Performance analysis of work stealing strategies in large scale multi-threaded computing

Grzegorz Kielanski[0000−0001−7149−7657] and
Benny Van Houdt[0000−0002−5955−8493]

University of Antwerp, Middelheimlaan 1, Antwerp, B-2020, Belgium
{First Name.Last Name}@uantwerpen.be

Abstract. Distributed systems use randomized work stealing to improve performance and resource utilization. In most prior analytical studies of randomized work stealing, jobs are considered to be sequential and are executed as a whole on a single server. In this paper we consider a homogeneous system of servers where parent jobs spawn child jobs that can feasibly be executed in parallel. When an idle server probes a busy server in an attempt to steal work, it may either steal a parent job or multiple child jobs.
To approximate the performance of this system we introduce a Quasi-Birth-Death Markov chain and express the performance measures of interest via its unique steady state. We perform simulation experiments that suggest that the approximation error tends to zero as the number of servers in the system becomes large. Using numerical experiments we compare the performance of various simple stealing strategies as well as optimized strategies.

Keywords: Performance analysis · Matrix analytic methods · Distributed computing.

1 Introduction

Jobs in multithreaded computing systems consist of several threads [2,24]. Upon starting the execution a main thread (which we call a parent job) several other threads are spawned (which we call child jobs). These spawned child jobs are initially stored locally, but can be redistributed at a later stage. One way of redistributing jobs is called "randomized work stealing": servers that become empty start probing other servers at random (uniformly) and if the probed server has pending jobs, some of its jobs are transferred to the probing server [2, 5]. Another option is to make use of "randomized work sharing", where servers that have pending jobs probe others to offload some of their work to other servers.
Work stealing solutions have been studied by various authors and are often used in practice. They have been implemented for example in the Cilk programming language [3, 6], Intel TBB [19], Java fork/join framework [12], KAAPI [9] and .NET Task Parallel Library [13]. Some early studies on work sharing and stealing include [5,16,22]. In [5] the performance of work stealing and sharing is compared

for homogenous systems with exponential job sizes. Using similar techniques the work in [5] was generalized to heterogeneous systems in [16]. The key takeaway from these papers is that work stealing clearly outperforms work sharing in system with high load. [22] focused on shared-memory systems and assumes that migrated jobs have a higher service demand and migrating jobs requires some time.
More recent work includes [8,14,15,21,23]. In [8] the authors analyse the system consisting of several homogeneous clusters with exponential job sizes and where half of the jobs are transferred when a probe is successful. A fair comparison between stealing and sharing strategies is given for homogeneous networks and exponential job sizes in [14,15] and for non-exponential job sizes in [23]. Further, the comparison in [15] is extended to heterogeneous networks in [21]. The key difference with the current paper is that in these prior works jobs are considered to be sequential and are always executed as a whole on a single server.

In this paper, we consider a system of homogeneous servers that uses a randomized work stealing policy. We consider a set of policies where if a server with pending child jobs is probed by an idle server, some of its child jobs are transferred. When a server is probed that does not have any pending child jobs, a pending parent job is transferred instead (if available). The work presented in this paper is closely related to [20], where two systems are considered: one system where parent jobs can be stolen and the other system where child jobs can be stolen one at a time. In the current paper we allow that several child jobs can be stolen at once and the main objective is to provide insights on how to determine the number of child jobs that should be transferred in such an event. When several child jobs can be stolen at once, child jobs may be transferred several times before being executed and this considerably complicates the analysis compared to [20]. In [20] we also introduced a mean field model and showed that this mean field model has a unique fixed point given by the steady state vector of a structured Markov chain. For the model considered in this paper a similar type of result can be established (albeit with more effort). However, due to the page limitations, we decided to directly present the structured Markov chain instead.

The main contributions of the paper are the following:

1. We introduce a Quasi-Birth-Death (QBD) Markov chain describing a single server queueing system with negative arrivals that is used to approximate the performance of the work stealing system. We present simulation results that suggest that as the number of servers becomes large, the approximation error tends to zero.
2. We prove that this QBD has a unique stationary distribution for which we provide formulas for the waiting, service, mean waiting and mean service time. These are the main technical results of the paper.
3. We compare the performance of several stealing strategies. Our main insight is that the strategy of stealing half of the child jobs performs well for low loads and/or high probe rates and stealing all child jobs is a good heuristic when the load is high and/or the probe rate is low.

The rest of this paper is organized as follows. In Section 2 we describe the system while the Quasi-Birth-Death (QBD) Markov chain is introduced in Section 3 and the response time distribution is analyzed in Section 4. In Section 5 we describe the work stealing strategies considered and present the performance of these strategies using numerical examples. Section 6 contains some concluding remarks and possible future work. The QBD approximation is validated using simulation in Appendix A.

2 System description and strategies

We consider a system with N homogeneous servers each with an infinite buffer to store jobs. Parent jobs arrive in each server according to a local Poisson arrival process with rate λ. Upon entering service a parent job spawns $i \in \{0, 1, \ldots, m\}$, with $m \geq 1$, child jobs, the number of which follows a general distribution with finite support p_i (i.e., $p_i \geq 0$ for every i and $\sum_{i=0}^{m} p_i = 1$). These child jobs are stored locally and have priority over any parent jobs (either already present or yet to arrive), while the spawning parent job continues service. Thus, when a (parent or child) job completes service the server first checks to see whether it has any waiting child jobs, if so it starts service on a child job. If there are no child jobs present, service on a waiting parent job starts (if any are present). We assume that parent and child jobs have exponentially distributed service requirements with rates μ_1 and μ_2 respectively.

When a server is idle, it probes other servers at random at rate $r > 0$, where r is a system parameter. Note that r determines the amount of communication between the servers and increasing r should improve performance at the expense of a higher communication overhead. When a server is probed (by an idle server) and it has waiting (parent or child) jobs, we state that the probe is successful. When a successful probe reaches a server without waiting child jobs, a parent job is transferred to the idle server. Note that such a transferred parent job starts service and spawns its child jobs at the new server.

When a successful probe reaches a server with pending/waiting child jobs, several child jobs can be transferred at once. If the probed server is serving a parent job and there are i child jobs in the buffer of the probed server, $j \leq i$ child jobs are stolen with probability $\phi_{i,j}$ (i.e., for every i we have $\sum_{j=1}^{i} \phi_{i,j} = 1$). On the other hand if a child job is being processed by the probed server and there are i child jobs waiting in the buffer of the probed server, $j \leq i$ child jobs are stolen with probability $\psi_{i,j}$ (i.e., for every i we have $\sum_{j=1}^{i} \psi_{i,j} = 1$). For ease of notation we set $\phi_{i,j} = \psi_{i,j} = 0$ if $j > i$. Probes and job transfers are assumed to be instantaneous.

The main objective of this paper is to study how the probabilities $\phi_{i,j}$ and $\psi_{i,j}$ influence the response time of a job, where the response time is defined as the time between the arrival of a parent job and the completion of the parent and all its spawned child jobs. Given the above description, it is clear that we get a Markov process if we keep track of the number of parent and child job in

each of the N servers. This Markov process however does not appear to have a product form, making its analysis prohibitive.

