# Performance comparison of aggressive push and traditional pull strategies in large distributed systems

Benny Van Houdt
Department of Mathematics and Computer Science
University of Antwerp - IBBT
Middelheimlaan 1, B-2020 Antwerp, Belgium
Email:benny.vanhoudt@ua.ac.be

***Abstract*—Distributed systems benefit substantially from the ability to exchange jobs between temporarily heavily and lightly loaded nodes. Depending on whether the lightly or heavily loaded nodes initiate the job exchange, such strategies are termed pull or push strategies. In this paper we compare the performance of an aggressive push strategy with the more traditional pull strategies in large distributed systems using mean field models. We consider homogeneous systems, systems with fast and slow servers as well as master-worker setups.**

**We show that even at high loads the aggressive push strategy can outperform traditional pull strategies in a homogeneous system (at the expense of increased network traffic), while the pull strategies are superior in a master-worker setup where the proportion of masters is low. We also indicate that the choice of the best strategy is rather insensitive to the variation in the job processing time and that the queue length distribution under the aggressive push strategy decays faster than geometric.**

## I. Introduction

The performance of large distributed systems is heavily influenced by the strategy they use to distribute the overall work among the different processors/nodes. In this regard, the two main adaptive load sharing mechanisms identified are the pull and push strategies [3], [13]. Under a pull strategy idle/underloaded nodes try to pull/take a job from other nodes with pending jobs, as such the pull strategy is also known as work stealing. Push strategies work in the opposite manner: nodes with pending jobs attempt to push/assign some of their jobs to idle/underloaded nodes.

Both pull and push strategies can be implemented in a centralized or distributed manner. In the centralized setup, requests to pull/push a job are sent to a central dispatcher that has a complete (or partial) overview of the network, as such requests are typically successful. In a distributed setup, a node selects another node at random hoping that it can perform a pull/push operation with this node. Push strategies are generally considered more suitable for centralized systems, whereas pull solutions have been implemented in a number of distributed libraries (e.g., Cilk).

The performance of both push and pull strategies has been studied by various authors. An initial celebrated comparison for a homogeneous distributed system was presented in [3], showing that pull strategies are more effective under high loads, while push strategies are superior under low to medium loads. Similar observations were made for heterogeneous systems in [10], where two types of nodes where studied using decoupling assumptions as in [3] as well as matrix analytic methods [7]. A similar approach to study the influence of task migrations in shared-memory multi-processor systems was presented in [14]. In all of these papers, a pull operation is initiated whenever the number of jobs drops below some threshold $T \geq 1$, while push operations are initiated by new arrivals only, i.e., if the number of jobs in the queue is at least $T$ upon arrival of a new job. When job transfer and signaling delays are assumed negligible setting $T = 1$ is optimal [10], [9].

In this paper we will introduce a more aggressive class of push strategies that attempt to push *all* the queued jobs at regular time instants and we compare its performance with the more traditional pull strategies (with $T = 1$) using mean field models. Mean field models are a provable approach to obtain exact results for infinitely large distributed systems (at time $t$) [6], [1], [5] and their usefulness in the analysis of large distributed systems was already demonstrated in [11], [4], that analyzed a set of distributed pull strategies where idle nodes steal one or half of the pending jobs, and [15] that considers a number of centralized push strategies where each node consists of a set of $C$ processors and local jobs are pushed as soon as all the processors in a node are occupied.

Using our models, the following insights on the effectiveness of the aggressive push strategy are obtained:

1) In a homogeneous system, the aggressive push strategy outperforms the traditional pull strategy, at the expense of increased network traffic. The queue length distribution under the aggressive push strategy decays *faster* than geometric, as opposed to the traditional strategies in [3], [10], [11] that have a geometric decay[1].
2) The relative performance of the different strategies is rather insensitive to the job length distribution.
3) When the nodes have different processing speeds, a good strategy exists in avoiding pull (push) operations from (to) nodes that are faster (slower). Ignoring the different processing speeds however causes only a minor performance degradation.
4) In a master-worker setup (i.e., when all the job arrivals occur in some of the nodes only, called the masters), the traditional pull strategies tend to perform better as

[1]Other push strategies with faster than geometric decay were introduced and analyzed in [12].

the proportion of masters decreases and are superior to the aggressive push strategy, while requiring a smaller request rate.

The paper is structured as follows. Section II introduces the different push and pull strategies analyzed in this paper and discusses their relation with existing strategies, while Section III discusses their corresponding mean field models. The computational aspects and convergence results of these models are treated in Section IV and V, respectively. Finally, a large variety of numerical results and the insights they provide are found in Section VI.

## II. Pull and Push Strategies

We consider a discrete-time system consisting of $N_k$ type $k$ queues, for $1 \leq k \leq K$, where each queue consists of a single server and a buffer that is able to store $B_k$ jobs. We denote $\gamma_k = N_k/N$, with $N = \sum_k N_k$, as the fraction of queues of type $k$. As in [4], [11], [10], each type $k$ queue is subject to its own local Poisson arrival process with rate $\lambda_k$. Jobs processed by a type $k$ server require a discrete-time phase-type [7] distributed amount of work with parameters $(\beta_k, H_k)$, irrespective of where the job originated from.

All the strategies will make use of a stochastic matrix $P$ with entries $p_{ij}$, with $1 \leq i, j \leq K$. The use of $P$ is analogue to the matrix $F$ in [10] to support what the authors call biased probing. We start with the introduction of the aggressive push strategy, called *Push*, and the somewhat less aggressive *Push* $R$ strategy:

1) *Push:* Whenever a type $k$ queue has $q > 0$ packets buffered at the end of a time slot, it will, for *each* of its $q$ packets, select a type $k'$ queue at random with probability $p_{kk'}$ and will transmit a request packet to this queue to inquire whether it is idle and therefore able to process one of the waiting jobs. Only idle servers are allowed to accept a job, thus if an idle server receives multiple requests, it accepts one at random[2].
2) *Push $R$:* This strategy is the same as the push strategy above, but a single queue is only allowed to transmit $R$ request packets per time slot. The above push method therefore corresponds to the Push $\infty$ strategy. The parameter $R$ is assumed identical for all the queues.

The aim of this paper is to assess the relative performance of these new strategies to the following two more traditional pull strategies:

1) *Pull:* Whenever a type $k$ server is or becomes idle at the end of a slot, it attempts to *pull* (i.e., steal) a job from another server. To this end, it randomly selects a type $k'$ queue with probability $p_{kk'}$ and indicates that it is willing to process one job from this queue[3]. In other words a specific type $k'$ queue receives the request with probability $p_{kk'}/N_{k'}$. As all the idle servers in the system transmit such a request, a single server might receive multiple requests. When a server receives $s$ requests while it has $q > 0$ packets waiting in its buffer, it will accept $a = \min(s, q)$ requests by randomly assigning $a$ jobs to the $s$ requesting servers.
2) *Pull $R$:* This strategy is the same as the pull strategy above, but aims at reducing the number of requests packets that need to be transmitted at the end of each time slot. The parameter $R$ indicates that an idle server is only allowed to transmit a request if it became idle during one of the last $R$ time slots. The above pull method therefore corresponds to the Pull $\infty$ strategy.

