García-Carballeira, F., Calderón, A. & Carretero, J. Enhancing the power of two choices load balancing algorithm using round robin policy. *Cluster Comput* **24**, 611–624 (2021).

DOI: 10.1007/s10586-020-03139-6

Enhancing the Power of Two Choices Load Balancing Algorithm using Round Robin Policy

Felix Garcia-Carballeira,Alejandro Calderon, Jesus Carretero

Avda. Universidad 30, 28911 Leganés, Madrid, Spain

Computer Science and Engineering Department

Carlos III University of Madrid

Abstract

This paper proposes a new version of the *Power of Two Choices*, SQ(d), load balancing algorithm that improves the performance of the classical model based on the power of two choices randomized load balancing. This model considers jobs that arrive to a dispatcher as a Poisson stream of rate λn, $\lambda < 1$, at a set of n servers. Using the power of two choices, the dispatcher chooses for each job some d constant independently and uniformly from the n servers in a random way, and sends the job to the server with the fewest number of jobs. This algorithm offers advantage over the load balancing based on shortest queue discipline, because it offers a good performance, and reduces the overhead over the servers and over the communication network. In this paper, we propose a new version, *Shortest Queue of d with Randomization and Round Robin Policies*, SQ-RR(d), that combines randomization techniques and static local balancing based on round robin policy. In this new version the dispatcher chooses the d servers as follows: one is selected using round robin policy and the $d-1$ servers are chosen independently and uniformly in a random way from the n servers. Then, the dispatcher sends the job to the server with the fewest number of jobs. We demonstrate with a theoretical approximation of this approach, that this new version improves the performance obtained with the classical solution in all situations, included systems at 99 percent of capacity. Furthermore, we provide simulations that demonstrate the theoretical approximation developed.

Keywords: The Power of Two Choices, Load Balancing, Distributed Systems.

1. Introduction

This paper takes as basis the work and results obtained in [21], where the following model, called *Supermarket Model* is described: independent jobs arrive as a Poisson stream of rate λn, $\lambda < 1$, at a set of n homogeneous servers. For each job, d servers are chosen. These servers are chosen independently and uniformly at random with replacement from the n servers for some fixed constant d. The job is sent to the server with the lowest number of jobs. In case of ties, the server is chosen arbitrarily. Jobs are served according to the first-in first-out (FIFO) policy, and the service time for each job is exponentially

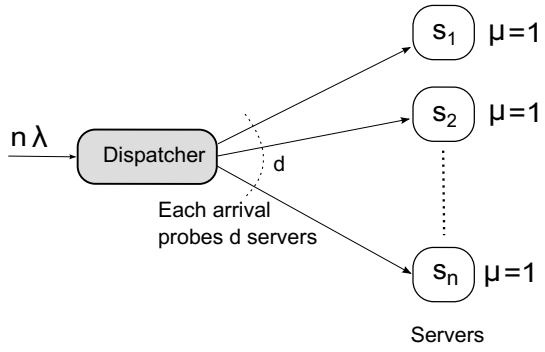

Figure 1: System Model.

distributed with mean 1. This algorithm is called *The Power of two choices*, SQ(d). This algorithm produces exponentially improved response time over the random algorithm, that chooses the server to send the job randomly, and reduces the communication overhead of the shortest-queue algorithm (JSQ), that dispatches jobs to the server with the least number of jobs.

The power of two choices has multiple applications[22]: It can be applied to hashing, in order to reduce the maximum time required to search a hash table; Another area where this method can be applied is load balancing with limited information, to dynamically assign tasks to servers, disks, or network servers; The method can also be applied to low-congestion circuit routing. In [19], the application of the power of two choices to Bloom filters is described, and it is demonstrated that this method reduces the false positive probability using the same amount of space and more hashing. SQ(d) has also been used in storage systems. For example, Dai et al. [7] use the method in a dynamic I/O scheduler for object storage systems.

Recently the power of two choices has been used in several real-world systems. For example, it has been used in Apache Storm [23], a distributed realtime stream processing engine, for the streaming partitioning. It is also used in Spark [24], a fast and general engine for large-scale data processing. Spark uses the power of two choices to reduce the number of partitions of Resilient Distributed Datasets (RDD). A RDD is a distributed collection of items that is the core of Sparks's fault tolerance. The coalescing operations on RDD are performed by using the power of two choices algorithm.

The main contribution of this paper is to propose a variation of this algorithm that improves the performance obtained with the classical Power of two Choices algorithm. The new algorithm selects also d servers, but not in a completely random way. One server is selected in a round robin fashion, and the other $d-1$ servers are selected randomly. Then, the job is sent to the server with the fewest number of jobs. In the new algorithm, we combine randomization and a static load balancing approach using round robin selection. We call this algorithm SQ-RR(d) (*shortest queue of d with randomization and round robin*). The advantage of this method, as is shown in the Simulation Results Section, is that it reduces the queue average size, by reducing the probability of choosing a server, recently chosen. This is due to

the actual arrival process to each server is characterized by a random distribution with a coefficient of variation less than the coefficient of variation of the actual arrival process to each server in the SQ(d) algorithm.

The rest of the paper is organized as follows. First, Section 2 analyzes the related work. Section 3 shows a previous analysis of SQ(d) algorithm. Section 4 describes the new algorithm proposed, *Shortest Queue of d with Randomization and Round Robin Policies*. Section 5 obtains a theoretical approximation for the SQ-RR(d) algorithm based on analyzing the potential arrivals process to each server. Section 6 shows some results obtained by simulation, and compares the SQ(d) and SQ-RR(d) algorithms. Finally, Section 7 summarizes the major conclusions extracted from this work.

2. Related Work

Load balancing is a classical problem in distributed systems, that tries to distribute objects, as for example jobs or tasks [25], to compute resources in order to maximize the performance. This maximization is normally focused on optimizing resource usage, maximizing throughput, minimizing response time, avoiding idle resources, and avoiding overloaded resources. A classical example is the distribution of a set of tasks or jobs among a set of processors in order to reduce the final completion time. This problem is specially important in current large scale distributed systems.

Load balancing strategies can be static or dynamic. Static load balancing algorithms do not use state information to distribute the objects. For example, round robin or random policies are examples of this kind of algorithms. Dynamic policies use the system state to distribute the objects, in order to react to the system behavior.

The *Join-the-Shortest-Queue* (JSQ) policy is a strategy that assigns a new job to the computing resource with the fewest number of jobs. This policy is known that posses certain optimality properties [13] [29] [26], [9]. However, it requires time to know the queue length at each server at the arrival of every job, increasing the communication overhead, and reducing in real systems the effectivity of this algorithm.

