\} \xrightarrow{s} \{\emptyset, S\} \xrightarrow{s} \{\emptyset, S\} \xrightarrow{s} \{\emptyset, S\}$$

Figure 1: FCFS matching with exchange transformation, $Z_4 = 3 = (c^3, s^4, c^5, s^6)$

**Theorem 1.** The stationary distribution of $Z_N$ is given by

$$\pi(\gamma) = B \prod_{i=1}^J \alpha_i^{#c_i} \beta_j^{#s_j}$$

(1)

where $#c_i$ is the number of customers of type $c_i$, and $#s_j$ is the number of servers of type $s_j$ in $\gamma$, and $B$ is the normalizing constant.

A necessary and sufficient condition for finite $B$ and ergodicity is complete resource pooling defined by:

$$\alpha_C < \beta_S(\mathcal{C}), \text{ for all non trivial subsets } C \subset \mathcal{C}. \quad (2)$$

We note that the clean and beautiful simplicity of this multi-Bernoulli product form of $\pi$ masks the true complexity of the process, which is revealed if we examine what states are allowed: we note that each $c^m$ in the sequence $\gamma$ must be incompatible with all later $s^n$. The proof of Theorem 1 uses time reversal, and the symmetry in the process. It leads us to make the following conjecture:

**Conjecture 1 (Reversibility).** Let $s^n, c^m, \ m, n \in Z$, be two independent i.i.d. sequences, with FCFS matching between them. Let $s^n, c^m$ be obtained by the exchange transformation, with the corresponding matching. Then $s^n, c^m$ are independent i.i.d. sequences, and the matching between them is FCFS in the reversed time direction.

We can in fact show that $s^n, c^m$ are FCFS in the reversed time direction, and that both of these sequences is i.i.d. but, we are still missing the proof that the sequence $s^n$ is independent of $c^m$.

From $\pi$ one can obtain various marginal distributions of interest, and various performance measures. Most relevant for our queueing system are the matching rates $r_{c_i, s_j}$ at which customers of type $c_i$ are served by servers of type $s_j$.

The formula for that is:

$$r_{c_i, s_j} = \beta_j \sum_{\mathcal{P}_j} B \prod_{k=1}^{J-1} \left( \frac{\phi_k}{\beta_k(\gamma_k - \alpha_k(\gamma_k))} \left( \sum_{k=1}^{J-1} \frac{\phi_k}{\beta_k(\gamma_k - \alpha_k(\gamma_k))} \right) \right)$$

$$\times \left( \prod_{l=1}^{J-1} \frac{\beta_l(\gamma_l - \alpha_l(\gamma_l))}{\beta_l(\gamma_l - \alpha_l(\gamma_l)))} + \phi J \sum_{l=1}^{J-1} \frac{\beta_l(\gamma_l - \alpha_l(\gamma_l))}{\beta_l(\gamma_l - \alpha_l(\gamma_l)))} \right)$$

(3)

where the summation is over $\mathcal{P}_j$, the set all the permutations of $\mathcal{S}$. For more details on the notation used see [1].

If complete resource pooling does not hold the system decomposes uniquely to subsystems, $(C^{(1)}, S^{(1)}), \ldots, (C^{(k)}, S^{(k)})$ defined recursively, the fist of them being:

$$C_1 = \arg \min_{C \subseteq \mathcal{C}} \frac{\beta_C}{\alpha_C}, \quad S_1 = S(C_1)$$

**3. A DESIGN ALGORITHM**

We now present an algorithm for the design of the queueing system of Section 1. We assume that we want to operate it in ED (efficiency driven) mode, with overloaded servers, and controlled abandonments. We let:

**Input:** The compatibility graph $G$, the arrival rates $\lambda_i$, the patience distributions $F_i$, and the service-time distributions $G_{ij}$, of which we only need the means $m_{ij}$.

**Quality of service decision:** Partition customer types into $C^{(l)}$, $l = 1, \ldots, L$, where higher $l$ implies more preferential service. Let $S^{(l)} = S(C^{(l)}) \setminus \bigcup_{l < k} S^{(k)}$. Specify target waiting times $W_1 > W_2 > \cdots > W_L$.

Continue to design each subsystem separately. We drop the superscripts and use notation $c_l \in C$, $s_j \in S$, $W_i$ within each subsystem:

**Division of labor decision:** Specify the fraction of services performed by each type of server, $\beta_j$.

**Calculations:** Define total service rate $\hat{\lambda} = \sum_{i} \lambda_i (1 - F_i(W))$ and let $\alpha_l = \lambda_l (1 - F_l(W))/\hat{\lambda}$. Verify complete resource pooling within the subsystem, and compute the matching rates $r_{c_i, s_j}$ using (3). Calculate required staffing:

$$n_j = \lambda \sum_{c_i \in C(s_j)} r_{c_i, s_j} m_{ij} \quad (4)$$

**Output:** Number of servers of each type, $n_j$, the matching rates $r_{ij}$, the cut-off waiting times $W_i$, and the abandonment rates $F_i(W_i)$.

The idea behind the algorithm is that if $\lambda$ is large relative to the typical waiting time $W$, there will be many customers in the system at all times, with a fraction $F(W)$ of abandonments, and the types of successive customers which will receive service will still be i.i.d $\alpha$. At the same time, with large $\lambda$ the number of servers $n_j$ will be large, and the instants of service completions will be a superposition of many almost stationary, almost independent, point processes which should be approximately Poisson. As a result types of successive available servers will be i.i.d. $\beta$.