The traditional pull strategies considered in [3], [10] are similar to the Pull ($R$) strategy above. The main difference is that the pull strategy of [3], [10] (with $T = 1$, which coincides with the *poll when idle* strategy in [8]) sends a request *only* when the server becomes idle, so at first it looks like the Pull $R = 1$ strategy, but an idle server will retry when the request fails (at most $L_p$ times). Thus, their pull strategy resembles a Pull $R$ with $R = L_p$ (which even for fairly small $L_p$ will be shown to perform close to the Pull strategy). However, sending the retries in the same slot as in [3], [10] should reduce the mean response time somewhat as there are no idle slots between retries. The push strategies of [3], [10] are quite different from our Push ($R$) strategy as a server only tries to push *newly arriving* jobs that find the server busy, hence, they are even less aggressive than our Push 1 strategy.

To compare the performance of the push and pull strategy we will introduce a mean field model in the next section. For this model, we will assume that request and job transfer times are equal to zero as in [3], meaning jobs are exchanged instantaneously. The assumption of having a zero transfer time for requests is quite realistic as transferring jobs typically requires considerably more time than sending a request. The models in [10], [9] do take an exponentially distributed job transfer time into account (while still assuming zero transfer time for the requests). Their results show that when increasing the transfer time, the delays clearly increase, while the performance differences between the various strategies become less significant (but remain similar).

## III. Mean field models

In this section we present a mean field model for each of the four strategies introduced in Section II. As explained in Section V, these models will capture the system behavior when the number of queues $N$ grows to infinity and will provide us with a good approximation when the number of queues $N$ is large. Notice, when $N$ grows to infinity, the ratios $\gamma_k = N_k/N$ remain identical.

We will observe the queue length and server phase of each queue at the end of each time slot (and the time since the server became idle in case of the Pull $R$ strategy). The events are assumed to occur in the following order: (a) we let the possible service completion or change of server phase take place, (b) we add the arrivals that occurred during the last time slot and (c) we transmit the necessary request packets and adjust the queue lengths accordingly. The state transition

[2]Notice, when a single queue transmits $q > 1$ requests it should clearly select $q$ different queues. In the mean field the probability of selecting the same queue twice is zero.

[3]Actually, when $k = k'$ the queue should avoid selecting itself, as this is clearly never useful. However, in the mean field model, the probability of having such an event is zero.

from time $t$ to $t+1$ in each of the models is therefore described by the product of three matrices, one for step (a), (b) and (c).

The changes that occur to the state of a specific type $k$ queue in step (a) and (b) clearly do not depend on the state of the other queues, as such these will be described by the matrices $S_k(t)$ and $A_k(t)$. The changes that occur in step (c) clearly do depend on the state of the other queues. More specifically they will depend on the fraction of queues that are in a specific state as explained below and the change of state of a type $k$ queue during step (c) is therefore described by a matrix $Q_k(\mu(t))$, where $\mu(t)$ will be the occupancy vector at time $t$.

### A. Pull strategy

The state of a single queue of type $k$ at time $t$ under the pull strategy consists of its queue length $b_t$, that takes values between 0 and $B_k+1$ with $B_k$ the size of the waiting room, and its current service phase $s_t$ that varies between 1 and $n_k$, where $n_k$ is the order of the phase-type representation $(\beta_k, H_k)$, that is, $\beta_k$ is a stochastic vector of length $n_k$ and $H_k$ a square substochastic matrix of size $n_k$. When the server is idle, the phase $s_t$ represents the initial phase of the next customer that will be served by the queue.

*Step (a):* During this step we take the possible service completions and changes of service phase into account. The matrix that captures these transitions is of size $n_k(B_k+2)$ and the probability of going from state $(b_t, s_t)$ to $(b'_t, s'_t)$ is defined as follows:

$$(S_k)_{(b_t,s_t),(b'_t,s'_t)} = \begin{cases} 1 & b_t = b'_t = 0, s_t = s'_t, \\ (H_k)_{s_t s'_t} & 0 < b_t = b'_t, \\ (1 - \sum_{u=1}^{n_k} (H_k)_{s_t u})\beta_{s'_t} & 0 < b_t = b'_t + 1, \\ 0 & otherwise, \end{cases} \quad (1)$$

where $(H_k)_{ij}$ represents entry $(i,j)$ of the matrix $H_k$ and $1 - \sum_{u=1}^{n_k}(H_k)_{s_t u}$ is therefore the probability of having a service completion given that the server was in phase $s_t$.

*Step (b):* This step adds the possible arrivals that occurred during the last time slot. Denote $\exp(-x)x^i/i!$ as $a_i^{(x)}$ and let $a_{i+}^{(x)} = \sum_{j \geq i} a_i^{(x)}$. The arrivals clearly do not affect the service phase which implies that the probability of going from state $(b'_t, s'_t)$ to $(b''_t, s''_t)$ is defined as follows:

$$(A_k)_{(b'_t,s'_t),(b''_t,s''_t)} = \begin{cases} a_i^{(\lambda_k)} & b'_t + i = b''_t \leq B_k, i \geq 0, s'_t = s''_t, \\ a_{(B_k+1-b'_t)+}^{(\lambda_k)} & b'_t \leq b''_t = B_k + 1, s'_t = s''_t, \\ 0 & otherwise, \end{cases} \quad (2)$$

because there are at most $B_k+1$ jobs present in a type $k$ queue.

*Step (c):* During this step the request packets are transmitted and the queue lengths get adjusted according to the outcome of these requests. With respect to the state of a type $k$ queue, we need to know (i) the probability that a request is accepted as this determines the probabilities out of the states of the form $(0, s''_t)$ and (ii) the probability that $r$ requests arrive to a type $k$ queue as this affects the transition probabilities of the states of the form $(b''_t, s''_t)$ with $b''_t > 1$ as the number of buffered packets equals $b''_t - 1$.

These probabilities will depend on the fraction of the queues that are in a specific state $(b''_t, s''_t)$, as such we denote $\mu_k(b''_t, s''_t)$ as the fraction of queues that are: (i) type $k$ and (ii) in state $(b''_t, s''_t)$ after step (b). Denote $\mu(t)$ as the stochastic vector of length $\sum_k n_k(B_k+2)$ that holds all the values $\mu_k(b''_t, s''_t)$. Further let $\alpha_k(b''_t) = \sum_{s''_t=1}^{n_k} \mu(b''_t, s''_t)$, i.e., it is the fraction of queues that are: (i) type $k$ and (ii) have a queue length of $b''_t$.

To simplify the derivation of these probabilities, we start with the special case of a homogeneous system, i.e., $K=1$, and drop the index $k$. If we consider a system consisting of $N$ queues, we find that $N\alpha(0)$ queues will send out a request packet and therefore any queue will receive $r$ request packets with probability

$$\binom{N\alpha(0)}{r} \frac{1}{N^r} \left(1 - \frac{1}{N}\right)^{N\alpha(0)-r} \to \frac{\alpha(0)^r}{r!} \exp(-\alpha(0)) \quad (3)$$

as $N$ goes to infinity. In other words the number of requests received has a Poisson distribution with parameter $\alpha(0)$. In a heterogeneous system, one similarly derives that the number of requests received by a type $k$ queue from the idle type $k'$ queues is Poisson distribution with parameter $\alpha_{k'}(0)p_{k'k}/\gamma_k$ as

$$\binom{N\alpha_{k'}(0)}{r} \left(\frac{p_{k'k}}{N_k}\right)^r \left(1 - \frac{p_{k'k}}{N_k}\right)^{N\alpha_{k'}(0)-r} \to \frac{(\alpha_{k'}(0)p_{k'k}/\gamma_k)^r}{r!} \exp(-\alpha_{k'}(0)p_{k'k}/\gamma_k)$$

and the total number of requests received by a type $k$ queue is therefore Poisson with parameter

$$\lambda_{k,req}(\mu(t)) = \sum_{k'=1}^{K} \alpha_{k'}(0)p_{k'k}/\gamma_k.$$

We will simply refer to this rate as $\lambda_{k,req}$.