The power of two choices, SQ(d), algorithm uses partial information of the computing resources load, reducing the amount of information to be retrieved. This randomized load balancing algorithm has been studied theoretically in several works [28] [21] [8] [18]. These studies show that SQ(d) algorithm produces exponentially improved response time over the random algorithm. Furthermore, the communication overhead is greatly reduced over the JSQ model. Vvedenskaya [28] found that, for $d \geq 2$, as the number of queues n goes to infinity, the limiting probability that the number of jobs in a particular queue is at least k is given by $\lambda^{(d^k-1)/(d-1)}$. This value is substantially better than the case $d = 1$ (random policy), where the corresponding probability is λ^k.

In [6] the SQ(d) model is studied for servers with different service times. In this work, upon the arrival

of a new job, the system chooses d servers with a probability proportional to their service rates, and then it sends the job to the server with the fewest number of jobs. This paper also studies the diffusion limits of the queue length processes and the workload processes. Lu et al. [17] propose an algorithm called Join-Idle-Queue (JIQ) for distributed load balancing in large distributed systems. The idea of this algorithm is that idle processors inform dispatchers at the time of their idleness, without interfering with job arrivals. Idle processors decide which dispatcher to inform by using the SQ(d) algorithm.

Bramson et al. [1] analyzed the SQ(d) algorithm for general service time distributions, and in [2] they studied the asymptotic independence of queues under random load balancing. Izaguirre et al. [14] analyze SQ(d), with $d = 2$, for heterogeneous servers and general job requirement distribution, focusing on the light traffic regime under a fixed number of servers. In [27] a system is described, that uses the random choices to perform data aware scheduling, in order to minimize the time taken by tasks to read their inputs, for a DAG of tasks. Breitgand et al. [4] propose an Extended Supermarket Model, and they show that there is an optimal number of servers that should be monitored to obtain minimal average service time at a given cost. Xu et al. in [30] study the Supermarket Game model, similar to the Supermarket Model, but taking into account the cost for both waiting and sampling a queue.

As far as authors know, any previous work has been done, combining randomization and round robin policies. In next section we propose an analytical approximation, providing that this new algorithm enhances the results of the SQ(d) algorithm.

3. Previous Analysis of SQ(d)

This section presents the analysis of the initial SQ(d) algorithm. This analysis of the SQ(d) algorithm is based on the work developed in [1] and [3], that considers, for simplicity, the SQ(2) policy, and indicates the corresponding solution for SQ(d), with $d > 2$. This section describes first the analysis of the SQ(d) model, in order to understand the the SQ-RR(d) model that we present in Section 5

Consider any particular server (say server 1) in the system, and the arrivals that have this server 1 as one of its two possible destinations. These arrivals constitute the *potential arrival process* at server 1. As the jobs arrive to the system as a Poisson stream of rate λn, $\lambda < 1$, it can be assumed, for $n \to \infty$, that the potential arrival process at server 1 is given by the superposition of two Poisson streams of rate λ. For SQ(d), the potential arrival process to the server 1 is given by the superposition of d Poisson streams of rate λ. The superposition of two Poisson streams of rate λ is a Poisson stream of rate 2λ. For a finite n, the actual arrival process to server 1 is not Poisson since a potential arrival to server 1 becomes an actual arrival depending of the number of jobs at the other servers. But, when $n \to \infty$, the number of jobs present at the servers becomes independent of each other [1], and then the actual arrival process converges to a Poisson process.

At any time t, we define $\pi_j(t)$ as the fraction of queues having j jobs at time t, and π_j, the fraction of

queues with j jobs in equilibrium. Let $P_k = \sum_{j \geq k} \pi_j$ be the tail of the equilibrium queue-size distribution at queue 1 in the limit. Note that P_k is also equal to the asymptotic fraction of queues with at least k jobs [1].

Consider a n-queue system in equilibrium where the queue 1 has k jobs at some time t. In the SQ(2) system, a potential arrival to queue 1 also samples other queue that has j jobs. The potential arrival becomes an actual arrival, joining to queue 1, if $k < j$ or, if $k = j$, it becomes an actual arrival with probability 0.5. The probability that the potential arrival becomes an actual arrival is $(P_k + P_{k+1})/2$. As $n \to \infty$ the actual arrivals occur following a state-dependent Poisson process of rate λ_k when the size of the queue is k:

$$\lambda_k = 2\lambda \frac{(P_k + P_{k+1})}{2} = \lambda \left(P_k + P_{k+1}\right) \tag{1}$$

For $d > 2$, the actual arrival λ_k is given by:

$$\lambda_k = \lambda \left(\frac{(P_k)^d - (P_{k+1})^d}{P_k - P_{k+1}} \right) \tag{2}$$

According to [1], the queue 1, with a state-dependent Poisson arrival process is a simple birth-death chain:

$$\pi_{k+1} = \lambda_k \pi_k \Leftrightarrow P_{k+1} - P_{k+2} = \lambda(P_k - P_{k+1}) \tag{3}$$

In [1] the equations (1) y (2) are solved, and the solution obtained is:

$$P_k = \lambda^{\frac{d^k - 1}{d-1}}$$

The actual arrivals to queue 1 occur following a state-dependent Poisson process of rate λ_k when the size of the queue is k. The effective actual arrivals rate to queue 1 for $d = 2$ is given by:

$$\lambda_{eff} = \sum_{i=0}^{\infty} \lambda_i \pi_i = \sum_{i=0}^{\infty} \lambda(P_i + P_{i+1})(P_i - P_{i+1}) = \sum_{i=0}^{\infty} \lambda(P_i^2 - P_{i+1}^2) = \lambda$$

For $d \geq 2$, it can be demonstrated, that the effective actual arrivals rate to queue 1 is also λ.

The average number of jobs in the system is given by

$$L = \sum_{i=0}^{\infty} i\pi_i = \sum_{i=0}^{\infty} i(P_i - P_{i+1}) = \sum_{i=1}^{\infty} \lambda^{\frac{d^i-1}{d-1}}$$

Using the Little's law, $L = \lambda W$, we can obtain the average time job spends (W):

$$W = \frac{1}{\lambda} \sum_{i=1}^{\infty} \lambda^{\frac{d^i-1}{d-1}} = \sum_{i=1}^{\infty} \lambda^{\frac{d^i-d}{d-1}}$$

which is also the result obtained [21]. In [21] it is demonstrated that the system described before is stable for $\lambda < 1$.

