Performance Evaluation 116 (2017) 70–83

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

# Balanced fair resource sharing in computer clusters

Thomas Bonald [a], Céline Comte [b,a,*,1]

[a] Télécom ParisTech, Université Paris-Saclay, France
[b] Nokia Bell Labs, Nozay, France

ARTICLE INFO

*Article history:*
Available online 1 September 2017

*Keywords:*
Parallel processing
Multi-server queues
Balanced fairness
Order independent queues
Whittle networks

ABSTRACT

We represent a computer cluster as a multi-server queue with some arbitrary graph of compatibilities between jobs and servers. Each server processes its jobs sequentially in FCFS order. The service rate of a job at any given time is the sum of the service rates of all servers processing this job. We show that the corresponding queue is quasi-reversible and use this property to design a scheduling algorithm achieving balanced fair sharing of the computing resources.



## 1. Introduction

Load balancing is a critical component of large-scale computer clusters. The flow of requests must be directed to the servers under various constraints like data availability, state of the servers and service level agreements. In this paper, we represent these constraints by an arbitrary graph of compatibilities between jobs and servers. The computer cluster can then be viewed as a multi-server queue where jobs are allocated to servers according to this graph. We assume that each server processes its jobs sequentially in FCFS order. The service rate of a job at any given time is the sum of the service rates of all servers processing this job, which means that resource pooling does not induce any processing overhead. We prove that, for Poisson job arrivals and exponential job sizes, this multi-server queue is quasi-reversible [1]. Exploiting this property, we design a novel scheduling algorithm achieving *balanced fair* sharing of the computing resources. This makes the stationary distribution of the system state insensitive to the job size distribution beyond the mean, a practically interesting property leading to simple and robust engineering rules.

Balanced fairness was introduced in the context of data networks as the most efficient resource allocation having the insensitivity property, allowing the service provider to develop dimensioning rules based on average traffic only, and not on detailed traffic characteristics [2]. Formally, it is the only allocation such that the underlying Markov process is reversible and at least one resource is saturated in each state. Balanced fairness has later been used to evaluate the performance of content-distribution networks [3,4]. However, no scheduling algorithm has been proved so far to achieve this allocation, except in some specific cases where it coincides with proportional fairness [5,6]. To the best of our knowledge, our scheduling algorithm is the first practical implementation of balanced fairness, just like the round-robin scheduling algorithm is a well-known practical implementation of the ideal processor-sharing (PS) service discipline.

Multi-server queues with specialized servers have already been considered in [7–10] but these models assume that each job can be processed by only one server at a time. Our model is closer to the multi-server queue with redundant requests introduced by Gardner et al. [11,12], where the class of a job defines the set of servers on which it is replicated. When several replicas of the same job are in service simultaneously on different servers, their service times are independent and the first

* Corresponding author.
*E-mail addresses:* thomas.bonald@telecom-paristech.fr (T. Bonald), celine.comte@nokia.com (C. Comte).
[1] The authors are members of LINCS, see http://www.lincs.fr.

http://dx.doi.org/10.1016/j.peva.2017.08.006
0166-5316/

instance to be completed stops the others. It is easy to see that, under the assumption of exponential service times, the two models are in fact equivalent. In both cases, the FCFS policy makes the system very sensitive to the job size distribution, so that the actual performance may vary significantly when the job sizes are not exponentially distributed with the same unit mean. Our objective in this paper is precisely to relax this assumption by designing a scheduling policy which makes the system insensitive to the job size distribution.

It turns out that our model belongs to the family of Order Independent (OI) queues [13,14]. As observed in [14], OI queues generalize a number of queueing systems like BCMP networks under the FCFS or PS service discipline [15], multiserver stations with concurrent classes of customers (MSCCC) and multiserver stations with hierarchical concurrency constraints (MSHCC) [16,17]. OI queues are known to be quasi-reversible [1]. In particular, the state of the queue has an explicit stationary distribution under the usual assumptions of Poisson arrivals and exponential service times. Moreover, the stationary distribution remains explicit in the presence of random routing, where jobs can leave or re-enter the queue upon service completion.