This observation allows us to specify the probability of going from state $(b''_t, s''_t)$ to state $(b_{t+1}, s_{t+1})$ during step (c) for $b''_t > 0$ as follows:

$$(Q_k(\mu(t)))_{(b''_t,s''_t),(b_{t+1},s_{t+1})} = \begin{cases} a_i^{(\lambda_{k,req})} & 1 < b_{t+1} = b''_t - i, i \geq 0, s''_t = s_{t+1}, \\ a_{(b''_t-1)+}^{(\lambda_{k,req})} & 1 = b_{t+1} \leq b''_t, s''_t = s_{t+1}, \\ 0 & otherwise, \end{cases} \quad (4)$$

where the dependency on the vector $\mu(t)$ is via the request rate $\lambda_{k,req}$, which depends on the values $\mu_{k'}(0, s''_t)$ for $k' = 1, \ldots, K$ and $s''_t = 1, \ldots, n_{k'}$.

When $b''_t = 0$, we are considering a type $k$ queue that transmitted a request and we need to know the probability that this request got accepted to determine whether $b_{t+1}$ equals 0 (not accepted) or 1 (accepted). Notice, a request can also be rejected when there are $q > 0$ waiting jobs in case the number of arriving requests $r$ exceeds $q$. To simplify the derivation we start once more with the homogeneous system, i.e., $K=1$.

The probability that the request gets rejected depends on the number of jobs buffered in the selected queue, this number

equals $i-1$ with probability $\alpha(i)$, and on the number of *other* requests received by the selected queue, which has a Poisson distribution with parameter $\alpha(0)$ and is independent of the number of buffered jobs. As a result the probability that the request is *not* accepted equals

$$\alpha(0)+\alpha(1)+\sum_{i=2}^{B+1}\alpha(i)\sum_{r=i-1}^{\infty}a_r^{(\alpha(0))}\frac{r+1-(i-1)}{r+1},$$

which can be simplified to

$$\alpha(0)+\alpha(1)+\sum_{i=2}^{B+1}\alpha(i)\left(a_{(i-1)+}^{(\alpha_0)}-\frac{i-1}{\alpha(0)}a_{i+}^{(\alpha_0)}\right),$$

if $\alpha(0)>0$ and otherwise it equals $\alpha(1)$. In the heterogeneous case one similarly finds that a request send by a type $k$ queue is rejected with probability $q_{k,rej}(\mu(t))$

$$q_{k,rej}(\mu(t))=\sum_{k'=1}^{K}\frac{p_{kk'}}{\gamma_{k'}}\left(\alpha_{k'}(0)+\alpha_{k'}(1)+\sum_{i=2}^{B_k+1}\alpha_{k'}(i)\left(a_{(i-1)+}^{(\lambda_{k',req})}-\frac{i-1}{\lambda_{k',req}}a_{i+}^{(\lambda_{k',req})}\right)\right), \quad (5)$$

where the $k'$-th term equals $p_{kk'}\alpha_{k'}(1)/\gamma_{k'}$ whenever $\lambda_{k',req}$ equals zero. The transition probabilities for $b_t''=0$ therefore equal

$$(Q_k(\mu(t)))_{(0,s_t''),(b_{t+1},s_{t+1})}=\begin{cases}q_{k,rej}(\mu(t)) & b_t''=b_{t+1}=0, s_t''=s_{t+1},\\ 1-q_{k,rej}(\mu(t)) & b_t''=b_{t+1}-1=0, s_t''=s_{t+1},\\ 0 & otherwise.\end{cases} \quad (6)$$

### B. Pull $R$ strategy

The mean field model for the pull $R$ strategy is very similar to the one for the pull strategy, except that we need more state information when the server is idle. More specifically, we will replace each state of the form $(0,s_t)$ by a set of states $(-b_t,s_t)$ with $0\leq b_t\leq R$, where state $(-b_t,s_t)$ indicates that the server is idle, that $b_t$ more requests may be transmitted and the next job that is served will start service in phases $s_t$. This affects the matrices $S_k$, $A_k$ and $Q_k(\mu(t))$ in the following manner.

*Step (a):* When the server becomes idle we no longer enter a state of the form $(0,s_t')$, but instead enter state $(-R,s_t')$. In case the queue was in a state of the form $(b_t,s_t)$ with $b_t\leq 0$, we simply remain in the same state. In other words,

$$(S_k)_{(b_t,s_t),(b_t',s_t')}=\begin{cases}1 & b_t=b_t'\leq 0, s_t=s_t',\\ (H_k)_{s_ts_t'} & 0<b_t=b_t',\\ (1-\sum_{u=1}^{n_k}(H_k)_{s_tu})\beta_{s_t'} & 1<b_t=b_t'+1,\\ & or\ b_t=1, b_t'=-R,\\ 0 & otherwise.\end{cases} \quad (7)$$

*Step (b):* For $b_t'\geq 0$ all the transition probabilities remain as in Eqn. (2). Whenever the system state $(b_t',s_t')$ is such that $b_t'<0$, we remain in state $(b_t',s_t')$ with probability $a_0^{(\lambda_k)}$ or move to state $(i,s_t')$ with probability $a_i^{(\lambda_k)}$ for $0<i\leq B_k$ (and to state $(B_k+1,s_t')$ with probability $a_{(B_k+1)+}^{(\lambda_k)}$).

*Step (c):* We define $\mu_k(b_t'',s_t'')$ and $\alpha_k(b_t'')$ as in step (c) of Section III-A, except that $\alpha_k(0)$ no longer equals $\sum_{s_t''=1}^{n_k}\mu(0,s_t'')$, but equals $\sum_{s_t''=1}^{n_k}\sum_{b_t''=0}^{R}\mu(-b_t'',s_t'')$ instead. Additionally we define

$$\alpha_k(-)=\sum_{s_t''=1}^{n_k}\sum_{b_t''=1}^{R}\mu(-b_t'',s_t'')$$

such that the number of requests received by a type $k$ queue is Poisson with parameter

$$\lambda_{k,req}(\mu(t))=\sum_{k'=1}^{K}\alpha_{k'}(-)p_{k'k}/\gamma_k,$$

which we denote as $\lambda_{k,req}$ to ease the notation.

With this modified value for $\lambda_{k,req}$, we can now make use of Eqn. (4) to describe the transitions out of the states with $b_t''>0$. Further, with the new definition of $\alpha_k(0)$ Eqn. (5) still expresses the probability that a request from a type $k$ queue is rejected. Thus, when $b_t''\leq 0$ we now find

$$(Q_k(\mu(t)))_{(b_t'',s_t''),(b_{t+1},s_{t+1})}=\begin{cases}q_{k,rej}(\mu(t)) & b_{t+1}=b_t''+1\leq 0, s_t''=s_{t+1},\\ 1-q_{k,rej}(\mu(t)) & b_t''<0, b_{t+1}=1, s_t''=s_{t+1},\\ 1 & b_t''=b_{t+1}=0, s_t''=s_{t+1}\\ 0 & otherwise,\end{cases} \quad (8)$$

as the number of attempts to pull in a job is reduced by one after each failed attempt, while the queues with $b_t''=0$ do not send requests.

### C. Push $(R)$ strategy

The Push and Push $R$ strategy can be captured by means of a single mean field model where setting $R=\infty$ results in the Push strategy. As in the pull model of Section III-A, the state of a type $k$ queue is represented by $(b_t,s_t)$ where the queue length $b_t$ varies from 0 to $B_k+1$ and the service phase $s_t$ from 1 to $n_k$. Moreover, both step (a) and (b) are identical to the pull model, meaning the matrices $S_k$ and $A_k$ are given by Eqn. (1) and Eqn. (2), respectively. As such it suffices to determine the matrices $Q_k(\mu(t))$ of step (c).