4. Shortest Queue of d with Randomization and Round Robin Policies (SQ-RR(d))

We consider as model a system with n homogeneous servers, and jobs that arrive as a Poisson stream of rate λn, $\lambda < 1$. For each job, d servers are chosen, but not in completely random way. In the SQ-RR(d) algorithm, one server is selected in a round robin way, and the other $d-1$ servers are selected randomly. Then, the job is sent to the server with the fewest number of jobs. In case of ties, the server is chosen arbitrarily. Jobs are served according to the first-in first-out (FIFO) policy, and the service time for each job is exponentially distributed with mean $\frac{1}{\mu} = 1$.

In this algorithm we combine randomization and a static load balancing approach using round robin policy. With this method, the queue average size is reduced, by reducing the probability of choosing a server, recently chosen.

In next sections we provide an approximation to the expected time that a job spends in the system. This approximation takes as basis the average waiting time for the SQ(d) model. Then, we compare SQ(d) and SQ-RR(d) models. The main difference of these models is that the potential arrival process to each server is different. In the SQ(d) algorithm, the potential arrival process is exponential, and in the SQ-RR(d) the potential arrival process is given by the superposition of two Erlang distributions. Finally, we derive the average waiting time for the SQ-RR(d) using the technique called *single queue approximation* (SQA) [11], that analyzes the entire multi server model SQ-RR(d), using a single queue, and models its behavior independently of all the other queues. Finally, we apply the Kingmans' formula [16] to the single queue approximation, to obtain an approximation to the average waiting time for the SQ-RR(d) algorithm.

5. Analysis of the SQ-RR(d)

The analysis of the SQ-RR(d) is based on knowing the distribution of the potential arrival process to server 1. In SQ-RR(d) algorithm, the potential arrival process to server 1 is given by the superposition

of two distributions. The first distribution, due to the round robin selection [12] is an Erlang $Erl(n, n\lambda)$, where the inter-arrival time in the server 1 is a sum of n exponential phases, with mean durations of $1/n\lambda$. Jobs arrive at the end of phase n. The second distribution (when $d = 2$) is given by an exponential distribution with rate λ, i.e. with mean $1/\lambda$. This second distribution can be consider as an Erlang distribution with parameters $n = 1$, and mean $1/\lambda$, i.e and $Erl(1, \lambda)$.

To obtain the superposition of two Erlangs distributions, we use the work developed in [10]. Let the parameters of the two merging Erlang distributions $X \sim Erl(1, \lambda)$ and $Y \sim Erl(n, n\lambda)$. Let $f(x)$, $F(x)$, and $\overline{F}(x)$ denote the probability density function, the cumulative density function, and the complementary cumulative density function, respectively of the first arrival process. Let $g(y)$, $G(y)$, and $\overline{G}(y)$ denote the same functions for the second arrival process. According to [10], the density function, $h(x)$ of the superposed process is defined by:

$$
\begin{aligned}
h(x)= & \\
& \frac{1}{E[X]+E[Y]}\left(f(x)\int_0^\infty \overline{G}(x+u)du+\overline{F}(x)\int_0^\infty g(x+u)du\right) \\
& +\frac{1}{E[X]+E[Y]}\left(g(x)\int_0^\infty \overline{F}(x+u)du+\overline{G}(x)\int_0^\infty f(x+u)du\right)
\end{aligned}
$$

In order to obtain the mean and the variation for the inter arrival time of the superposed process, given by the above density function $h(x)$, we need to know the first and the second non-central moments of the superposed process. To this aim, we use the results presented in [10], that describes the method used to obtain the higher order approximations to the merging of Erlang distributions. The $k-th$ unconditional non-central moment of the superposition of two Erlang distributions is given by:

$$
M^{(k)}=\sum_{v=1}^{2}\sum_{w=1}^{2}\phi_{kvw}
$$

where $\phi_{kvw}, v{=}1, 2; w{=}1, 2$, defines the four components of the $k-th$ moment ($M^{(k)}$) of the superposed process, that is obtained in [10].

The values of $\phi_{kvw}, v{=}1, 2; w{=}1, 2$ are obtained in [10] using the conditional moments, $M_{ij}^{(k)}(u)$, of the superposed process. Taking into account that the two superposed Erlang distributions are characterized by the following parameters: $X{\sim}Erl(1, \lambda)$ and $Y{\sim}Erl(n, n\lambda)$, the four components of the $k-th$ unconditional moment ($M^{(k)}$) are derived according to [10] as:

$$
\begin{aligned}
M_{12}^{(k)}(u) &= \frac{1}{\overline{G}(u)}\int_0^\infty x^k\overline{F}(x)g(x+u)dx \\
&= \frac{1}{\overline{G}(u)}\int_0^\infty x^k\frac{(n\lambda)^n(x+u)^{n-1}e^{-(-x+u)n\lambda}}{(n-1)!}e^{-\lambda x}dx
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{\overline{G}(u)} \int_0^{\infty} x^k (n\lambda)^n e^{-(-x+u)n\lambda} \sum_{h=0}^{n-1} \frac{u^{n-1-h} x^h}{h!(n-1-h)!} e^{-\lambda x} dx \\
&= \frac{(n\lambda)^n}{\overline{G}(u)} \sum_{h=0}^{n-1} \frac{(h+k)!}{h!(n-1-h)!(\lambda+n\lambda)^{h+k+1}} u^{n-1-h} e^{-n\lambda u}
\end{aligned}
$$

where the following identity have been used

$$\int_0^{\infty} x^k e^{-tx} dx = \frac{k!}{t^{k+1}}$$

Due to $E[X]+E[Y]=(n\lambda+n\lambda)/\lambda n\lambda)=2/\lambda$, in [10] the component ϕ_{k12} is obtained as:

$$
\begin{aligned}
\phi_{k12} &= \int_0^{\infty} \frac{\overline{G}(u) M_{12}^{(k)}(u)}{2/\lambda} du \\
&= \frac{\lambda}{2} \int_0^{\infty} \overline{G}(u) M_{12}^{(k)}(u) du \\
&= \frac{(n\lambda)^n \lambda}{2} \sum_{h=0}^{n-1} \frac{(h+k)!}{h!(n-1-h)!(\lambda+n\lambda)^{h+k}} \int_0^{\infty} u^{n-1-h} e^{-n\lambda u} du \\
&= \sum_{h=0}^{n-1} \frac{(h+k)!}{h!} \frac{\lambda (n\lambda)^{h+1}}{2n\lambda(\lambda+n\lambda)^{h+k+1}}
\end{aligned}
$$