Instead we use an approximation method, the accuracy of which is investigated in Appendix A. The idea of the approximation exists in focusing on a single server and assuming that the queue lengths at any other server are independent and identically distributed as in this particular server. Within the context of load balancing, this approach is known as the cavity method [4]. In fact all the analytical models used in [5,8,14–16,20–23] can be regarded as cavity method approximations. A common feature of such an approximation is that it tends to become more accurate as the number of servers tends to infinity, as we demonstrate in Appendix A for our model. The cavity method typically involves iterating the so-called cavity map [4]. However, in our case the need for such an iteration is avoided by deriving expressions for the rates at which child and parent jobs are stolen.

3 Quasi-Birth-Death Markov chain

In this section we introduce a Quasi-Birth-Death (QBD) Markov chain to approximate the system from the viewpoint of a single server. Let $\lambda_p(r)$ denote the rate at which parent jobs are stolen when the server is idle. Let $\lambda_{c,1}(r), \ldots, \lambda_{c,m}(r)$ denote respectively the rates at which $1, \ldots, m$ child jobs are stolen. We provide formulas for these rates further on. The evolution of a single server has the following characteristics, where the negative arrivals correspond to steal events:

1. When the server is busy, arrivals of parent jobs occur according to a Poisson process with rate λ. When the server is idle, parent jobs arrive at the rate $\lambda+\lambda_p(r)$, while a batch of i child jobs arrives at rate $\lambda_{c,i}(r)$ for $i = 1, \ldots, m$.
2. Upon entering service, a parent job spawns $i \in \{0, 1, \ldots, m\}, m \geq 1$, child jobs with probability p_i. Child jobs are stored locally.
3. Child jobs have priority over any parent jobs *waiting* in the queue and are thus executed immediately after their parent job completes when executed on the same server.
4. Parent and child jobs have exponentially distributed service requirements with rates μ_1 and μ_2, respectively.
5. If there are parent jobs and no child jobs waiting in the buffer of the server then a negative parent arrival occurs at the rate rq, where $q = 1 - \rho$ is the probability that a queue is idle (where ρ is defined in (1)).
6. If a parent job is in service and there are $i \in \{1, \ldots, m\}$ child jobs in the buffer of the server, a batch of j negative child job arrivals occurs at the rate $rq\phi_{i,j}$, for all $j \in \{1, \ldots, i\}$.
7. If a child job is in service and there are $i \in \{1, \ldots, m-1\}$ child jobs pending in the buffer of the server, a batch of j negative child job arrivals occurs at the rate $rq\psi_{i,j}$, for all $j \in \{1, \ldots, i\}$.

Note that the load of the system can be expressed as

$$\rho = \lambda \left(\frac{1}{\mu_1} + \frac{\sum_{n=1}^{m} np_n}{\mu_2} \right). \tag{1}$$

Table 1: Transitions for the QBD in Section 3

	From	To	Rate	For
1.	$(0,0,0)$	$(0,j,0)$	$\lambda_{c,j}(r)$	$j=1,\ldots,m$,
2.	$(0,0,0)$	$(0,j,1)$	$(\lambda+\lambda_p(r))p_j$	$j=0,1,\ldots,m$,
3.	(X,Y,Z)	$(X+1,Y,Z)$	λ	$X+Y+Z\geq 1$,
4.	$(X,Y,1)$	$(X,Y,0)$	μ_1	$X\geq 0, Y\geq 1$ or $X=0,Y=0$,
5.	$(X,Y,0)$	$(X,Y-1,0)$	μ_2	$X\geq 0, Y\geq 2$ or $X=0,Y=1$,
6.	$(X,1,0)$	$(X-1,j,1)$	$\mu_2 p_j$	$X\geq 1, j=0,1,\ldots,m$,
7.	$(X,0,1)$	$(X-1,j,1)$	$\mu_1 p_j$	$X\geq 1, j=0,1,\ldots,m$,
8.	(X,Y,Z)	$(X-1,Y,Z)$	rq	$X\geq 1, Y+Z=1$,
9.	$(X,Y,1)$	$(X,Y-j,1)$	$rq\phi_{Y,j}$	$X\geq 0, Y\geq j, j=1,\ldots,m$,
10.	$(X,Y,0)$	$(X,Y-j,0)$	$rq\psi_{Y-1,j}$	$X\geq 0, Y\geq j+1, j=1,\ldots,m-1$.

Denote by $X\geq 0$ the number of parent jobs waiting, by $Y\in\{0,1,\ldots,m\}$ the number of child jobs in the server (either in service or waiting), and by $Z\in\{0,1\}$ whether a parent job is currently in service ($Z=1$) or not ($Z=0$). The possible transitions of the QBD Markov chain are listed in Table 1, corresponding to: 1. a batch of j child jobs arriving at an idle queue and the first child job proceeding directly into service, 2. a parent job arriving at an idle queue and proceeding directly into service, spawning j child jobs, 3. a parent arriving to a non-idle queue, 4. completion of a parent in service, not succeeded by another parent job, 5. child service completion, succeeded by either another child job or no job, 6. child service completion, succeeded by a parent job that enters service and spawns j child jobs, 7. parent service completion, succeeded by a parent job that enters service and spawns j child jobs, 8. negative parent job arrival, 9. a parent is in service and a batch of negative child job arrivals occurs, 10. a child job is in service and a batch of negative child job arrivals occurs.

The three dimensional process $\{X_t(r),Y_t(r),Z_t(r):t\geq 0\}$ is an irreducible, aperiodic Quasi-Birth-Death process. We state that the *level* $\ell=*$ when the chain is in state $(0,0,0)$, while for any state with $X=\ell$ different from $(0,0,0)$, we state that the chain is in level ℓ (for $\ell\geq 0$). When the level $\ell\geq 0$, the *phase* of the QBD is two dimensional and given by (Y,Z). The $2m+1$ phases of level $\ell\geq 0$ are ordered such that the j-th phase corresponds to $(Y,Z)=(j,0)$, for $j=1,\ldots,m$ and phase $m+1+j$ to $(Y,Z)=(j,1)$ for $j=0,\ldots,m$.

As explained below, the generator of the process is

$$Q(r)=\begin{bmatrix} -\lambda_0(r) & \sum_{j=1}^m \lambda_{c,j}(r)e_j+(\lambda+\lambda_p(r))\alpha & & \\ \mu & B_0(r) & A_1 & \\ & A_{-1}(r) & A_0(r) & A_1 \\ & & \ddots & \ddots & \ddots \end{bmatrix}$$

with $\lambda_0(r)=\sum_{j=1}^m\lambda_{c,j}(r)+\lambda+\lambda_p(r)$, with e_j a row vector with 1 in its j-th entry and zeros elsewhere. The initial probability vector α records the distribution of child jobs upon a parent job entering service and is given by $\alpha=\left[0'_m\ p_0\ p_1\ \ldots\ p_m\right]$, where 0_i is a column vector of zeros of length i. Indeed,

at rate $\lambda_{c,j}(r)$ a batch of j child jobs arrives in an idle server, causing a jump to level 0 and phase j, while at rate $\lambda + \lambda_p(r)$ a parent job arrives that spawns j child jobs with probability p_j causing a jump to phase $m+1+j$ of level 0.