The first contribution of this paper is a scheduling algorithm which exploits this last property to mitigate the sensitivity to the job size distribution. Just like round-robin scheduling which implements the PS service discipline in the single-server case, our mechanism enforces insensitivity by interrupting the jobs frequently and moving them to the end of the queue. Routing is thus reinterpreted in terms of job interruptions and resumptions. The queue state is updated in the course of the job shiftings and the exponentially distributed sizes with unit mean in the multi-server queue now represent small fragments of the jobs. When the interruptions are frequent, each job tends to go back and forth in the queue and its average service rate is mainly determined by the number of jobs of each class which are present at the same time.

This last observation motivates us to adopt a higher viewpoint. Specifically, we aggregate the state of the multi-server queue to only retain the number of jobs of each class, but not their arrival order. This aggregate state turns out to be an appropriate level of granularity to analyze the behavior of the queue. Its stationary measure is exactly that of a Whittle network [18] containing as many PS queues as there are classes in the original multi-server queue. This leads us to our second contribution: a new theoretical understanding of the multi-server queue. Using the state aggregation, we show in Theorem 1 that the queue is stable under any vector of acceptable arrival rates. In practice, it suggests that our algorithm will tend to stabilize the system whenever possible. Our second theoretical result, stated in Theorem 2, concerns the service rate received on average by each job as its position in the queue evolves. We show that the average per-class service rates when the number of jobs of each class is given are exactly those obtained by applying balanced fairness.

In addition to help us to understand the behavior of our algorithm, this equivalence with balanced fairness allows us to derive explicit expressions for the performance metrics of the multi-server queue with an arbitrary graph of compatibilities. Indeed, the insensitivity property satisfied by balanced fairness was used for instance in [3,4,19] to obtain simple and explicit recursion formulas for the performance metrics. Thanks to the aggregation we propose, these formulas can be applied as they are in the multi-server queue. They predict the exact performance of our algorithm when the job sizes are exponentially distributed. For an arbitrary job size distribution, we show by simulation that the system becomes approximately insensitive when the number of interruptions per job increases, so that the performance tends to that obtained under balanced fairness. We further observe that only a few interruptions per job actually suffice to reach approximate insensitivity.

The remainder of the paper is organized as follows. In Section 2, we introduce the model and give the stability condition after recalling results on OI queues. In Section 3, it is shown that the resource allocation is balanced fairness in the presence of reentrant jobs. This result is used in Section 4 to design our scheduling algorithm. Some numerical results are presented in Section 5. Section 6 concludes the paper.

## 2. A multi-server queue

We consider a multi-server queue with $N$ job classes and $S$ servers. The class of a job may identify a client of the data center or a type of service; it defines the set of servers that can process this job. For each $i = 1, \dots, N$, class-$i$ jobs enter the queue according to an independent Poisson process of intensity $\lambda_i$. The job sizes are independent, exponentially distributed with mean 1. We assume for now that each job leaves the queue immediately after service completion.

For each $i = 1, \dots, N$, we denote by $\mathcal{S}_i \subset \{1, \dots, S\}$ the set of servers that can process class-$i$ jobs. Equivalently, these constraints can be represented as a bipartite graph of compatibilities between the $N$ job classes and the $S$ servers, where there is an edge between class $i$ and server $s$ if and only if $s \in \mathcal{S}_i$. Each job can be served in parallel by multiple servers and each server processes the job sequentially in FCFS order. Hence, when there are several servers available for a job at its arrival, all these servers process this job. When the service of a job is complete, all the servers that were processing it are reallocated to the next job they can serve in the queue. There is no service preemption, so that at most one job of each class can be served at any given time.

We describe the evolution of the sequence of jobs in the queue, ordered by their arrival times. Thus the queue state is some sequence $c = (c_1, \dots, c_n)$ of length $n$, where $n$ is the number of jobs in the queue and $c_k \in \{1, \dots, N\}$ is the class of job in position $k$, for each $k = 1, \dots, n$, starting from the head of the queue. $\emptyset$ denotes the empty state, with $n = 0$.

When a job is in service on several servers, its service rate is the sum of the capacities of the servers that are processing it. Denoting by $\mu_s > 0$ the capacity of server $s$ for each $s = 1, \dots, S$, the total service rate in any state $c$ is thus given by

$$\mu(c) = \sum_{s \in \bigcup_{k=1}^{n} \mathcal{S}_{c_k}} \mu_s.$$