Similarly, we have, according to [10]:

$$
\begin{aligned}
M_{11}^{(k)}(u) &= \frac{1}{\overline{G}(u)} \int_0^{\infty} x^k \overline{G}(x+u) f(x) dx \\
&= \frac{1}{\overline{G}(u)} \int_0^{\infty} x^k \lambda e^{-\lambda x} \sum_{j=0}^{n-1} \frac{(n\lambda)^j (x+u)^j}{j!} e^{-n\lambda(x+u)} dx \\
&= \frac{1}{\overline{G}(u)} \int_0^{\infty} x^k \lambda e^{-\lambda x} \sum_{j=0}^{n-1} (n\lambda)^j \sum_{h=0}^{j} \frac{x^h u^{j-h}}{h!(j-h)!} e^{-n\lambda(x+u)} dx \\
&= \frac{\lambda}{\overline{G}(u)} \sum_{j=0} n-1 \sum_{h=0}^{j} \frac{(h+k)!(n\lambda)^j}{h!(j-h)!(\lambda+n\lambda)^{h+k+1}} u^{j-h} e^{-n\lambda u}
\end{aligned}
$$

and

$$
\begin{aligned}
\phi_{k11} &= \int_0^{\infty} \frac{\overline{G}(u) M_{11}^{(k)}(u)}{2/\lambda} du \\
&= \frac{\lambda}{2} \sum_{j=0}^{n-1} \sum_{h=0}^{j} \frac{(h+k)!(n\lambda)^j}{h!(j-h)!(\lambda+n\lambda)^{h+k}} \int_0^{\infty} u^{j-h} e^{-n\lambda u} du \\
&= \sum_{j=0}^{n-1} \sum_{h=0}^{j} \frac{(h+k)!\lambda^2 (n\lambda)^h}{h!(2n\lambda)(\lambda+n\lambda)^{h+k+1}}
\end{aligned}
$$

and due to symmetry, the last two components can be obtained [10] as:

$$\begin{aligned}\phi_{k21} &= \int_0^\infty \frac{\overline{F}(u)M_{21}^{(k)}(u)}{2/\lambda}du \\ &= \sum_{j=0}^{n-1}\frac{(j+k)!}{j!}\frac{\lambda(n\lambda)^{j+1}}{2n\lambda(\lambda+n\lambda)^{j+k+1}}\end{aligned}$$

$$\begin{aligned}\phi_{k22} &= \int_0^\infty \frac{\overline{F}(u)M_{22}^{(k)}(u)}{2/\lambda}du \\ &= \frac{(k+n-1)!}{(n-1)!}\frac{(n\lambda)^{n+1}}{2n\lambda(\lambda+n\lambda)^{k+n}}\end{aligned}$$

Once obtained, the two unconditional non-central moments $M^{(1)}$ and $M^{(2)}$ can be obtained as:

$$M^{(1)}=\sum_{v=1}^{2}\sum_{w=1}^{2}\phi_{1vw}$$
$$M^{(2)}=\sum_{v=1}^{2}\sum_{w=1}^{2}2\phi_{2vw}$$

5.1. Approximation to the Actual Arrivals Process for SQ-RR(d)

When we use the SQ-RR(d) algorithm, the potential arrivals process to each server is given by a distribution P obtained by the superposition of an Erlang and an exponential distribution, with a mean interrarival time of $1/2\lambda$, and moments $M^{(j)}$ obtained as before. We can see this potential arrival process like a GI flow with arrival rate 2λ and variance $M^{(2)}-M^{(1)}\times M^{(1)}$. The actual arrivals process to queue 1, is a state-dependent arrivals process with rate λ_k, where k is the number of jobs in the queue, but using the argument described in Section 3, we can approximate the effective actual arrivals process to queue 1 by the splitting of the potential arrival in d streams, and selecting one of this d streams.

Let A denote the random variable of one of the d streams. The moment generating function of A is:

$$M_A(s)=\frac{(1/d)M_P(s)}{1-(1-1/d)M_P(s)}$$

where $M_P(s)$ is the moment generating function of the random variable P defined in Section 3.2. The two first non-central moments of A can be obtained using the algorithms developed in [15].

$$\begin{aligned}E[A] &= \frac{E[P]}{1/d} \\ E[A^2] &= \frac{E[P^2]}{1/d}+\frac{2(1-1/d)}{(1/d)^2}(E[P])^2\end{aligned}$$

The effective actual arrivals process to queue 1, can be approximated by a distribution A with mean of μ_A=$E[A]$=$1/\lambda$ and variance of σ_A^2=$E[A^2]$−$(E[A])^2$. The main characteristic of this distribution is that the coefficient of variation cv_A^2=$\frac{\sigma^2}{\mu^2}$<1, as we can see in Figure 2 for n=100. In the SQ(d) algorithm, the effective actual arrivals process depends on an exponential distribution, and their coefficient of variation is cv^2=1, for all values of d. The variability of the A distribution in SQ-RR(d) is less than the obtained in SQ(d). This is due to the fact that one of the individual superposed processes is an Erlang distribution with relatively low variability. This effect reduces the average number of jobs in the system, and then, the average waiting time for a job.

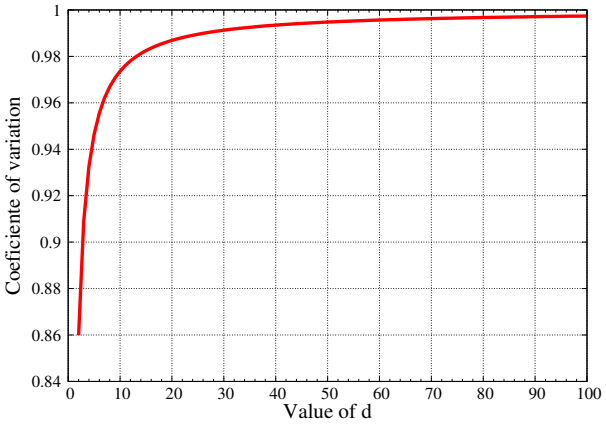

Figure 2: Coefficient of variation approximated of the effective actual arrivals process in the SQ-RR(d) for n=100

5.2. Single Queue Approximation (SQA) for SQ-RR(d)

To obtain an approximation to the average response time of SQ-RR(d) model we use the reasoning used in [11] that uses the technique called *single queue approximation* (SQA). The idea of this technique is to analyze the entire multi server model SQ-RR(d), using a single queue, and model its behavior independently of all the other queues. First, we approximate the SQ(d) model using SQA.