*Step (c):* We define $\mu_k(b_t'',s_t'')$ and $\alpha_k(b_t'')$ as in step (c) of Section III-A. Additionally, we define

$$\alpha_k(+)=\sum_{i=1}^{R-1}i\alpha_k(i+1)+R\sum_{i=R}^{B_k}\alpha_k(i+1).$$

Analogue to the pull model one finds that the number of requests that are received by a queue is Poisson with parameter $\alpha(+)$ in the homogeneous case, while in a heterogeneous system a type $k$ queue receives a Poisson distributed amount of requests with parameter

$$\lambda_{k,req}(\mu(t))=\sum_{k'=1}^{K}\alpha_{k'}(+)p_{k'k}/\gamma_k,$$

which we denote as $\lambda_{k,req}$ for brevity. The probability that an idle type $k$ queue remains idle is therefore equal to $\exp(-\lambda_{k,req})$, otherwise it will accept one job and move from state $(0,s_t'')$ to $(1,s_t'')$.

When a type $k$ queue transmits $r$ request packets (because it was in a state of the form $(r+1, s_t'')$ after step (b) if $r < R$), we must determine the probability that $j$ of these requests are successful, for $j = 0$ to $r$. A request is successful with probability $1/(i+1)$ in case $i$ other queues transmit a request to the same queue and given that this queue is idle. Thus, in the homogeneous system, i.e., when $K = 1$, a request is successful with probability

$$q_{suc}(\mu(t)) = \alpha(0) \sum_{i=0}^{\infty} \frac{a_i^{(\alpha(+))}}{i+1},$$

which simplifies to $\alpha(0)(1-\exp(-\alpha(+)))/\alpha(+)$ if $\alpha(+) > 0$ and to $\alpha(0)$ otherwise. In this case the number of successful requests is binomially distributed with parameters $(r, q_{suc}(\mu(t)))$.

In the heterogeneous system the number of successful requests is still binomially distributed, but the success probability depends on $k$ and is determined by

$$q_{k,suc}(\mu(t)) = \sum_{k'=1}^{K} \frac{p_{kk'}}{\gamma_{k'}} \alpha_{k'}(0) \frac{1-\exp(-\lambda_{k,req})}{\lambda_{k,req}},$$

where the $k'$-th term equals $p_{kk'}\alpha_{k'}(0)/\gamma_{k'}$ whenever $\lambda_{k',req}$ equals zero. To ease the notation we refer to $q_{k,suc}(\mu(t))$ as $q_{k,suc}$.

In conclusion, the matrix $Q_k(\mu(t))$ can be written as

$$(Q_k(\mu(t)))_{(b_t'',s_t''),(b_{t+1},s_{t+1})} =$$
$$\begin{cases} \exp(-\lambda_{k,req}) & b_{t+1} = b_t'' = 0, s_t'' = s_{t+1}, \\ 1-\exp(-\lambda_{k,req}) & b_{t+1} = 1, b_t'' = 0, s_t'' = s_{t+1}, \\ \binom{r}{i} q_{k,suc}^i (1-q_{k,suc})^{r-i} & b_{t+1} = b_t'' - i \geq 1, s_t'' = \\ & s_{t+1}, r = \min(b_t''-1, R), \\ 0 & otherwise, \end{cases}$$

## IV. Performance measures

As some of the main performance measures can be computed more naturally from the system state after step (b), we let $\mu_k(t)$ denote the state at time $t$ after step (b). We define $\mu_k(0) = (\gamma_k, 0, \ldots, 0)$, meaning we start with an empty system at time $t = 0$. Next, the occupancy vector at time $t+1$ can be computed naturally as

$$\mu_k(t+1) = \mu_k(t) Q_k(\mu(t)) S_k A_k,$$

for $k = 1, \ldots, K$, where $\mu(t+1) = (\mu_1(t+1), \ldots, \mu_K(t+1))$. In order to approximate the steady state of the finite system with $N$ queues, we compute a fixed point by repeating this iteration until $||\mu(t+1) - \mu(t)||_\infty < \epsilon$, for some $\epsilon$ small (e.g., $10^{-12}$).

When the system is not highly loaded, this iterative scheme, that requires $O(\sum_{k=1}^K n_k^2 B_k^2)$ time per iteration, converges very quickly. However, when the system is highly loaded (meaning some queues are close to being unstable if the queues have infinite length), many thousands of iterations might be required, making the iteration slow. Therefore, we rely on the slightly modified iteration below

$$\mu_k(t+1) = \mu_k(t+1) Q_k(\mu(t)) S_k A_k,$$

where the first $\mu_k(t)$ is replaced by $\mu_k(t+1)$ on the right-hand side. This implies that we need to solve a linear system during each iteration, as opposed to a vector-matrix multiplication, which increases the time complexity per iteration to $O(\sum_{k=1}^K n_k^3 B_k^3)$. However, the number of iterations is substantially smaller and remains small even when the system is highly loaded, while both iterations were found to converge to the same solution. As such this iteration computes the performance measures of any of the systems discussed in this paper, irrespective of the load, in a matter of seconds.

Having obtained the fixed point $\mu$, with entries $\mu_k(b'', s'')$, for $k = 1, \ldots, K$, $b'' = 0, \ldots, B_k + 1$ and $s'' = 1, \ldots, n_k$, we can compute performance measures such as (a) the rate of pull/push requests, (b) the rate of job migrations (i.e., amount of network traffic) and (c) the mean response time (via Little's formula). Whenever we compute the mean delay, we ignore the $0.5$ slots that should be added as the Poisson arrivals occur uniformly within a slot. For instance, for the heterogeneous Pull model these are found as

(a) $$\sum_{k=1}^{K} \sum_{s''=1}^{n_k} \mu_k(0, s'')$$

(b) $$\sum_{k=1}^{K} \sum_{s''=1}^{n_k} \mu_k(0, s'') (Q_k(\mu))_{(0,s''),(1,s'')}$$

(c) $$\left( \sum_{k=1}^{K} \sum_{s''=1}^{n_k} \sum_{b''=1}^{B_k+1} b'' \mu_k(b'', s'') \right) / \left( \sum_k \lambda_k \gamma_k \right).$$

Similar expressions can be obtained for the other strategies.

## V. Proofs of convergence

Each of the four models introduced in Section III fits in the framework introduced in [1] for a general system of interacting objects. When the number of objects tends to infinity and under some mild conditions, such a system is shown to converge to its mean field [1]. In our case, the objects are of $K$ different classes, the length $\ell_k = n_k(B_k+1)$ occupancy vector at time $t$ of the type $k$ objects in the finite system with $N$ objects is denoted as $M_k^N(t)$ and the evolution of the system is described by some matrix $R_k^N(M^N(t))$, with $M^N(t) = (M_1^N(t), \ldots, M_K^N(t))$. The main convergence result in [1] is re-stated here, in a simplified form that suffices for the model at hand, as Theorem 1, which relies on the following condition.