Define

$$S(r) = \begin{bmatrix} S_{00}(r) & 0 \\ S_{10} & S_{11}(r) \end{bmatrix},$$

where $S_{00}(r)$ is an $m \times m$ matrix and $S_{11}(r)$ is an $(m+1) \times (m+1)$ matrix,

$$S_{00}(r) = rq \begin{bmatrix} & & \\ \psi_{1,1} & & \\ \vdots & \ddots & \\ \psi_{m-1,m-1} & \cdots & \psi_{m-1,1} \end{bmatrix} + \begin{bmatrix} -\mu_2 & & & \\ \mu_2 & \ddots & & \\ & \ddots & \ddots & \\ & & \mu_2 & -\mu_2 \end{bmatrix},$$

$$S_{10} = \begin{bmatrix} 0 & \cdots \\ \mu_1 & \\ & \mu_1 \\ & & \ddots \end{bmatrix}, \; S_{11}(r) = rq \begin{bmatrix} & & \\ \phi_{1,1} & & \\ \vdots & \ddots & \\ \phi_{m,m} & \cdots & \phi_{m,1} \end{bmatrix} + \begin{bmatrix} -\mu_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & -\mu_1 \end{bmatrix}.$$

The matrix $A_0(r)$ contains the possible transitions for which the level $\ell > 0$ remains unchanged, this is when child jobs are stolen, or when a waiting child moves into service. Hence

$$A_0(r) = S(r) - \lambda I - rqI.$$

Note that even when there are no child jobs waiting, the rate rq appears on the main diagonal due to the negative parent arrivals. When $\ell = 0$ there are no parent jobs waiting and therefore the negative parent arrivals that occur in phase 1 and $m+1$ have no impact. This implies that

$$\begin{aligned} B_0(r) &= A_0(r) + rqV_0 \\ &= S(r) - \lambda I - rq(I - V_0), \end{aligned}$$

where $V_0 = \text{diag}(\begin{bmatrix} 1 & 0'_{m-1} & 1 & 0'_m \end{bmatrix})$. The level ℓ can only decrease by one due to a service completion from a phase with no pending child jobs, that is, from phase 1 and $m+1$. To capture these events define $\mu = \begin{bmatrix} \mu_2 & 0'_{m-1} & \mu_1 & 0'_m \end{bmatrix}'$. The level can also decrease due to a negative parent arrival when $\ell > 0$. The matrix $A_{-1}(r)$ records the transitions for which the level decreases and therefore equals

$$A_{-1}(r) = \mu\alpha + rqV_0.$$

Finally, parent job arrivals always increase the level by one:

$$A_1 = \lambda I.$$

Denote by $A(r) = A_{-1}(r) + A_0(r) + A_1$, the generator of the phase process, then

$$A(r) = S(r) + \mu\alpha - rq(I - V_0).$$

Define

$$\pi_*(r) = \lim_{t\to\infty} P[X_t(r) = 0, Y_t(r) = 0, Z_t(r) = 0],$$

and for $\ell \geq 0$,

$$\pi_\ell(r) = (\pi_{\ell,1,0}(r), \ldots \pi_{\ell,m,0}(r), \pi_{\ell,0,1}(r), \ldots, \pi_{\ell,m,1}(r))$$

where

$$\pi_{\ell,j,k}(r) = \lim_{t\to\infty} P[X_t(r) = \ell, Y_t(r) = j, Z_t(r) = k].$$

Due to the QBD structure [17], we have

$$\pi_0(r) = \pi_*(r)R_0(r), \tag{2}$$

where $R_0(r)$ is a row vector of size $2m + 1$ and for $\ell \geq 1$,

$$\pi_\ell(r) = \pi_0(r)R(r)^\ell, \tag{3}$$

where $R(r)$ is a $(2m+1) \times (2m+1)$ matrix and by [11] the smallest nonnegative solution to

$$A_1 + R(r)A_0(r) + R(r)^2 A_{-1}(r) = 0.$$

Also, due to the balance equations with $\ell = 0$, we have

$$\sum_{j=1}^{m} \lambda_{c,j}(r)e_j + (\lambda + \lambda_p(r))\alpha + R_0(r)B_0(r) + R_0(r)R(r)A_{-1}(r) = 0$$

and due to [11, Chapter 6]

$$A_1 G(r) = R(r)A_{-1}(r),$$

where $G(r)$ is the smallest nonnegative solution to

$$A_{-1}(r) + A_0(r)G(r) + A_1 G(r)^2 = 0.$$

Combining the above yields the following expression:

$$R_0(r) = -\left(\sum_{j=1}^{m} \lambda_{c,j}(r)e_j + (\lambda + \lambda_p(r))\alpha\right)(B_0(r) + \lambda IG(r))^{-1}, \tag{4}$$

where $B_0(r) + \lambda IG(r)$ is a subgenerator matrix and is therefore invertible. We note that $R(r)$ and $G(r)$ are independent of $\lambda_{c,1}(r), \ldots, \lambda_{c,m}(r)$ and $\lambda_p(r)$ and can be computed easily using the toolbox presented in [1]. To fully characterize the QBD in terms of λ, μ_1, μ_2 and the probabilities $p_i, \phi_{i,j}$ and $\psi_{i,j}$, we need to specify $\lambda_{c,1}(r), \ldots, \lambda_{c,m}(r)$ and $\lambda_p(r)$.

To determine these rates we use the following observation: as all parent and child jobs are executed on some server, $q = 1 - \rho$ should be the probability that the QBD is in state $(0, 0, 0)$. In this state batches of j child jobs arrive at rate $\lambda_{c,j}(r)$. Therefore $q\lambda_{c,j}(r)$ should equal the parent arrival rate λ times the expected number of times that a batch of j child jobs is stolen per parent job. The main difficulty in using this equality lies in the fact that we must also take into account that a child job can be stolen several times before it is executed.

To this end and as a preparation for Proposition 1, we define recursively $p_{0,i}(r)$, $i = 1, \ldots, m$, as the probability that the QBD visits phase $(i, 0)$ during the service of a job and similarly $p_{1,i'}(r)$, $i' = 0, \ldots, m$ that phase $(i', 1)$ is visited. By conditioning on whether we first have a service completion or steal event, we have

$$\begin{aligned} p_{1,m}(r) &= p_m, \\ p_{1,i}(r) &= p_i + \frac{rq}{rq + \mu_1} \sum_{j>i} p_{1,j}(r)\phi_{j,j-i}, \end{aligned}$$

for $i \in \{0, \ldots, m-1\}$, and

$$p_{0,i}(r) = \frac{\mu_1}{rq + \mu_1} p_{1,i}(r) + \frac{\mu_2}{rq + \mu_2} p_{0,i+1}(r) + \frac{rq}{rq + \mu_2} \sum_{j>i} p_{0,j}(r)\psi_{j-1,j-i},$$

for $i \in \{1, \ldots, m\}$, with $p_{0,m+1} = 0$. Note that

$$p_{1,0}(r) + p_{0,1}(r) = 1, \tag{5}$$

as phase $(1, 0)$ or $(0, 1)$ is visited before any job completes service.
We also define $p_i^j(r)$, for $1 \le i \le j \le m$, as the probability that the QBD visits phase $(i, 0)$ given that it is in the phase $(j, 0)$ before a job completes service. We have

$$\begin{aligned} p_j^j(r) &= 1, \\ p_i^j(r) &= \frac{\mu_2}{rq + \mu_2} p_{i+1}^j(r) + \frac{rq}{rq + \mu_2} \sum_{k=i+1}^{j} \psi_{k-1,k-i} p_k^j(r), \end{aligned}$$