The SQ(d) model acts as an $M/M/n/SQ(d)$ queueing model, where SQ(d) is the policy used to route job arrivals to the servers. Jobs arrive to the system as a Poisson stream with rate $n\lambda$ and they are sent to one of the d servers, from n, chosen randomly with the fewest jobs. The server time for each job is exponentially distributed with mean u^{-1}=1, and jobs are served according to the FIFO policy.

The SQA approximation models a queue Q in the SQ(d) system by a queue Q', where the arrival rate into Q' is given by

$$\lambda_k = \lambda \left(\frac{(P_k)^d - (P_{k+1})^d}{P_k - P_{k+1}} \right)$$

where potential arrivals occur at the queue Q as a Poisson process of rate 2λ, and the actual arrivals occur following a state-dependent Poisson process of rate λ_k when the size of the queue Q is k. However,

as $n \to \infty$, the background distribution [1] becomes independent of queue Q, and the actual arrival process is a Poisson process with rate that depends only of the number of jobs in queue Q'. This idea allows us to write:

$$M/M/n/SQ(d) \approx M_n/M/1$$

where M_n denotes a state-dependent Markovian arrival process.

We can use a similar argument for $M/M/n/SQ-RR(d)$ and write:

$$M/M/n/SQ-RR(d) \approx G_n/M/1$$

where G_n denotes a general state-dependent arrival process with an actual arrival process that depends on the A distribution described in Section 5.1.

Using the Kingmans' approximation [16]:

$$W_q(G_n/M/1) \approx W_q(M_n/M/1)\left(\frac{cv_A^2+1}{2}\right)$$

being W_q the queue average waiting time and cv_A^2 is the coefficient of variation of the random distribution A obtained in section 5.1.

As $W = W_q + \frac{1}{\mu} = W_q + 1$ then, we can approximate the average response time for a job in the SQ-RR(d) model as:

$$W(G_n/M/1) = W_q(G_n/M/1) + 1 \approx W_q(M_n/M/1)\left(\frac{cv_A^2+1}{2}\right) + 1$$

and finally

$$W_{SQ-RR(d)} \approx (W_{SQ(d)} - 1)\left(\frac{cv_A^2+1}{2}\right) + 1 \tag{4}$$

where $W_{SQ(d)}$ is the time obtained in (1). As $cv_A^2 < 1$, then

$$W_{SQ-RR(d)} < W_{SQ(d)} \tag{5}$$

for all values of d.

6. Simulation Results

To evaluate the behaviour of SQ-RR(d) algorithm and to show its feasibility, we have implemented a simulator to run the tests and to compare the results with SQ(d). The simulation has been made using the SimGrid [5] simulation environment, that allows to study the behavior of large-scale distributed systems. We used the Mersenne Twister algorithm [20] as random number generator. This algorithm has a large period length ($2^{19937}-1$) and good spectral properties where correlation structures within the random number sequence are very small.

In this Section, first, we validate our simulator comparing our results with the results obtained by Mitzenmacher in [21]. Once the simulator has been validated, we show the results obtained for the SQ-RR(d) model and compare those results with the theoretical approximation obtained in Section 5.

6.1. Validation of the Simulation

In order to validate our simulator, this section compares our results with the obtained in [21] for the SQ(d) model. We simulated a system of n=100 servers at various rates. The results are based on the average of 100 runs, where each execution simulates 100,000 arrivals. The first 10,000 arrivals are ignored in order to obtain the results in equilibrium. Table 1 shows the theoretical value of SQ(d), the results and relative error obtained in [21], and the results and relative error obtained in our simulator. As we can see in the table, our results offers a relative error very similar to the obtained in [21], and for rates of up to 95 percent of the service (λ=0.95 and λ=0.99) is better than the obtained in [21].

Table 2 compares the results for a system with n=500 servers and λ=0.99 (99 percent of capacity). In a similar way to [21], the simulation results improve when the number of servers is increased. In this case, the results obtained are within 3 percent when two, three or five queues are selected.

The results obtained in this section allow us to validate the simulator developed.

6.2. Results for the SQ-RR(d) Algorithm

This section shows the results obtained for the SQ-RR(d) model. The objective of this evaluation is focused on small values of d (2 or 3), because the objetive of the power of two choices load balancing algorithm is to reduce the communications among dispatchers and servers. Furthermore, when the d value is increased (see Figure 2), the differences between SQ(d) and SQ-RR(d) become smaller, and both models converge to the shortest queue first (JSQ) algorithm.

We simulated a system of n=100 and n=1000 servers at various rates. The results are based on the average of 100 runs, where each execution simulates 100,000 arrivals. The first 10,000 arrivals are ignored in order to obtain the results in equilibrium. Table 3 shows the results for n=100 servers, and Table 4 shows the results for n=1000 servers. Both tables compare the average response time for SQ-RR(d) algorithm and the average response time for SQ(d), showing the theoretical values for SQ(d) and SQ-RR(d) respectively, the simulation results for SQ(d) and SQ-RR(d) obtained with our simulator, and

Table 1: Average response time for SQ(d) algorithm, comparing with the results obtained in [21]. Number of nodes = 100

d	λ	Theor.	Simul. in [21]	Error (%) [21]	Simul.	Error (%)
2	0.50	1.2657	1.2673	0.13	1.2742	0.67
	0.70	1.6145	1.6202	0.36	1.6265	0.75
	0.80	1.9474	1.9585	0.57	1.9684	1.08
	0.90	2.6141	2.6454	1.20	2.6543	1.54
	0.95	3.3830	3.4610	2.31	3.4489	1.95
	0.99	5.4320	5.9275	9.12	5.5992	3,08
3	0.50	1.1252	1.1277	0.22	1.1316	0.57
	0.70	1.3568	1.3634	0.48	1.3680	0.82
	0.80	1.5809	1.5940	0.83	1.6050	1.53
	0.90	2.0279	2.0614	1.65	2.0766	2.40
	0.95	2.5351	2.6137	3.10	2.5962	2.41
	0.99	3.8578	4.4080	14.26	4.1060	6.44
5	0.50	1.0312	1.0340	0.27	1.0364	0.50
	0.70	1.1681	1.1766	0.73	1.1823	1.22
	0.80	1.3289	1.3419	0.98	1.3457	1.26
	0.90	1.6329	1.6714	2.36	1.6745	2.55
	0.95	1.9888	2.0730	4.24	2.0825	4.71
	0.99	2.9017	3.4728	19.68	3.2685	12.64

finally the relative error of the SQ-RR(d) simulated value compared with the theoretical approximation (last column of the table).