**Condition 1** (Hypothesis **H** in [1])**.** *For all $i, j$, as $N \to \infty$, $[R_k^N(m)]_{ij}$ converges uniformly in $m \in \mathbb{R}^\ell$, with $\ell = \sum_k \ell_k$, to some $[R_k(m)]_{ij}$, which is a continuous function of $m$.*

**Theorem 1** (Theorem 4.1 in [1])**.** *Assume that the initial occupancy measure $M_k^N(0)$ converges almost surely to a deterministic limit $\mu_k(0)$. Define $\mu(t) = (\mu_1(t), \ldots, \mu_K(t))$ iteratively from its initial value $\mu(0) = (\mu_1(0), \ldots, \mu_K(0))$, for $t \geq 0$, as $\mu_k(t+1) = \mu_k(t) R_k(\mu(t))$. Then, for any fixed time $t$, almost surely, $\lim_{N\to\infty} M_k^N(t) = \mu_k(t)$.*

Therefore, to apply Theorem 1 we need to ensure the almost sure convergence of $M_k^N(0)$ to $\mu_k(0)$, as $N \to \infty$, as well as to verify that Condition 1 holds. For our models $R_k(m) =$

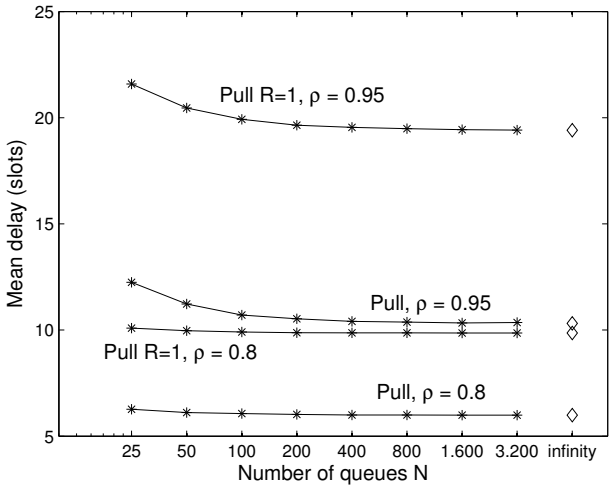


Figure 1. Convergence to the fixed point of the mean field for the Pull strategies

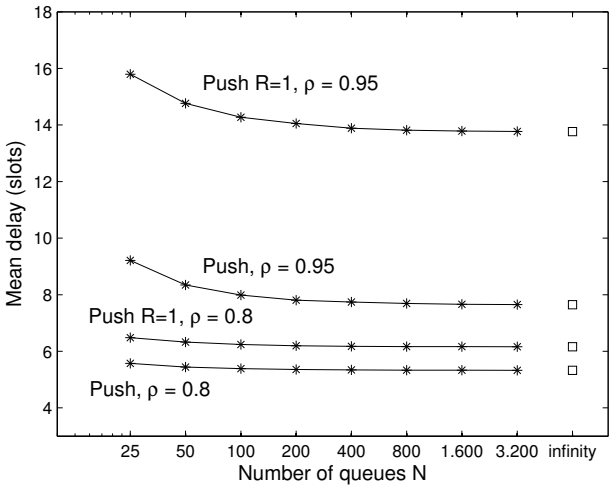


Figure 2. Convergence to the fixed point of the mean field for the Push strategies

$Q_k(m)S_kA_k$, for $k = 1, \ldots, K$. These matrices are clearly continuous in $m$ as the entries of $Q_k(m)$ are combinations of polynomials and exponentials in the entries of $m$ (i.e., they are even continuously differentiable in $m$). Although we have not discussed the transition matrices $R_k^N(m)$ in Section III, it is not hard to see that these converge uniformly to $R_k(m)$. For instance, for the homogeneous Pull model, this follows directly from the fact that the convergence in Eqn. (3) is uniform in $\alpha(0)$ for $\alpha(0) \in [0, 1]$.

To establish the almost sure convergence of $M_k^N(0)$ to $\mu_k(0)$, we first need to specify the type $k$ of each of the $N$ queues. We do this in a manner similar to [5], where we define $\lfloor N\gamma_k \rfloor = \lfloor N_k \rfloor$ queues of type $k$. The remaining $N - \sum_k \lfloor N_k \rfloor < K$ queues are allocated randomly with the probability of being a type $k$ queue proportional to the fractional part of $N\gamma_k - \lfloor N_k \rfloor$. Additionally, we assume that the system is empty at time zero, that is, $m_k(0) = (\gamma_k, 0, \ldots, 0)$ and $M_k^N(0) = (\lfloor N_k \rfloor + i, 0, \ldots, 0)/N$ given that $i$ of the remaining queues are of type $k$. As in [5], this guarantees the almost sure convergence of $M_k^N(0)$ to $\mu_k(0)$.

By applying Theorem 8 in [5], we can also show that the difference between the finite system and the mean field at time $t$ decreases according to $\sqrt{N}$ (for any $t$). These results however are only related to the convergence at time $t$, while we are interested in the limit as $t$ goes to infinity. To prove that the fixed point[4] is the limit of the steady state distributions of the finite systems, one additionally needs to prove that (i) the fixed point is a global attractor (i.e., the same fixed point is obtained irrespective of the state at time $t = 0$) and that the limits in $N$ and $t$ can be exchanged. Our numerical experiments seem to indicate that this is the case, but we do not have a formal proof (which is not uncommon when developing models of this type, see [11], [4]).

In order to demonstrate the convergence in $N$ of the steady state of the finite system towards the fixed point, we used time consuming simulations in Figures 1 and 2 for the mean delay of the Pull, Pull 1, Push and Push 1 strategy in a homogeneous system with a load of $0.8$ and $0.95$ (as well as for other cases not depicted here). Each of the simulation results was obtained from a single long simulation run, such that the width of the $95\%$ confidence intervals was well below $1\%$ of the mean delay using the batch means method. From these figures it is apparent that, for a fixed $N$, the mean field becomes less accurate as the load increases (this was also confirmed using additional experiments), but convergence to the mean field still occurs. Furthermore, the relative order of the strategies seems quite insensitive to the number of queues in the system. Finally, the mean field is always optimistic in comparison with the finite system, which is not all that surprising as the system becomes more deterministic as the number of queues increases.

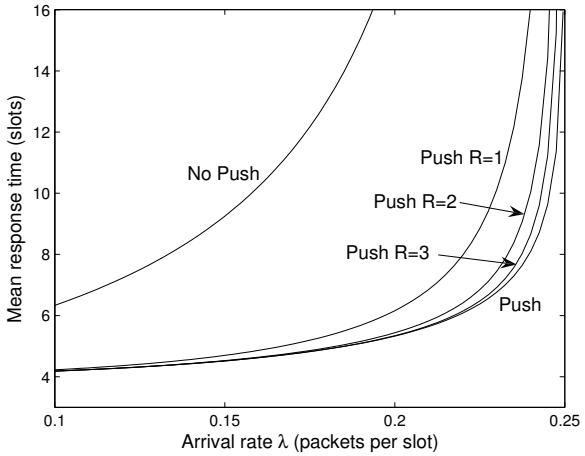


Figure 3. Mean packet delay as a function of the arrival rate $\lambda$ for the Push strategies

[4]We have at least one fixed point due to Brouwer's theorem as the set of stochastic vectors is convex and compact, while the entries of $R_k(m)$ are continuous in $m$

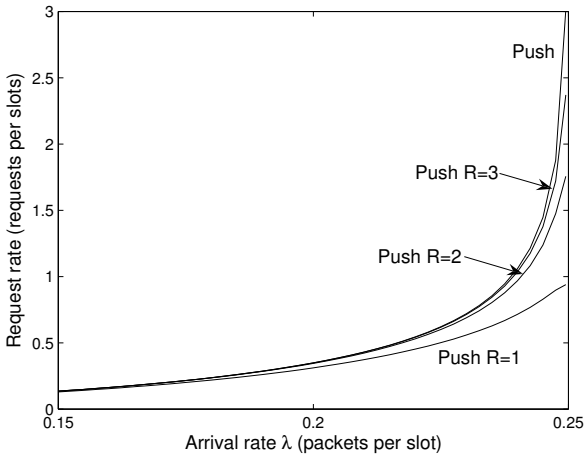


Figure 4. Request rate as a function of the arrival rate $\lambda$ for the Push strategies