for $i \in \{1, \ldots, j-1\}$. Note that we have $p_1^j(r) = 1$, for $1 \le j \le m$, as the QBD visits phase $(0, 1)$ before completing service if it is in phase $(0, j)$. We are now in a position to define $\lambda_{c,i}(r)$ recursively as:

$$\begin{aligned} \lambda_{c,m}(r) &= \frac{\lambda}{q} p_{1,m}(r) \frac{rq}{rq + \mu_1} \phi_{m,m} \\ \lambda_{c,i}(r) &= \frac{\lambda}{q} \frac{rq}{rq + \mu_1} \sum_{j \ge i} p_{1,j}(r)\phi_{j,i} + \frac{\lambda}{q} \frac{rq}{rq + \mu_2} \sum_{j>i} p_{0,j}(r)\psi_{j-1,i} \\ &\quad + \sum_{j=i+1}^{m} \lambda_{c,j}(r) \sum_{k=i+1}^{j} p_k^j(r)\psi_{k-1,i} \frac{rq}{rq + \mu_2} \end{aligned} \tag{6}$$

for $i \in \{1, \ldots, m-1\}$. Note that $p_{1,m}(r)rq\phi_{m,m}/(rq+\mu_1)$ indeed equals the expected number of batches of size m that are stolen per parent job (as the job must spawn m child jobs and these must be stolen as a batch before the parent completes service). For $i < m$, the first two sums represent the expected number of size i batches that are stolen from the original server, while the double sum counts the expected number of such steals that occur on a server different from the original server.

It remains to define $\lambda_p(r)$, for this we demand that $\pi_*(r) = q$ and that

$$\pi_*(r) + \sum_{\ell \geq 0} \pi_\ell(r) e = 1,$$

where e is a column vector of ones. Then from equations (2) and (3),

$$q\Big(1 + R_0(r)(I - R(r))^{-1} e\Big) = 1, \tag{7}$$

where the inverse of $I - R(r)$ exists due to Proposition 1. Using (4) and (7) we get:

$$\lambda_p(r) = \frac{(1-q) - q(\sum_{j=1}^{m} \lambda_{c,j}(r) e_j + \lambda\alpha)w}{q\alpha w}, \tag{8}$$

with $w = -(B_0(r) + \lambda IG(r))^{-1}(I - R(r))^{-1}e$. Note that $\lambda_p(r)$ is well-defined for $q > 0$, i.e. $\rho < 1$. This completes the description of the QBD Markov chain.

Proposition 1. *The QBD process* $\{X_t(r), Y_t(r), Z_t(r) : t \geq 0\}$ *has a unique stationary distribution for any* $r \geq 0$ *if* $\rho < 1$.

Proof. The positive recurrence of the QBD process only depends on the matrices $A_{-1}(r)$, $A_0(r)$ and A_1 [17]. These three matrices are the same three matrices as those of the QBD characterizing the M/MAP/1 queue where the MAP service process is characterized by $(S_0(r), S_1(r))$ with $S_0(r) = S(r) - rqI$ and $S_1(r) = \mu\alpha + rqV_0$. As such the QBD process is positive recurrent if and only if the arrival rate λ is less than the service completion intensity of the MAP $(S_0(r), S_1(r))$. This intensity equals $\theta^{(r)} S_1(r) e / \theta^{(r)} e$, where the vector $\theta^{(r)}$ is such that $\theta^{(r)}(S_0(r) + S_1(r)) = 0$.

We note that $S_0(r) + S_1(r) = A_{-1}(r) + A_0(r) + A_1 = A(r)$ and define

$$\begin{aligned}
\theta^{(r)}_{(0,1)} &= \frac{1}{\mu_2} p_{0,1}(r), \\
\theta^{(r)}_{(0,i')} &= \frac{1}{rq + \mu_2} p_{0,i'}(r), \\
\theta^{(r)}_{(1,0)} &= \frac{1}{\mu_1} p_{1,0}(r), \\
\theta^{(r)}_{(1,i)} &= \frac{1}{rq + \mu_1} p_{1,i}(r),
\end{aligned}$$

for $i' = 2, \ldots, m$ and for $i = 1, \ldots, m$. Define $v^{(r)} = \theta^{(r)} A(r)$. Then, using (5),

$$v_i^{(r)} = p_{i-m-1} - p_{1,i-m-1}(r) + \frac{rq}{rq+\mu_1} \sum_{j>i-m-1} p_{1,j}(r)\phi_{j,j-i-m-1} = 0,$$

for $i = m+1, \ldots, 2m+1$, and

$$\begin{aligned} v_{i'}^{(r)} &= -p_{0,i'}(r) + 1[i<m]\frac{\mu_2}{rq+\mu_2} p_{0,i'+1}(r) \\ &\quad + \frac{rq}{rq+\mu_2} \sum_{j>i} p_{0,j}(r)\psi_{j-1,j-i'} + \frac{\mu_1}{rq+\mu_1} p_{1,i'}(r) = 0, \end{aligned}$$

for $i' = 1, \ldots, m$. Hence $\theta^{(r)} A(r) = \theta^{(r)}(S_0(r) + S_1(r)) = 0$. As

$$\begin{aligned} \frac{\theta^{(r)} S_1(r) e}{\theta^{(r)} e} &= \frac{1}{\theta^{(r)} e} \left(\frac{\mu_2 + rq}{\mu_2} p_{0,1}(r) + \frac{\mu_1 + rq}{\mu_1} p_{1,0}(r) \right) \\ &\geq \frac{1}{\theta^{(r)} e} (p_{0,1}(r) + p_{1,0}(r)) = \frac{1}{\theta^{(r)} e}, \end{aligned}$$

it suffices that $\lambda < 1/\theta^{(r)} e$ for the chain to be positive recurrent. For $r = 0$ we have $p_{1,i}(r) = p_i$ and $p_{0,i'} = \sum_{j \geq i'} p_j$, which implies that $\theta^{(0)} e = \rho/\lambda$. Therefore $\lambda < 1/\theta^{(0)} e$ is equivalent to demanding that $\rho < 1$. As $\theta^{(r)} e$ is the mean time between two service completions of the MAP process where the state is reset according to the vector α, we have that $\theta^{(r)} e$ decreases in r. This completes the proof as $\rho < 1$ implies that $\lambda < 1/\theta^{(0)} e \leq 1/\theta^{(r)} e$.

4 Response time distribution

We define $T(r)$ as the response time of a job in a system with probe rate r. The response time is defined as the length of the time interval between the arrival of a parent job and the completion of this parent job and all of its spawned child jobs. $T(r)$ can be expressed as the sum of the waiting time $W(r)$ and the service time $J(r)$. The waiting time is defined as the amount of time that the parent job waits in the queue before its service starts. Clearly, the waiting and the service time of a job are independent in our QBD model.