The results shown in Tables 3 and 4 demonstrate that the SQ-RR(d) enhances the results obtained for the SQ(d), for the values of d simulated. For n=100 the relative error is within the 7 percent for all values of λ and d. This relative error is better when n=1000. In this case, the relative error (see Table 4) is too within 7 percent for all values, even for arrival rates of λ=0.9, λ=0.95, and λ=0.99. This demonstrate that the approximation presented in Section 5.2 is quite good.

Figure 6.2 shows, graphically, the average response time for 1000 servers and figure 4 shows the percentage of improvement of the SQ-RR(d) over the SQ(d) algorithm, for d=2 and different number of servers (n=100, n=1000, and n=10000). For a 95% confidence interval, the error is less than $\pm$2% for all values.

Figure 4 shows that the SQ-RR(d) is approximately 13% better for λ=0.9, and 15% better for λ=0.95

Table 2: Average time for SQ(d) algorithm, comparing with the results obtained in [21]. Number of nodes = 500

d	λ	Theor.	Simul. in [21]	Error (%) [21]	Simul.	Error (%)
2	0.99	5.4320	5.5413	2.10	5.3006	2.42
3	0.99	3.8578	3.8578	2.44	3.8151	1.11
5	0.99	2.9017	2.9017	3.43	2.9541	1.81

Table 3: Average response time for SQ-RR(d) algorithm, comparing with SQ(d). Number of nodes = 100

d	λ	SQ(d) Theor.	SQ-RR(d) Theor.	SQ(d) Simul.	SQ-RR(d) Simul.	Error (%)
2	0.50	1.2657	1.2311	1.2742	1.2189	0.99
	0.70	1.6145	1.5344	1.6265	1.5099	1.60
	0.80	1.9474	1.8239	1.9684	1.7824	2.28
	0.90	2.6141	2.4038	2.6543	2.3352	2.85
	0.95	3.3830	3.0725	3.4489	2.9449	4.15
	0.99	5.4320	4.8546	5.5992	4.8101	0.92
3	0.50	1.1252	1.1144	1.1316	1.1229	0.77
	0.70	1.3568	1.3258	1.3680	1.3428	1.28
	0.80	1.5809	1.5304	1.6050	1.5526	1.45
	0.90	2.0279	1.9386	2.0766	1.9847	2.38
	0.95	2.5351	2.4018	2.5962	2.4844	3.44
	0.99	3.8578	3.6096	4.1061	3.8591	6.91
5	0.50	1.0312	1.0296	1.0364	1.0364	0.60
	0.70	1.1681	1.1593	1.1823	1.1777	1.58
	0.80	1.3289	1.3118	1.3457	1.3444	2.49
	0.90	1.6329	1.5999	1.6745	1.6650	4.07
	0.95	1.9888	1.9372	2.0825	2.0480	5.72
	0.99	2.9017	2.8026	3.2685	2.9189	3.79

and λ=0.99. It is interesting to note that the percentage of improvement increases for high arrival rates.

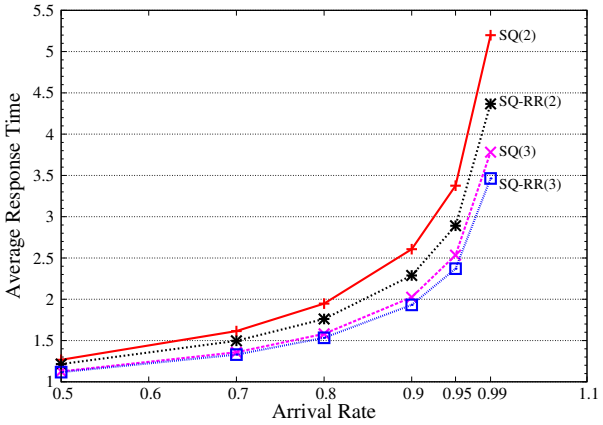

Figure 3: Average Response Time for SQ-RR(d) and SQ(d) for n=1000

Table 4: Average response time for SQ-RR(d) algorithm, comparing with SQ(d). Number of nodes = 1000

d	λ	SQ(d) Theor.	SQ-RR(d) Theor.	SQ(d) Simul.	SQ-RR(d) Simul.	Error (%)
2	0.50	1.2657	1.2306	1.2670	1.2147	1.29
	0.70	1.6145	1.5335	1.6149	1.4963	2.42
	0.80	1.9474	1.8225	1.9477	1.7612	3.36
	0.90	2.6141	2.4011	2.6064	2.2877	4.72
	0.95	3.3831	3.0686	3.3754	2.8903	5.81
	0.99	5.4320	4.8473	5.2225	4.5207	6.74
3	0.50	1.1252	1.1142	1.1260	1.1169	0.24
	0.70	1.3568	1.3255	1.3579	1.3299	0.34
	0.80	1.5809	1.5298	1.5813	1.5333	0.23
	0.90	2.0279	1.9375	2.0241	1.9323	0.27
	0.95	2.5351	2.4001	2.5342	2.3707	1.23
	0.99	3.8579	3.6064	3.7814	3.4624	3.99
5	0.50	1.0315	1.0296	1.0318	1.0301	0.14
	0.70	1.1681	1.1592	1.1700	1.1653	0.52
	0.80	1.3289	1.3116	1.3295	1.3214	0.75
	0.90	1.6329	1.5995	1.6371	1.6172	1.11
	0.95	1.9888	1.9366	1.9912	1.9587	1.14
	0.99	2.9017	2.8013	2.8943	2.8378	1.30

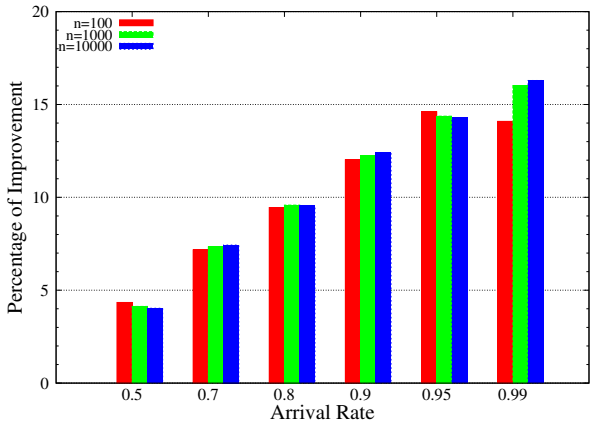

Figure 4: Percentage of improvement, SQ-RR(d) vs SQ(d) for d=2 (n=100, n=1000, and n=10000)

Figure 5 shows the percentage of improvement of the SQ-RR(d) over the SQ(d) algorithm, for d=2 and small number of servers (n=10, n=20, and n=50). This figure demonstrates that SQ-RR(d) offers a better performance, specially for high arrival rates, even for small number of servers.