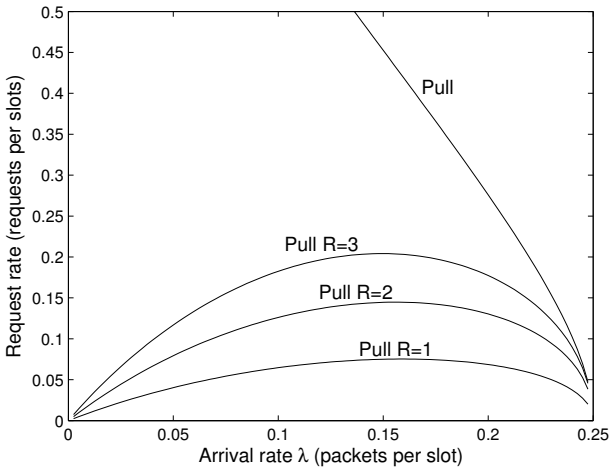


Figure 6. Request rate as a function of the arrival rate $\lambda$ for the pull strategies

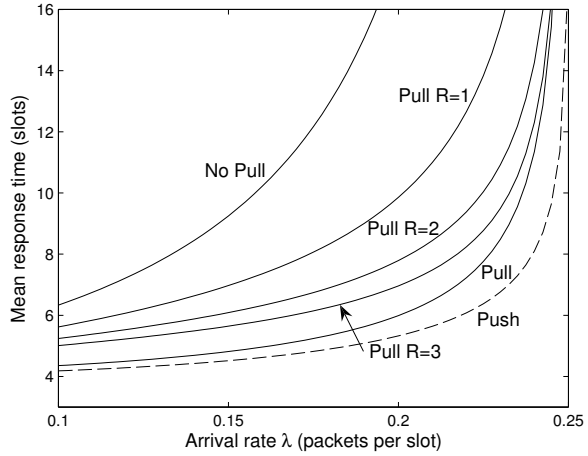


Figure 5. Mean packet delay as a function of the arrival rate $\lambda$ for the pull strategies

## VI. Numerical Results

### A. Homogeneous system

In the homogeneous system setup, all the queues are subject to the same arrival rate, while the processing time of a job is assumed to be geometric with a mean of $4$ slots. The arrival rate will be varied from $0$ to $0.25$, meaning the system load is between $0$ and $1$. The buffer size was set to $B = 60$, meaning the loss rate can be neglected.

We start by comparing the mean packet delay and request rate for the push strategies in Figures 3 and 4, respectively. As expected increasing $R$ reduces the delays, where moderate values of $R$ suffice, unless the system load is close to one. The signaling overhead for the Push strategy is substantial for high loads, but perhaps not as high as expected. Indeed, the request rate of the Push strategy corresponds to the mean queue length and even for loads as high as $99\%$ this value stays below 3. The Push $R$ strategy causes some reduction in the signaling overhead, but the reduction is only significant when the load approaches one.

Figures 5 and 6 compare the mean packet delay and request rate for the traditional pull strategies, respectively (where the results for the Push strategy are repeated for the purpose of comparison). Again, increasing $R$ reduces the mean delay at the expense of a higher request rate. More importantly, when we compare the Pull and Pull $R$ strategies, small $R$ values are able to approximate the mean delay of the Pull strategies rather well while significantly reducing the signaling overhead for a broad range of arrival rates $\lambda$. This result somewhat resembles the one in [3], where small $L_p$ values where shown to suffice as well (see Section II for its definition).

When comparing the Pull and Push strategies, we find that all perform well for low and medium loads. The Push strategy however outperforms the Pull strategy for all the arrival rates $\lambda$, there the gains become quite significant at high loads. Of course, this reduction in the mean packet delay comes at the expense of a higher request rate. The performance of the Pull strategy can clearly be further improved by allowing idle servers to send multiple requests in the same slot. However, selecting the proper number of request packets is hard for an idle server as it has no information to rely on. This is in contrast to the Push strategy where the number of waiting packets is known and used.

Figure 7 depicts the queue length distribution for various pull and push strategies for a system with a load of $0.95$. The tails of the distributions all appear to be geometric except for the Push strategy. This observation is in agreement with the results for the pull strategies in [11], for which the queue length distribution was proven to decay geometrically (for the simple models). The *faster* than geometric decay of the aggressive Push strategy can be understood intuitively as follows. The queues that belong to a distributed system that uses a push strategy have an effective service rate that is clearly larger than the speed of their server, as some jobs are pushed to other queues. This relative rate is clearly bounded when we apply the Push $R$ strategy (resulting in a geometric decay), but grows proportionally with the number of buffered jobs in case of the Push strategy. Such a proportional growth, as in a traditional M/G/$\infty$ queue, leads to a faster than geometric

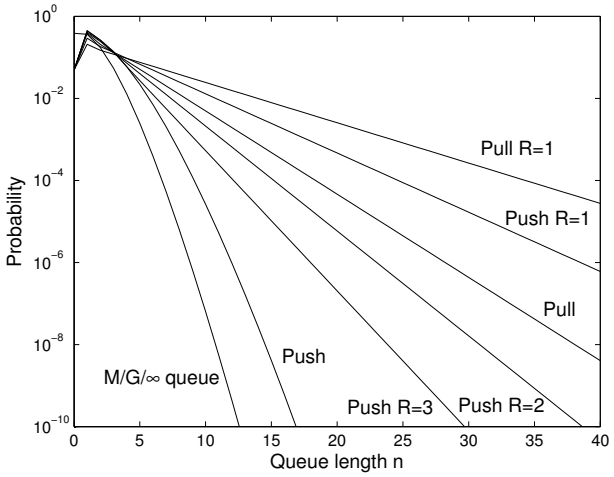


Figure 7. Queue length distribution for various Pull and Push strategies for $\rho = 0.95$

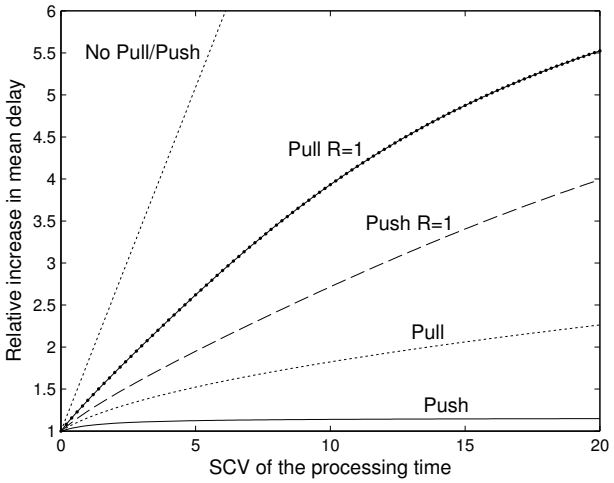


Figure 8. Mean delay as a function of the SCV divided by the mean delay when the job lengths are deterministic

decay, that is, the number of busy servers in an M/G/$\infty$ queue is Poisson distributed with parameter $\rho$ (its distribution is also depicted in Figure 7, for $\rho = 0.95$, showing a similar behavior). This fast decay also explains the short mean queue lengths even when the load is high.