Theorem 1. *The distribution of the waiting time is given by*

$$P[W(r) > t] = (e' \otimes \pi_0 (I - R(r))^{-1}) e^{\mathbb{W}t} vec\langle I \rangle$$

with $\mathbb{W} = ((A_0(r) + A_1)' \otimes I) + ((A_{-1}(r))' \otimes R(r))$ *and where* $vec\langle \cdot \rangle$ *is the column stacking operator. The mean waiting time is*

$$E[W(r)] = \int_0^\infty P[W(r) > t]\, dt = (e' \otimes \pi_0 (I - R(r))^{-1})(-\mathbb{W})^{-1} vec\langle I \rangle.$$

Proof. We repeat the arguments of the proof of [20, Theorem 6.1]. Let $(N(k,t))_{j,j'}$ be the probability that there are exactly k transitions that decrease the level by one in $(0,t)$ and the phase at time t equals j' given that the level never decreased below 1 and the phase was j at time 0. Due to the PASTA property we have

$$P[W(r) > t] = \sum_{n=1}^{\infty} \pi_{n-1} \sum_{k=0}^{n-1} N(k,t)e,$$

as $(\pi_{n-1})_j$ is the probability that a tagged parent job is the n^{th} parent job waiting in the queue immediately after it arrived and the service phase equals j. In such case there can be at most $n-1$ events that decrease the level otherwise $W(r) < t$. Thus,

$$\begin{aligned} P[W(r) > t] &= \sum_{k=0}^{\infty} \pi_0 \sum_{n=k+1}^{\infty} R(r)^{n-1} N(k,t)e \\ &= \pi_0 (I - R(r))^{-1} \sum_{k=0}^{\infty} R(r)^k N(k,t)e. \end{aligned}$$

Using the same arguments as in [18] or [10] one finds that

$$vec \left\langle \sum_{k=0}^{\infty} R(r)^k N(k,t) \right\rangle = e^{\mathbb{W}t} vec \left\langle I \right\rangle .$$

The proof is completed by noting that $vec\langle ABC\rangle = (C' \otimes A) vec\langle B\rangle$.

The service time distribution $J(r)$ is more difficult to compute compared to the model in [20]. This is due to the fact that child jobs can be stolen multiple times before finally going into service.

We define $J_{0,k}(r)$ as the distribution of the time that it takes for k child jobs in a server to be completed $(k = 1, \ldots, m)$. Similarly, we define $J_{1,k}(r)$ as the distribution of the time that it takes for a parent job and k child jobs in a server to be completed $(k = 0, \ldots, m)$. The service time distribution can then be expressed as

$$P[J(r) \le t] = \sum_{k=0}^{m} p_k P[J_{1,k}(r) \le t].$$

Clearly, $P[J_{0,1}(r) \le t] = 1 - e^{-\mu_2 t}$ and $P[J_{1,0}(r) \le t] = 1 - e^{-\mu_1 t}$. For $k > 1$, we can condition on the first service completion or steal event to find that

$$\begin{aligned} P[J_{0,k}(r) \le t] = \int_0^t \Bigg(& rq \sum_{j=1}^{k-1} \psi_{k-1,j} P[J_{0,k-j}(r) \le t-s] P[J_{0,j}(r) \le t-s] \\ & + \mu_2 P[J_{0,k-1}(r) \le t-s] \Bigg) e^{-(rq+\mu_2)s} ds, \end{aligned} \tag{9}$$

and for $k > 0$ this yields

$$P[J_{1,k}(r) \leq t] = \int_0^t \left(rq \sum_{j=1}^{k} \phi_{k,j} P[J_{1,k-j}(r) \leq t-s] P[J_{0,j}(r) \leq t-s] \right.$$
$$\left. + \mu_1 P[J_{0,k}(r) \leq t-s] \right) e^{-(rq+\mu_1)s} ds. \tag{10}$$

While the above formulas recursively determine the service time, they are less suited for numerical computations, we therefore also develop a recursive scheme for the mean service time.

Consider a set of s servers, where the k-th server contains i_k child jobs, where $s \geq 1$, $0 \leq i_1 + \cdots + i_s \leq m$ and $i_k \geq 0$ for $k = 1, \ldots, s$. Let $E_{i_1,\ldots,i_s}(r)$ be the expected time until all these child jobs have completed service. Define similarly $E^p_{i_1,\ldots,i_s}(r)$, except that the first server contains i_1 child jobs and a parent job (that is in service).

By definition, we can drop i_k's that are zero (expect i_1 in $E^p_{i_1,\ldots,i_s}(r)$) and can permute the indices of $E_{i_1,\ldots,i_s}(r)$ and all indices except the first one of $E^p_{i_1,\ldots,i_s}(r)$. We have for $s \geq 1$

$$E_{1'_s}(r) = \frac{1}{\mu_2} \sum_{k=1}^{s} \frac{1}{k}. \tag{11}$$

We now define recursively, assuming $i_k \geq 1$ for $k = 1, \ldots, s$:

$$E_{i_1,\ldots,i_s}(r) = \frac{1}{s\mu_2 + rq \sum_{k=1}^{s} 1[i_k \geq 2]} \left(1 + \mu_2 \sum_{k=1}^{s} E_{i_1,\ldots,i_{k-1},i_k-1,i_{k+1},\ldots,i_s}(r) \right.$$
$$\left. + rq \sum_{k=1}^{s} \sum_{n=1}^{i_k-1} \psi_{i_k-1,n} E_{i_1,\ldots,i_{k-1},i_k-n,i_{k+1},\ldots,i_s,n}(r) \right).$$

We have $E^p_0(r) = 1/\mu_1$ and we define recursively for $s \geq 1$

$$E^p_{1'_s}(r) = \frac{1}{\mu_1 + (s-1)\mu_2} \left(1 + \mu_1 E_{1'_s}(r) + (s-1)\mu_2 E^p_{1'_{s-1}}(r) \right)$$
$$= \frac{1}{\mu_1 + (s-1)\mu_2} \left(1 + \frac{\mu_1}{\mu_2} \sum_{k=1}^{s-1} \frac{1}{k} + (s-1)\mu_2 E^p_{1'_{s-1}}(r) \right),$$

where we have used (11) in the last equality. Finally, we define recursively, assuming $i_k \geq 1$ for $k = 2, \ldots, s$:

$$E^p_{i_1,\ldots,i_s}(r) = \frac{1}{\mu_1 + (s-1)\mu_2 + rq1[i_1 \geq 1] + rq \sum_{k=2}^{s} 1[i_k \geq 2]} \left(1 \right.$$
$$\left. + \mu_1 E_{i_1,\ldots,i_s}(r) + rq \sum_{n=1}^{i_1} \phi_{i_1,n} E^p_{i_1-n,i_2,\ldots,i_s,n}(r) \right.$$

$$+ \mu_2 \sum_{k=2}^{s} E^p_{i_1,\ldots,i_{k-1},i_k-1,i_{k+1},\ldots,i_s}(r)$$
$$+ rq \sum_{k=2}^{s} \sum_{n=1}^{i_k-1} \psi_{i_k-1,n} E^p_{i_1,\ldots,i_{k-1},i_k-n,i_{k+1},\ldots,i_s,n}(r) \Bigg).$$

The expectation $E[J(r)]$ can now be computed as:

$$E[J(r)] = \sum_{k=0}^{m} p_k E^p_k(r).$$

Note that when $r \to \infty$, children spawned by a parent job get immediately distributed amongst empty servers. Therefore, as $r \to \infty$, we get

$$E[J(r)] \to \sum_{k=0}^{m} p_k E^p_{0,1'_k}(r). \tag{12}$$

5 Numerical experiments

In this section we perform numerical experiments to compare the performance of several stealing strategies. Due to the lack of space, we present only a subset of the experiments performed. The main conclusions in these additional experiments (e.g., different μ_2 values) are in agreement with the results presented. Define Ψ as the matrix where $[\Psi]_{i,j} = \psi_{i,j}$ and define Φ similarly. Note that a strategy is fully characterized by Ψ and Φ. The strategies considered are as follows:

1. **Steal one:** The strategy of always stealing one child job, that is $\phi_{i,1} = \psi_{i,1} = 1$ for every i.
2. **Steal half:** The strategy of always stealing half of the pending child jobs. If n, the number of pending child jobs, is uneven, there is a fifty percent chance that $\lfloor n/2 \rfloor$ child jobs get stolen and $\lceil n/2 \rceil$ jobs otherwise;
3. **Steal all:** The strategy of stealing all of the pending child jobs, that is $\phi_{i,i} = \psi_{i,i} = 1$ for every i.