Table 5 shows the probability of choosing an empty server when d=2 servers are selected using the SQ(d) and the SQ-RR(d) algorithms for n=1000 servers. These results have been obtained by simulation. The table values demonstrate that the probability of choosing an empty server is higher in the SQ-RR(d) algorithm. These results demonstrate that the SQ-RR(d) algorithm reduces the probability of choosing a server, recently chosen.

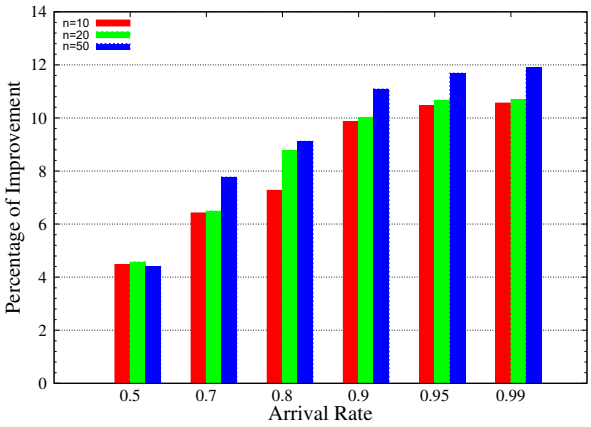

Figure 5: Percentage of improvement, SQ-RR(d) vs SQ(d) for d=2 (n=10, n=20, and n=50)

In a similar way, Table 6 shows the probability of choosing a server with 0 or 1 job in the queue. The SQ-RR(d) algorithm presents a higher probability. All these results are vey similar for other values of n.

Table 5: Probability of choosing an empty server for n=1000 and d=2

λ	SQ(d) Simul.	SQ-RR(d) Simul.
0.50	0.748	0.834
0.70	0.503	0.601
0.80	0.357	0.449
0.90	0.196	0.253
0.95	0.085	0.136
0.99	0.024	0.039

The results shown int Table 5 and 6, allow us to apply the threshold model provided in [22] to the SQ-RR(d) algorithm. In the SQ(d) threshold model, for each new job, the dispatcher chooses a single server uniformly at random. If the queue length at this first choice is at most T, el job is sent to this queue; otherwise, the dispatcher chooses a second server randomly. In [22] two variations are described. In the *strong threshold model*, if both choices are over the threshold, the job is sent to the shorter queue. In the SQ(d) *weak threshold model*, the job is sent to the second server regardless of whether the queue in this server is longer of shorter than the first. This method reduces the communication with the servers, because only one server is checked.

Applying this method to SQ-RR(d) means that when a new job arrives, the dispatcher chooses a server using round-robin, and asks to this server the number of jobs in his queue. If the queue length is at most T, the job is sent to this queue; otherwise, a second server is chosen randomly, with the weak or the strong variation.

Table 6: Probability of choosing a server with 0 or 1 job, for n=1000 and d=2

λ	SQ(d) Simul.	SQ-RR(d) Simul.
0.50	0.981	0.994
0.70	0.879	0.939
0.80	0.731	0.839
0.90	0.468	0.604
0.95	0.255	0.381
0.99	0.071	0.115

Figure 6 shows the results for the weak threshold model for n=1000 servers, comparing the performance with de SQ(d) and SQ-RR(d) model. The threshold value used in the evaluation was T=1 (for T=2, the average response time increases). Figure 7 shows the results for the strong threshold model. It is interesting to note that for the strong model, the performance of the SQ-RR(d)-Threshold is very similar to the SQ-RR(d), and always is better than the SQ(d) algorithm. The main advantage of the threshold model is that allow to reduce the communications with the servers.

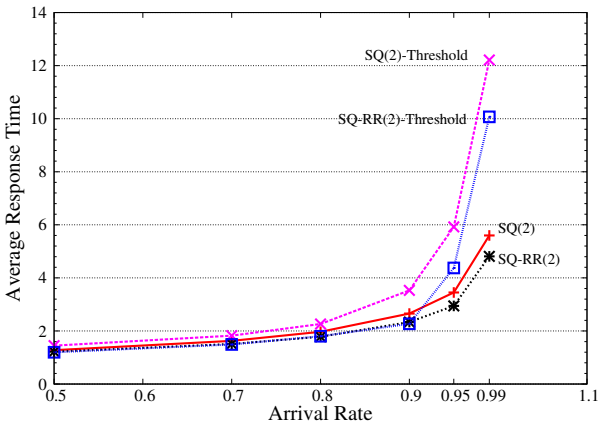

Figure 6: Average Response Time for the weak threshold model (n=1000)

7. Conclusions

This paper has described a variation of the classical Power of Two Choices load balancing algorithm. The new version, called SQ-RR(d), combines randomization techniques and static local balancing based on round robin selection. In this new version the dispatcher chooses the d servers as follows: one is selected using round robin policy, and $d-1$ servers are chosen independently and uniformly at a random from the n. Then, the dispatcher sends the job to the server with the fewest number of jobs.

We have demonstrated, with a theoretical approximation of this approach, that this new version offers

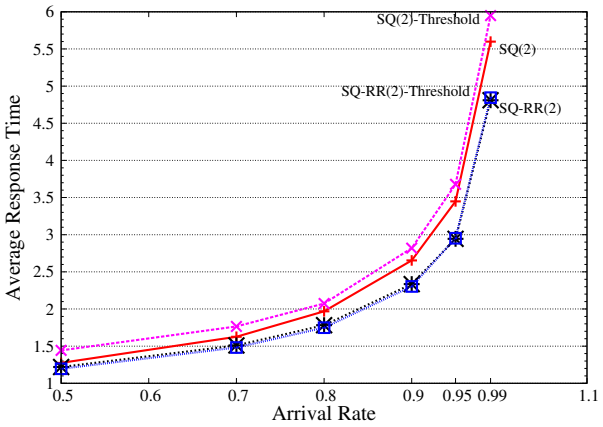

Figure 7: Average Response Time for the strong threshold model (n=1000)

a best performance than the obtained with the classical solution in all situations, including systems at 99% of capacity. The main advantage of our proposal is that it reduces the probability of choosing a server recently chosen. We have provided simulations that demonstrate the theoretical approximation developed in the paper. These simulations also demonstrate that the probability of choosing an empty server is 60 percent higher for high services rates.