### B. Impact of job length variability

In this section we study the impact of the variability of the processing time of a job. As in the previous section we assume a mean processing time of $4$ slots, but let the squared coefficient of variation (SCV) vary from $0$ to $20$, where an SCV of $3/4$ corresponds to the geometric distribution. To obtain a discrete-time phase-type distribution $(\beta, S)$ that matches the mean and SCV, we use the method of [2].

Figure 8 shows the mean delay as a function of the SCV divided by the mean delay when the job lengths are deterministic and this for the Pull, Pull $1$, Push and Push $1$ strategy. The arrival rate was fixed at $0.225$, meaning the system has a load of $0.9$ (similar results are obtained for other loads). We find that the Push strategy is the least sensitive to the job

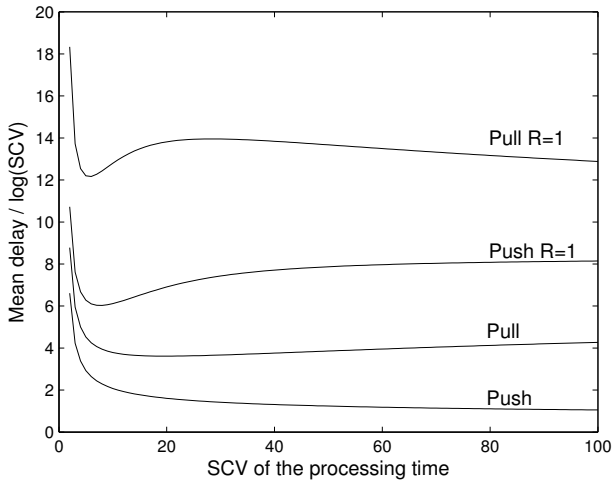


Figure 9. Mean delay as a function of the SCV divided by $\log_2 SCV$

length distribution, followed by the Pull, Push $1$ and Pull $1$ strategy. This implies that the relative order of the strategies is quite insensitive to the SCV and focusing on geometric service times is also quite representative for other distributions.

For the system without pulling or pushing (i.e., when we have a set of independent $M/PH/1$ queues) the mean delay equals

$$m + \frac{\lambda(SCV+1)m^2}{2(1-\lambda m)},$$

where $m = 4$ is the mean service time, i.e., the mean delay increases linearly with the $SCV$. For the pull and push strategies, the increase behaves more like a function of the logarithm of the SCV, see Figure 9 where we divided the mean delay as a function of the SCV by the logarithm of the SCV.

### C. Fast and slow servers

In this section we look at a heterogeneous setup and assume we have two types of queues: queues with fast (type 1) and slow (type 2) servers. We assume that a job processed by a fast server requires a mean service of 3 time slots, while the slow servers process the same job in an average time of $5$ slots. Given the results in the previous two sections we limit ourselves to geometric service times and only compare the Pull and Push strategy. We further assume uniform arrivals, meaning each queue (whether type 1 or 2) is still subject to Poisson arrivals with rate $\lambda$ and therefore the slow servers are more heavily loaded. The systems considered in [10] also include setups with fast and slow servers (called type-2 systems), but in their case the load in each queue is assumed identical.

We will consider three different selection strategies for sending the request packets, which results in the following combined strategies:

- Pull/Push Uniform: when a server transmits a request, it randomly selects its destination irrespective of whether it is slow or fast ($P = (0.5, 0.5; 0.5, 0.5)$)
- Pull Slow: a server only tries to pull jobs from slow servers, i.e., it randomly selects a slow server ($P = (0, 1; 0, 1)$)

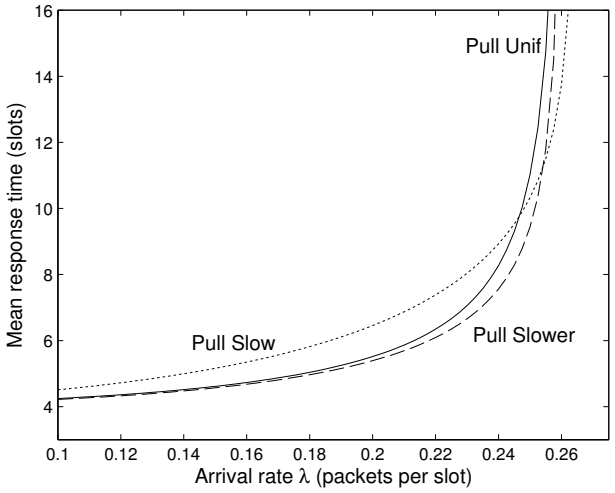


Figure 10. Mean delay as a function of the arrival rate $\lambda$ for the pull strategies in a fast/slow server setting

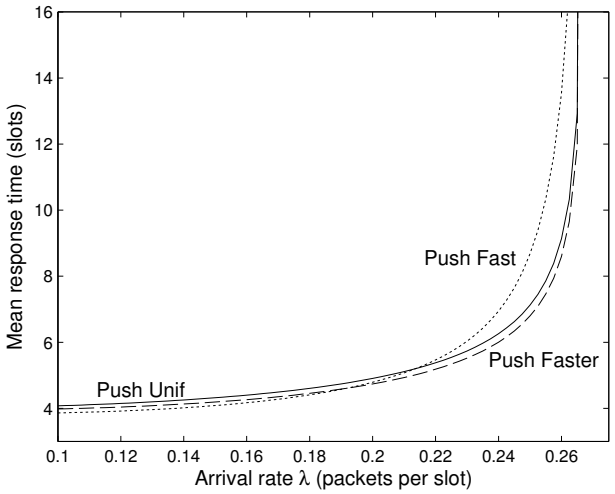


Figure 11. Mean delay as a function of the arrival rate $\lambda$ for the push strategies in a fast/slow server setting

- Push Fast: a server only tries to push jobs towards the fast servers, i.e., it randomly selects a fast server ($P = (1, 0; 1, 0)$)
- Pull Slower: a server is not allowed to send a request to a faster server ($P = (0.5, 0.5; 0, 1)$)
- Push Faster: a server is not allowed to send a request to a slower server ($P = (1, 0; 0.5, 0.5)$)

These strategies can be compared to the different types of biased probing considered in [10]. Note, we only consider strategies that are independent of the system parameters and do not attempt to optimize the entries of $P$ as such optimal values will clearly depend on the detailed statistics of the arrival and service processes.

In Figure 10 and 11 we compare the mean delay of the pull and push strategies, respectively.

*a) Pull Strategies:* A first observation is that the Pull Slower strategy outperforms the Pull Uniform for all loads, thus if we know the identities of the fast and slow servers, we can achieve some gains over the entire range of loads. The Pull Slow strategy on the other hand is less effective, unless the load is very high. The superiority of the Pull Slow strategy under high loads stems from its stability properties discussed next.

If we consider a system with $N$ queues of infinite length, where a fraction $\gamma_1$ of the servers is fast (with a mean service time of $m_1$) and $\gamma_2 = 1 - \gamma_1$ is slow (with a mean service time of $m_2$), one can show that the system is stable under the Pull Slow strategy as long as

$$\lambda < \frac{\gamma_1}{m_1} + \frac{\gamma_2}{m_2}. \tag{9}$$

To prove this it suffices to show that if the system becomes unstable, all the slow queues become saturated simultaneously. When the slow servers are saturated any request from a fast server is successful under the Pull Slow strategy. Thus, the highest allowable arrival rate corresponds to the average service rate of a server.

For the Pull Uniform and Pull Slower one can also argue that instability corresponds to saturated slow servers. However, instability occurs sooner as some of the requests send by the fast servers may be unsuccessful even when the slow servers are saturated. Thus, there is a positive probability that some of the fast servers are idle. When the slow servers are saturated the Pull Uniform and Pull Slower strategy coincide as they only differ in the selection strategy of the slow servers, but a slow server never transmits a request.