Note that these strategies do not rely on any knowledge on the (mean) job sizes or system load.

We compare the mean response time for these strategies with the optimal monotone deterministic strategy. A strategy is called deterministic if for every $i \leq m$ there exists a $j \leq i$ such that $\phi_{i,j} = 1$ and a $k \leq i$ such that $\psi_{i,k} = 1$. It is called monotone deterministic (MD) if in addition having $\psi_{i,j} = 1$ and $\psi_{i',j'} = 1$ with $i < i'$ implies that $j \leq j'$ for all i, j, i', j' and the same holds for Φ. Experiments not included in the paper suggest that the optimal strategy, that is, the optimal Ψ and Φ matrices, corresponds to an MD strategy. The optimal MD strategy is determined using brute-force and its mean response time is denoted as $T_{MD}(r)$. Let $\mathbf{p} = [p_0, p_1, \ldots, p_m]$.

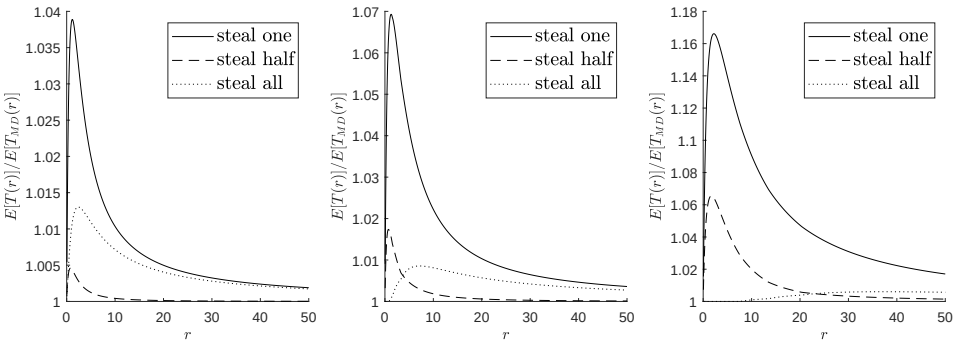

Fig. 1: Example 1 with $\rho = 0.15$ (left), $\rho = 0.5$ (mid) and $\rho = 0.85$ (right).

Example 1. In Figure 1 we examine the effect of increasing the steal rate on how well the three strategies perform compared to the optimal MD strategy. More specifically, we plot the mean response time $E[T(r)]$ of our three policies normalized by the mean response time $E[T_{MD}(r)]$ of the optimal MD policy. We do this for $\rho \in \{0.15, 0.5, 0.85\}, \mu_1 = 1, \mu_2 = 2, \mathbf{p} = 1_5'/5$ and $r \in [0.05, 50]$. We note that there exists no universal best strategy. The strategy of stealing one job performs the worst. This is due to the fact that relatively very little work of the pending jobs is transferred. When $\mu_2 < \mu_1$, examples can be constructed where the strategy of stealing a single child outperforms the others. For moderately high values of r or for low loads the strategy where half of the child jobs get stolen is close to the optimal MD strategy. This is intuitively clear as in such case there is a small chance that there are pending parent jobs in a queue, so stealing half of the child jobs more or less balances the work. In fact, it seems that as r becomes large enough the optimal strategy for systems where $\mu_1 \leq \mu_2$ is stealing $\lfloor i/2 \rfloor + 1$ out of i children. For low values of r the strategy of stealing all child jobs performs well, as there is a fair chance that there are pending parents in the queue and it can take a long time until the server is probed again.

For $\rho = 0.85$ the matrices Ψ, Φ of the optimal MD strategy change as follows: for low values of r the best strategy is the one of stealing all jobs, that is Ψ and Φ are identity matrices of size $m-1$ and m respectively. Then, at approximately $r = 7.6$, $\psi_{3,2}$ becomes one. Around $r = 13.5$, the $\phi_{4,3}$ becomes one and finally $\phi_{3,2} = 1$ around $r = 20.35$. For $\rho = 0.5$ we see a similar evolution: for low values of r the best strategy is stealing all child jobs. Then, at approximately $r = 0.85$, $\psi_{3,2}$ becomes one. Around $r = 1.55$, the $\phi_{4,3}$ becomes one and finally $\phi_{3,2} = 1$ around $r = 3.35$.

The number of MD strategies grows quickly in function of m (in fact one can prove that for a given m there exist $C(m)C(m+1)$ such strategies, where $C(k)$ denotes the k-th Catalan number). This implies that it can take a long time to determine the optimal MD strategy for systems with larger m values. We therefore introduce a smaller family of strategies and compare our three

strategies with the optimal strategy in this smaller family to limit the brute-force search. We call a strategy bounded monotone deterministic (BMD) if it is monotone deterministic and $\psi_{i,j} = 1$ implies $\psi_{i+1,j} = 1$ or $\psi_{i+1,j+1} = 1$ for every i and the same holds for Φ. Note that there are $2^{m-2}2^{m-1} = 2^{2m-3}$ BMD strategies for a given $m \geq 2$. The optimal BMD strategy is determined using brute-force and we denote its mean response time by $T_{BMD}(r)$. The mean response time of the optimal BMD strategy may exceed that of the optimal MD strategy as indicated in the next example.

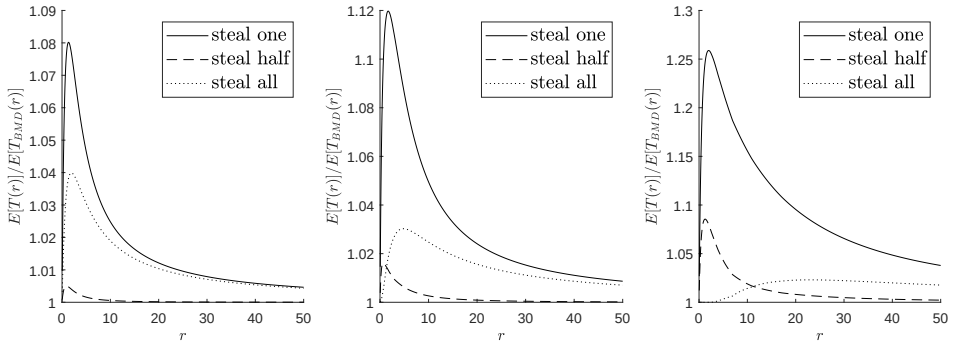

Fig. 2: Example 2 with $\rho = 0.15$ (left), $\rho = 0.5$ (mid) and $\rho = 0.85$ (right).