Acknowledgments

This work has been partially funded under the grant TIN2013-41350-P of the Spanish Ministry of Economics and Competitiveness.

[1] M. Bramson, Y. Lu, B. Prabhakar, *Randomized Load Balancing with General Service Time Distributions*, SIGMETRICS'10, June 14-18, New York, USA, 2010.

[2] M. Bramson, Y. Lu, B. Prabhakar, *Asymptotic Independence of Queues Under Randomized Load Balancing*, Queueing Systems, 71(3), 2012, pp. 247-292.

[3] M. Bramson, Y. Lu, B. Prabhakar, *Decay of Tails at Equilibrium for FIFO Join the Shortest Queue Networks*, The Annals of Applied Probability, Vol 23, No. 5, 2013, pp. 1841-1878.

[4] D. Breitgand, R. Cohen, A. Nahir, D. Raz ,*On cost-aware monitoring for self-adaptive load sharing*, IEEE Journal on Selected Areas in Communication, 28(1), 2010, pp. 70-83.

[5] H. Casanova, A. Giersch, A. Legrand, M. Quison, and, F. Sutero, *Higher Order Approximations For The Single Server Queue With Splitting, Merging and Feedback*, Journal of Parallel and Distributed Computing 10(74), 2014, pp. 2899-2917.

[6] H. Chen, H. Q. Ye, *Asymptotic Optimality of Balanced Routing*, Operations research, 60(1), 2012, pp. 163-179.

[7] D. Dai, Y. Chen, D. Kimpe, R. Ross, *Two-choice randomized dynamic I/O scheduler for object storage systems*, In SC'14 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis 2014, pp. 635-646.

[8] L. D. Eager, E. D. Lazowska, J. Zahorjan, *Adaptive load sharing in homogeneous distributed systems*, IEEE Transactions on Software Engineering (5), 1996, pp. 662-675.

[9] A. Ephremides, P. Varaiya, J. Walrand., *A Simple Dynamic Routing Problem*, IEEE Transactions on Automatic Control, 25 (4), 1980, pp. 690-693

[10] M. K. Girish, J-Q. Hu, *Higher Order Approximations For The Single Server Queue With Splitting, Merging and Feedback*, European Journal of Operational Research 124, 2000, pp. 447-467.

[11] V. Gupta, M. Harchol-Balter, K. Sigman, W. Whitt, *Analysis of Join-the-Shortest-Queue Routing for Web Server Farms*, Performance Evaluation, Vol. 64 Issue 9-12, October, 2007, pp. 1062-1081.

[12] E. Hyytia, S. Aalto, *Round-Robin Routing Policy*, ValueTools'13, December 10-12, 2013, Turin, Italy.

[13] A. Hordijk, and G. Koole, *On the Optimaility of the Generalized Shortest Queue Policy*, Probability in the Engineering and Informational Sciences, 4(04), 1990, pp. 477-487.

[14] A. Izagirre, and A. M .Makowskid, *Light Traffic Performance Under the Power-of-Two Load Balancing Strategy: The Case of Server Heterogeneity*, Conference IFIP Performance 2014, Turin, Italy.

[15] M. A. Johnson, *Selecting Parameters of Phase Distributions: Combining Nonlinear Programming, Heuristics and Erlang Distributions*, ORSA Journal of Computing 5(1), 1993, pp. 69-83.

[16] J. F. C. Kingman, *The Single Server Queue in Heavy Traffic*, Mathematical Proceedings of the Cambridge Philosophical Society 57 (4), 1961, pp. 902-904.

[17] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, A. Greenberg*Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services*, Performance Evaluation, 68(11), 2011, pp. 1056-1071.

[18] J. M. Luczak, J. Norris, *Strong Approximation for the Supermarket Model*, The Annals of Applied Probability Vol. 15, No. 3, Aug., 2005, pp. 2038-2061.

[19] S. Lumetta, M. Mitzenmacher, *Using the Power of Two Choics to Improve Bloom Filters*, Internet Mathematics, vol 4 number 1, 2009, pp. 17-33.

[20] M. Matsumoto, T. Nishimura, *Mersenne Twister: a 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*, ACM Transactions on Modeling and Computer Simulation, 8, 1998, pp. 3-30.

[21] M. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*, IEEE Transactions on Parallel and Distributed Systems, Vol. 12, N 10, October 2001, pp. 1094-1104.

[22] M. Mitzenmacher, A. Richa, and R. Sitaraman, *The Power of Two Random Choices: A Survey of Techniques and Results*, Book chapter, in Handbook of Randomized Computing: volume 1, edited by P. Pardalos, S. Rajasekaran, and J. Rolim. 2001, pp. 255-312.

[23] M A U Nasir, G F Morales, D Garcia-Soriano, N Kourtellis, *The Power of Both Choices: Practical Load Balancing for Distributed Stream Processing Engines*, Proceedings of the 31st International Conference on Data Engineering, ICDE2015, April, 2015, Korea.

[24] The Spark Project, Lightning-fast cluster computing. http://spark.apache.org
Schroeder2004

[25] B Schroeder, M Harchol-Balter, *Evaluation of Task Assignment Policies for Supercomputing Servers: The Case for Load Unbalancing and Fairness*, Cluster Computing 7, 2004, pp.151-161

[26] R. W. Weber, *On Optimal Assignment of Customers to Parallel Servers*, Journal of Applied Probability, 15, 1978, pp. 406-413.

[27] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, I. Sotica, *The power of choice in data-aware cluster scheduling*, Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation, 2014 USENIX Association, pp. 301-316.

[28] N. D. Vvedenskaya, R. L. Dobrushin, F. I. Karpelevich, *Queueing System with Selection of the Shortest of Two Queues: An Asymptotic Approach*, Queueing System with Selection of the Shortest of Two Queues: An Asymptotic Approach, Problemy Peredachi Informatsii, 32(1), 1996, pp. 20-34.

[29] W. Winston, *Optimality of the shortest line discipline*, Journal of Applied Probability, Vol. 14, 1977, pp. 181-189.

[30] J. Xu, B. Hajek, *The supermarket game*, 2012 IEEE International Symposium on Information Theory, 2012, pp. 2511-2515.