We now test our algorithm on simple graph $G$, to show that it performs quite well with moderate number of servers, $C, S$ and $G$ are shown in Figure 2. We use the following input data. The proportion of arrivals for the $c_i$ is $\alpha^i = (0.2, 0.5, 0.3)$. The patience distribution for all customers is exponential with mean $10$, so $F_i(t) = 1 - e^{-\beta^i t}$. The service time distributions $G_{ij}$, are as follows:

<table>
<thead>
<tr>
<th>Service time distributions : Means</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$\exp(0.2)$</td>
<td>$\exp(0.2)$</td>
<td>$\exp(0.2)$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$\text{pareto}(3, 3):4.5$</td>
<td>$\text{pareto}(2, 3):3$</td>
<td>$\exp(0.125):8$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$\text{uniform}(2, 6):4$</td>
<td>$\text{uniform}(1, 5):3$</td>
<td>$\text{uniform}(1, 5):3$</td>
</tr>
</tbody>
</table>
We explore two designs. The first provides equal service quality to all types of customers, the second design gives preferential service to types $c_1, c_3$ over type $c_2$. We ran extensive simulations, of 100 runs with 100,000 customers each, for every experiment. Some of our results are:

**First design: equal service quality**

We choose common waiting time $W = 1$, so that abandonment rate is targeted as $\theta_i = F_i(W_i) = 1 - e^{-\alpha_i} \approx 0.095$. This leaves $\alpha = \alpha^{(1)} = (0.2, 0.5, 0.3)$. We choose $\beta = (0.3, 0.4, 0.3)$. This satisfies complete resource pooling. The calculated matching rates are

$$[r_{ij}] = \begin{bmatrix}
0 & 176 & 47 \\
72 & 0 & 218 \\
178 & 223 & 0
\end{bmatrix},$$

and the required staffing, as a function of the total arrival rate $\lambda$ works out at:

$$n(\lambda) \approx \lambda(1.305, 1.300, 1.981).$$

We use several values of $\lambda$. As expected the results improve as $\lambda$ increases, but are already quite good for a moderate number of servers.

**Second design: preferential service**

Here we use a waiting time of $W_1 = 2$, for $c^{(1)} = \{c_2\}$, and waiting time $W_2 = 0.5$ for $S^{(2)} = \{c_1, c_3\}$. This partitions the servers into $S^{(1)} = \{s_1, s_3\}$, and $S^{(2)} = \{s_2\}$ For $S^{(1)}$ we choose $\beta_1 = \beta_3 = 0.5$, while of course for $S^{(2)} \beta_2 = 1$. The staffing calculations lead to

$$n(\lambda) \approx \lambda(1.023, 1.712, 1.637).$$

Results of the simulation are given in the following tables.

### Tables

<p>| Table 1: Simulation results for equal service level. |</p>
<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>$n$</th>
<th>$E[W_1]$</th>
<th>$E[W_2]$</th>
<th>$E[W_3]$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
<th>$\theta_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(13, 13, 20)</td>
<td>0.870</td>
<td>1.226</td>
<td>0.884</td>
<td>.086</td>
<td>.119</td>
<td>.088</td>
</tr>
<tr>
<td>30</td>
<td>(39, 39, 59)</td>
<td>0.924</td>
<td>1.110</td>
<td>0.928</td>
<td>.089</td>
<td>.105</td>
<td>.089</td>
</tr>
<tr>
<td>50</td>
<td>(65, 65, 99)</td>
<td>0.929</td>
<td>1.053</td>
<td>0.932</td>
<td>.089</td>
<td>.100</td>
<td>.089</td>
</tr>
<tr>
<td>100</td>
<td>(130, 130, 198)</td>
<td>0.956</td>
<td>1.025</td>
<td>0.958</td>
<td>.090</td>
<td>.096</td>
<td>.090</td>
</tr>
</tbody>
</table>

<p>| Table 2: Simulation results for preferential treatment. |</p>
<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>$n$</th>
<th>$E[W_1]$</th>
<th>$E[W_2]$</th>
<th>$E[W_3]$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
<th>$\theta_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(10, 17, 16)</td>
<td>0.685</td>
<td>2.404</td>
<td>0.687</td>
<td>.068</td>
<td>.221</td>
<td>.069</td>
</tr>
<tr>
<td>30</td>
<td>(31, 51, 49)</td>
<td>0.596</td>
<td>2.011</td>
<td>0.596</td>
<td>.059</td>
<td>.184</td>
<td>.059</td>
</tr>
<tr>
<td>50</td>
<td>(51, 86, 82)</td>
<td>0.504</td>
<td>2.017</td>
<td>0.505</td>
<td>.049</td>
<td>.183</td>
<td>.050</td>
</tr>
<tr>
<td>100</td>
<td>(102, 171, 164)</td>
<td>0.525</td>
<td>2.010</td>
<td>0.526</td>
<td>.051</td>
<td>.180</td>
<td>.051</td>
</tr>
<tr>
<td>Theoretical</td>
<td></td>
<td>0.500</td>
<td>2.000</td>
<td>0.500</td>
<td>.050</td>
<td>.180</td>
<td>.050</td>
</tr>
</tbody>
</table>

Figure 2: The “almost complete” network. Nodes on the top row represent customer types, and the ones on the bottom row server types. The dashed arcs are present in the case of complete resource pooling, but vanish in the case of incomplete resource pooling.