Example 2. In Figure 2 we examine the effect of increasing the steal rate on how well the three strategies perform compared to the optimal BMD strategy when $m = 6$ instead of $m = 4$ as in the previous example. We do this for $\rho \in \{0.15, 0.5, 0.85\}, \mu_1 = 1, \mu_2 = 2, \mathbf{p} = 1_7'/7$ and $r \in [0.05, 50]$. It is clear that the main insights are similar as in the $m = 4$ case, except that more substantial gains can be achieved by optimizing Ψ and Φ. We also performed some experiments to compare the performance of the optimal MD and BMD strategies and noted that for $r \in [6.9, 7.4]$, the optimal MD strategy has $\psi_{3,2} = 1$ and $\psi_{4,4} = 1$, which is not BMD. The reduction in the mean response time was however very limited.

Example 3. In Figure 3 we illustrate the effect of increasing the load ρ on the mean response time. We do this for $\rho \in [0.05, 0.95], \mu_1 = 1, \mu_2 = 2, m = 4, \mathbf{p} = 1_5'/5$ and $r \in \{0.1, 1, 10\}$. These result confirm that stealing all is best when the load is sufficiently high, while stealing half of the child jobs is good for systems with a limited load.

6 Conclusions and future work

We introduced a model for randomized work stealing in multithreaded computations in large systems, where parent jobs spawn child jobs and where any number

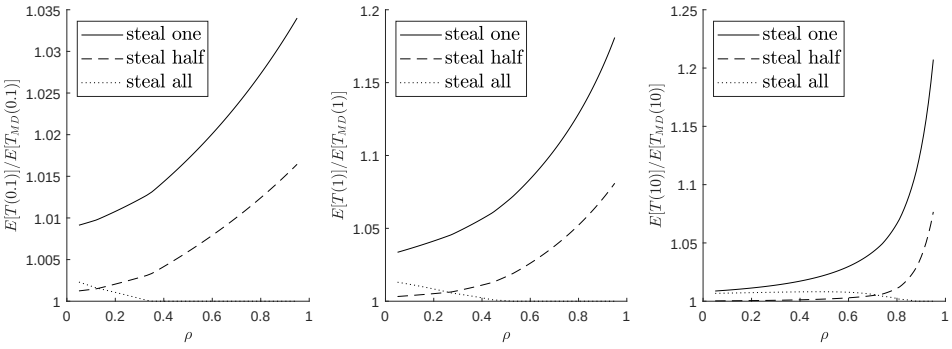

Fig. 3: Example 3 with $r = 0.1$ (left), $r = 1$ (mid) and $r = 10$ (right).

of existing child jobs can be stolen from a queue per probe. We defined a QBD Markov chain that approximates the behaviour of the system when the number of servers tends to infinity. We showed the existence and uniqueness of a stationary distribution for this QBD, provided formulas for the waiting and service times and provided a practical way of calculating expected service times. These are the main technical contributions of the paper. Using numerical experiments we examined the effect of changing the load ρ and the steal rate r. We concluded that the stealing policy where the half of child jobs gets stolen every time is in general a good stealing policy for higher values of r, while the strategy of stealing all children performs best for low values of r. We concluded further that stealing only one child performs the worst in most of the cases. Finally, using simulation, we validated the accuracy of the QBD model.

Possible generalizations include stealing multiple parent jobs (up to some finite amount) per probe and systems where offspring of a job can spawn further offspring (multigenerational multithreading). One can also attempt to relax the exponential service time requirements for child and/or parent jobs. This may be challenging as this complicates several aspects of the model such as determining the rates $\lambda_{c,j}(r)$.

References

[1] Bini, D., Meini, B., Steffé, S., Van Houdt, B.: Structured Markov chains solver: software tools. In: Proceeding from the 2006 workshop on Tools for solving structured Markov chains. pp. 1–14 (2006)

[2] Blumofe, R., Leiserson, C.: Scheduling multithreaded computations by work stealing. Journal of the ACM (JACM) **46**(5), 720–748 (1999)

[3] Blumofe, R., Joerg, C., Kuszmaul, B., Leiserson, C., Randall, K., Zhou, Y.: Cilk: An efficient multithreaded runtime system. Journal of parallel and distributed computing **37**(1), 55–69 (1996)

[4] Bramson, M., Lu, Y., Prabhakar, B.: Randomized load balancing with general service time distributions. In: ACM SIGMETRICS 2010. pp. 275–286 (2010). https://doi.org/10.1145/1811039.1811071, http://doi.acm.org/10.1145/1811039.1811071

[5] Eager, D., Lazowska, E., Zahorjan, J.: A comparison of receiver-initiated and sender-initiated adaptive load sharing,. Performance Evaluation **6**(1), 53–68 (1986)

[6] Frigo, M., Leiserson, C.E., Randall, K.H.: The implementation of the cilk-5 multithreaded language. In: In Proceedings of the SIGPLAN '98 Conference on Program Language Design and Implementation. pp. 212–223 (1998)

[7] Gast, N.: Expected values estimated via mean-field approximation are 1/n-accurate. Proceedings of the ACM on Measurement and Analysis of Computing Systems **1**(1), 17 (2017)

[8] Gast, N., Gaujal, B.: A mean field model of work stealing in large-scale systems. ACM SIGMETRICS Performance Evaluation Review **38**(1), 13–24 (2010)

[9] Gautier, T., Besseron, X., Pigeon, L.: Kaapi: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In: Proceedings of the 2007 international workshop on Parallel symbolic computation. pp. 15–23 (2007)

[10] Horváth, G., Van Houdt, B., Telek, M.: Commuting matrices in the queue length and sojourn time analysis of map/map/1 queues. Stochastic Models **30**(4), 554–575 (2014)

[11] Latouche, G., Ramaswami, V.: Introduction to matrix analytic methods in stochastic modeling, vol. 5. SIAM (1999)

[12] Lea, D.: A java fork/join framework. In: Proceedings of the ACM 2000 Conference on Java Grande. p. 36–43. JAVA '00, Association for Computing Machinery, New York, NY, USA (2000), https://doi.org/10.1145/337449.337465

[13] Leijen, D., Schulte, W., Burckhardt, S.: The design of a task parallel library. In: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications. p. 227–242. OOPSLA '09, Association for Computing Machinery, New York, NY, USA (2009), https://doi.org/10.1145/1640089.1640106

[14] Minnebo, W., Hellemans, T., Van Houdt, B.: On a class of push and pull strategies with single migrations and limited probe rate. Performance Evaluation **113**, 42–67 (2017)
[15] Minnebo, W., Van Houdt, B.: A fair comparison of pull and push strategies in large distributed networks. IEEE/ACM Transactions on Networking (TON) **22**(3), 996–1006 (2014)
[16] Mirchandaney, R., Towsley, D., Stankovic, J.: Adaptive load sharing in heterogeneous distributed systems. Journal of parallel and distributed computing **9**(4), 331–346 (1990)
[17] Neuts, M.: Matrix-geometric solutions in stochastic models: an algorithmic approach. John Hopkins University Press (1981)
[18] Ozawa, T.: Sojourn time distributions in the queue defined by a general QBD process. Queueing Systems and its Applications **53**(4), 203–211 (2006)
[19] Robison, A., Voss, M., Kukanov, A.: Optimization via reflection on work stealing in tbb. In: 2008 IEEE International Symposium on Parallel and Distributed Processing. pp. 1–8. IEEE (2008)
[20] Sonenberg, B., Kielanski, G., Van Houdt, B.: Performance analysis of work stealing in large scale multithreaded computing. To appear in ACM ToMPECS (2021)
[21] Spilbeeck, I.V., Houdt, B.V.: Performance of rate-based pull and push strategies in heterogeneous networks. Performance Evaluation **91**, 2–15 (2015)
[22] Squillante, M., Nelson, R.: Analysis of task migration in shared-memory multiprocessor scheduling. SIGMETRICS Perform. Eval. Rev. **19**(1), 143–155 (1991), http://doi.acm.org/10.1145/107972.107987
[23] Van Houdt, B.: Randomized work stealing versus sharing in large-scale systems with non-exponential job sizes. IEEE/ACM Transactions on Networking **27**, 2137–2149 (2019)
[24] Wirth, N.: Tasks versus threads: An alternative multiprocessing paradigm. Software - Concepts and Tools **17**, 6–12 (01 1996)