Determining the exact value for the instability point is very hard in this case and seems to depend on the service time distributions (and not merely on their means). Nonetheless, a tight lower bound that applies to any service time distribution can still be derived. For the example in Figure 10 this bound equals $9/35 \approx 0.2571$, while the average service rate is $4/15 \approx 0.2666$.

*b) Push Strategies:* Similar to the pull strategies the Push Uniform is always outperformed by the Push Faster, indicating that one can exploit the knowledge of the fast and slow servers. The Push Fast is however not the best strategy under high loads, because as opposed to the pull strategies, all the push strategies have the same stability characteristics (given by Eqn. (9)). This can be understood easily by showing that instability corresponds once more to saturated slow servers and when some of the servers are saturated all the idle fast queues will receive requests even if a request is only send to a fast server with a low positive probability.

For low and medium loads, the Push Fast strategy performs best and achieves a mean delay below 4 (which is the mean delay as $\lambda$ approaches zero), as some of the jobs that would require on average 5 time slots in a slow server are now served by a fast server with a mean of 3 slots only.

### D. Master-worker system

In this section we consider a master-worker setup as in [4], that is we consider a system with two types of queues where the type 1 queues, called masters, receive all the work, while the type 2 queues, called workers, receive no jobs. The objective is to compare the performance of the Pull and Push strategy only, given that a fraction $\gamma_1$ of the queues are

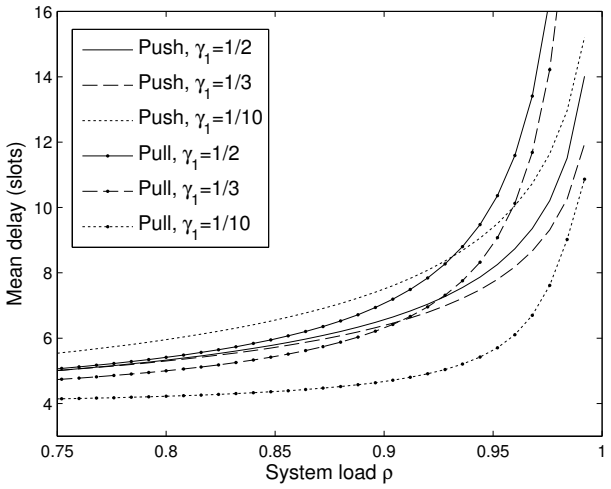


Figure 12. Mean delay as a function of the arrival rate $\lambda$ for the Pull and Push strategy in a master-worker setting

masters. Notice, in such a setting we cannot deploy the Pull $R$ strategy as a worker would become disabled if $R$ consecutive requests failed. The Push $R$ strategy can still be used, but to get a reasonable performance $R$ should be chosen sufficiently large (e.g., $1/\gamma_1$), in which case the Push $R$ would behave very similar to the Push strategy, except when the system is highly loaded.

Once more we assume geometric service times with a mean of $4$ slots. Under the Push strategy, only the masters will perform push operations and they will only push jobs towards workers (i.e., $p_{12} = 1$). For the pull strategy there is clearly no use in pulling jobs from a worker as it never has any jobs waiting (i.e., we can set $B_2 = 0$), thus all the queues send a request to a master when becoming/being idle. This is in contrast to some of the strategies considered in [4] where a worker steals several jobs (half) and therefore often has some waiting jobs.

Figure 12 depicts the mean delay as a function of the system load for $\gamma_1 = 1/2, 1/3$ and $1/10$, meaning half, one third and one tenth of the queues are masters, respectively. For $\gamma_1 = 1/2$, we find that the Push strategy outperforms the Pull strategy for all system loads $\rho$, while the reverse is true when $\gamma_1 = 1/10$. When one third of the queues are masters, the Push strategy results in lower delays under high loads only. Further, the Pull strategy tends to perform better when the proportion of masters is smaller, while the Push strategy first slightly improves as well, but becomes worse as $\gamma_1$ further decreases. In short, the aggressive Push strategy no longer outperforms the traditional Pull strategy when there are far more workers than masters.

With respect to the request rate (not shown here), we should add that the Pull strategy still requires fewer messages, but the rate increases as $\gamma_1$ decreases. More specifically, in the homogeneous system we noticed that the request rate under the Pull strategy decreases to zero as the load approaches one, this can clearly no longer be the case in a master-worker setting, instead the request rate decreases to $(1 - \gamma_1)/4$ (when the mean service time is $4$) as the load approaches one.

## VII. Conclusions and model extensions

In this paper we introduced a class of aggressive push strategies for large distributed systems and compared their performance with a more traditional set of pull strategies. We developed a number of mean field models and identified the best strategies in a number of different settings, i.e., the homogeneous system, slow/fast servers and the master-worker setup. The models presented in this paper can be extended in various manners. For instance, it is easy to incorporate a parameter $P \geq 1$ such that the queues only attempt to push/pull jobs every $P$ time slots (essentially by observing the system every $P$ time slots and using $Q_k(\mu(t))(S_k(t)A_k(t))^P$ instead of $Q_k(\mu(t))S_k(t)A_k(t)$). Similarly, we could also introduce $L$ rounds of push/pull attempts during each time slot instead of just one. With some additional effort it is also possible to analyze strategies that rely on a threshold $T > 1$ as in [3], [10] or to pull or push multiple jobs at once as in [5].

## References


[1] J. L. Boudec, D. McDonald, and J. Mundinger. A generic mean field convergence result for systems of interacting objects. In *Proc. of QEST 2007*, pages 3–15, Edinburgh, UK, 2007.

[2] R. Boute, S. Disney, M. Lambrecht, and B. Van Houdt. An integrated production and inventory model to dampen upstream demand variability in the supply chain. *European Journal of Operational Research*, 178:121–142, 2007.

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

[4] N. Gast and B. Gaujal. A mean field model of work stealing in large-scale systems. *SIGMETRICS Perform. Eval. Rev.*, 38(1):13–24, 2010.

[5] N. Gast and B. Gaujal. A mean field approach for optimization in discrete time. *Discrete Event Dynamic Systems*, 21(1):63–101, 2011.

[6] T. Kurtz. *Approximation of population processes*. Society for Industrial and Applied Mathematics, 1981.

[7] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods and stochastic modeling*. SIAM, Philadelphia, 1999.

[8] M. Livny and M. Melman. Load balancing in homogeneous broadcast distributed systems. In *Proceedings of the Computer Network Performance Symposium*, pages 47–55, New York, NY, USA, 1982. ACM.

[9] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Analysis of the effects of delays on load sharing. *IEEE Trans. Comput.*, 38(11):1513–1525, 1989.

[10] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Adaptive load sharing in heterogeneous distributed systems. *J. Parallel Distrib. Comput.*, 9(4):331–346, 1990.

[11] M. Mitzenmacher. Analyses of load stealing models based on families of differential equations. *Theory of Computing Systems*, 34:77–98, 2000.

[12] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12:1094–1104, October 2001.

[13] N. G. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *Computer*, 25(12):33–44, 1992.

[14] M. S. Squillante and R. D. Nelson. Analysis of task migration in shared-memory multiprocessor scheduling. *SIGMETRICS Perform. Eval. Rev.*, pages 143–155, 1991.

[15] B. Van Houdt, C. Develder, J. Perez, M. Pickavet, and B. Dhoedt. Mean field calculation for optical grid dimensioning. *IEEE/OSA Journal of Optical Communications and Networking*, 2(6):355–367, 2010.