A Model validation

Based on numerical experiments in the Section 5, we see that stealing all or half of the children are good stealing policies: stealing all works best for low values of r, while stealing half of the children works well for higher values. Therefore, we validate the mean field model for the policy of stealing all or half of the children. We always start the simulations from an empty system and simulate the behaviour for $T = 10^5$ with a warm up period of 33% of T.

In Figure 4 we focus on the case where all children are stolen. The 95% confidence intervals were computed based on 5 runs with $N = 500$ servers, $m = 4$, $\mu_1 = 1, \mu_2 = 2, \rho = 0.75$, $\mathbf{p} = (1, 1, 1, 1, 1)/5$ and $r \in \{1, 5\}$. We see that there is an excellent match between the simulated waiting and service times and those of the QBD model (calculated using Section 4).

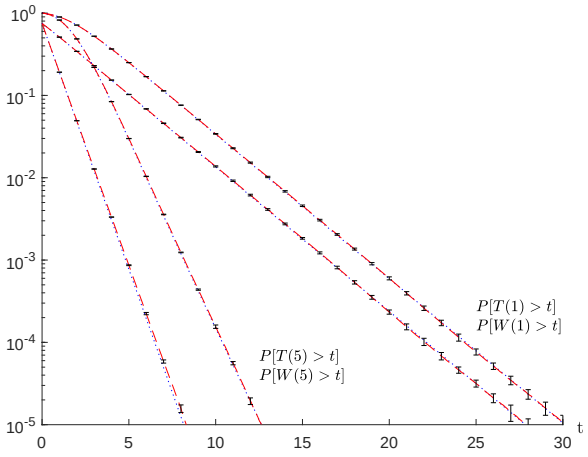

Fig. 4: Waiting and response times from the QBD (blue dots) and simulations (red dashed line) with confidence intervals for 5 runs.

In Table 2 we compare the relative error of the simulated mean response time, based on 20 runs, to the one obtained from Section 4. We do this for $\mu_1 = 1, \mu_2 = 2$, $\mathbf{p} = (1,1,1,1,1)/5$, $\rho \in \{0.75, 0.85\}$, $r \in \{1, 10\}$ and $N \in \{250, 500, 1000, 2000, 4000\}$.
The relative error in all cases is below 1.5% and tends to increase with the steal rate r. Further, the relative error seems roughly to halve when doubling N, which is in agreement with the results in [7].

Next we validate the model for the strategy of stealing half of the children using the same simulation settings. In Figure 5, we see that there is an excellent match between the simulated waiting and service times and those of the QBD model. Similarly to Table 2, we see in Table 3 that the relative error is below 1.5% in all cases, tends to increase with the steal rate r and seems about halved when doubling N.

Table 2: Relative error of simulation results for $E[T(r)]$, based on 20 runs

	$\rho = 0.75$		$\rho = 0.85$	
N	sim. ± conf.	rel.err.%	sim. ± conf.	rel.err.%
$r = 1$				
250	3.7650 ± 1.08e-02	0.2986	5.5121 ± 3.08e-02	0.3386
500	3.7588 ± 6.98e-03	0.1334	5.5053 ± 1.62e-02	0.2157
1000	3.7568 ± 6.16e-03	0.0818	5.4980 ± 1.60e-02	0.0821
2000	3.7548 ± 3.28e-03	0.0283	5.4945 ± 8.76e-03	0.0197
4000	3.7541 ± 2.53e-03	0.0091	5.4953 ± 6.96e-03	0.0344
QBD	3.7537		5.4935	
$r = 10$				
250	1.7766 ± 2.11e-03	0.7247	2.1371 ± 6.32e-03	1.2816
500	1.7701 ± 1.53e-03	0.3553	2.1232 ± 3.96e-03	0.6249
1000	1.7671 ± 8.21e-04	0.1894	2.1165 ± 2.50e-03	0.3090
2000	1.7655 ± 7.12e-04	0.0957	2.1131 ± 1.88e-03	0.1454
4000	1.7646 ± 7.18e-04	0.0437	2.1119 ± 8.00e-04	0.0878
QBD	1.7638		2.1100	

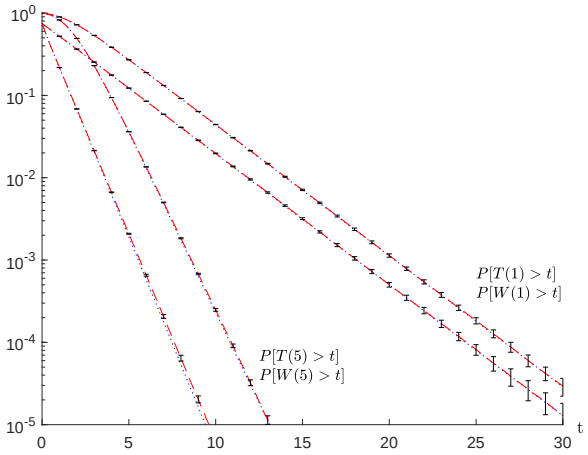

Fig. 5: Waiting and response times from the QBD (blue dots) and simulations (red dashed line) with confidence intervals for 5 runs.

Table 3: Relative error of simulation results for $E[T(r)]$, based on 20 runs

	$\rho = 0.75$		$\rho = 0.85$	
N	sim. ± conf.	rel.err.%	sim. ± conf.	rel.err.%
$r = 1$				
250	3.9305 ± 1.45e-02	0.2392	5.8435 ± 2.91e-02	0.2830
500	3.9261 ± 1.26e-02	0.1271	5.8331 ± 1.68e-02	0.1045
1000	3.9231 ± 5.55e-03	0.0506	5.8288 ± 1.04e-02	0.0307
2000	3.9225 ± 4.63e-03	0.0353	5.8281 ± 1.33e-02	0.0187
4000	3.9219 ± 2.71e-03	0.0200	5.8279 ± 9.34e-03	0.0153
QBD	3.9211		5.8270	
$r = 10$				
250	1.7822 ± 2.34e-03	0.7748	2.1782 ± 5.92e-03	1.3017
500	1.7752 ± 2.00e-03	0.3790	2.1642 ± 3.21e-03	0.6506
1000	1.7720 ± 1.25e-03	0.1965	2.1576 ± 2.82e-03	0.3437
2000	1.7703 ± 8.84e-04	0.1007	2.1537 ± 1.82e-03	0.1623
4000	1.7695 ± 3.83e-04	0.0567	2.1520 ± 1.65e-03	0.0832
QBD	1.7685		2.1502	