Probability in the Engineering and Informational Sciences, **33**, 2019, 348–366.
doi:10.1017/S0269964818000323

MULTI-CLASS RESOURCE SHARING WITH BATCH ARRIVALS

PAUL EZHILCHELVAN AND ISI MITRANI
School of Computing Science,
Newcastle University, Newcastle upon Tyne, NE4 5TG, UK
E-mail: paul.ezhilchelvan@ncl.ac.uk; isi.mitrani@ncl.ac.uk

A cloud provider hosts virtual machines (VMs) of different types, with different resource requirements. There are bounds on the total amounts of each kind of resource that are available. Requests arrive in batches of different sizes. Under the 'complete blocking' policy, a request is accepted only if all the VMs in its batch can be accommodated. The 'partial blocking' policy would accept a request if there is room for at least one of the VMs in the batch. Blocked requests are lost, with an associated loss of revenue. The trade-offs between costs and benefits are evaluated by means of appropriate models, for which novel solutions based on fixed-point iterations are proposed. The applicability of those solutions is extended, by means of simplifications, to very large-scale systems. Numerical examples and comparisons with simulations are presented.

Keywords: applied probability, queueing theory, simulation

1. INTRODUCTION

A cloud provider may offer services of different types, with different patterns of demand, resource requirements and charges. A job of a given type is run by instantiating an appropriate Virtual Machine (VM), provided that the resources it requires are available. There are bounds on the total amounts of different resources, so that whether a VM can be instantiated or not, depends both on the type of the new job and on the numbers and types of the other jobs already running.

We are concerned with systems where user demands arrive in batches whose sizes may be fixed or random, and may depend on the type. One admission policy for such demands is to say that either all VMs in a batch must be instantiated, or none. This is known as the *complete blocking* policy. Alternatively, one may accept some of the VMs in a batch and reject the ones that cannot be allocated; that is the *partial blocking* policy.

There are many applications which require a batch of VMs in order to complete a certain task within a certain period of time. These are often concerned with the analysis of large volumes of data, such as those arising in the fields of sociology, biology or high energy physics. In particular, the 'MapReduce' framework (e.g., see [5]), allowing the deployment of batches of VMs, is widely used.

© Cambridge University Press 2018 0269-9648/18 $25.00

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

The trade-offs in this context are between the costs incurred by providing resources, and the revenues obtained by running jobs. In the case of partial blocking, the revenue per job may depend on whether the batch was accepted in full, or in part. The general problem is to decide what amounts of the various resources to provide, so as to maximize the average long-term profit (revenues minus costs) per unit time. To that end, we analyze and solve Erlang-type loss models with multi-class batch arrivals and either complete or partial blocking. Those solutions allow us to evaluate the expected profit for a given set of parameters, and hence search numerically for the optimal resource provision.

We assume that the demand parameters are given, and the system reaches a steady state during a period where those parameters remain fixed. In practice, the resource provisioning policies would have to be supplemented by some monitoring and parameter estimation technique that would detect when the traffic parameters change. Such techniques exist (see below). It is also worth pointing out that batch arrivals can, and have been, used to model bursty arrival streams.

In a 1995 paper [3], Choudhury, Cheung and Whitt claimed to provide a product-form solution for both the partial blocking and the complete blocking policies. That solution agreed with the results obtained by Kaufman and Regge [14], for the general distribution of batch sizes, and by van Doorn and Planken [6], for geometrically distributed batches. However, both of those papers had analyzed only the partial blocking policy.

We agree that a product-form solution holds in systems with the partial blocking of batches, but will show that it does not hold in the case of complete blocking. Moreover, when a product-form solution exists, it tends to suffer from problems of scale. This is because the evaluation of the normalization constant becomes numerically intractable even for systems of moderate size.

The main purpose of the present paper is to propose some easily implementable, accurate, and numerically stable approximate solutions for both the complete blocking and partial blocking policies. These approximations are based on fixed-point iterations and, with certain simplifications, can be applied to very large-scale systems.

The rest of the existing literature on multi-class resource sharing deals mainly with demands arriving one at a time in Poisson streams (i.e., no batches). Much of the work is in the context of circuit-switched networks, for example, Kelly [15], Hampshire et al. [10], Kaufman [13], Roberts [19] and Ross [20]. In the telephony field, the resources are the circuits available on various links, and the job types are indexed by the set of links that can be reserved for a call. The optimal allocation of VMs on servers hired from a cloud was explored in Ezhilchelvan and Mitrani [8], and in Tan et al. [22]. Again, one-at-a-time Poisson arrivals were assumed.

More distantly related is quite a large body of work on server allocation with a single job type. Ezhilchelvan and Mitrani [7] showed that dynamic allocation policies do not bring significant benefits over static ones. The trade-off between performance and energy consumption was examined by Mazzucco et al. [16,17], using models and empirical observations. Their focus, and also that of Bodík et al. [1], was on estimating the traffic and reacting to changes in the parameters.

Four of the following sections are devoted to the complete blocking policy. They define the model and the profit maximization problem, demonstrate that the solution presented by Choudhury et al. is incorrect in the context of complete blocking, develop and evaluate the proposed fixed-point approximation, and extend the latter to very large-scale systems. In Section 6, those methods are applied to the partial blocking policy and are shown to lead to fast, accurate and numerically stable approximations. The paper ends with a summary of conclusions and directions for future work.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

A preliminary version of the present paper [9] was presented at the 14th International Conference on Quantitative Evaluation of SysTems (QEST 2017). It completely ignored the partial blocking policy, and the numerical examples were restricted to a single resource type. The inclusion in the present version of a two-dimensional example and of Section 6 remedies these deficiencies. In addition, we have now addressed the question of existence of the fixed point in a little more depth.

2. THE COMPLETE BLOCKING MODEL

The hosting infrastructure provides R different types of resources, such as central processing units (CPUs), memory, interconnection bandwidth, and so on. The total amount of available resource of type r is D_r, referred to as the 'resource capacity' of type r $(r = 1, 2, \ldots, R)$. Those resources are shared by VMs, or jobs, of T different types. A job of type j requires an amount $d_{j,r}$ of type r resource $(j = 1, 2, \ldots, T; r = 1, 2, \ldots, R)$. In order to run a job, all its resource requirements must be satisfied. For every j, at least one of the requirements $d_{j,r}$ is greater than 0, which imposes a limit on the maximum number, m_j, of type j jobs that can run in parallel:

$$m_j = \min_r \left\{ \left\lfloor \frac{D_r}{d_{j,r}} \right\rfloor \right\}, \tag{1}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x.

Moreover, if D_r is replaced by the type r resource capacity *currently available* (determined by the numbers and types of jobs currently running), then (1) provides a limit on the number of type j jobs that can be admitted at a given moment.

Requests of type j arrive as an independent Poisson stream with rate λ_j. Each such request consists of a batch of jobs, all of type j, whose size has an arbitrary distribution dependent on j: there are k jobs in the batch with probability $q_{j,k}$. The probabilities $q_{j,k}$ can be arbitrary, provided that they add up to 1. However, a sensible batch size distribution would not include batches of size 0, and would be consistent with (1). In other words, there should be a limit, K_j, on batch sizes of type j, such that $K_j \leq m_j$. Otherwise, some batches would be rejected even if there are no other jobs present.

The assumption of Poisson arrivals is justified by the fact that requests are submitted by a large number of independent users. If there is some background work present in the system, it is assumed to be of low priority; it is preempted by the user jobs and does not interfere with them.

If there is at least one job in an incoming batch that cannot be run because at least one of the resources it requires cannot be provided, then the whole batch is rejected. That is the *complete blocking* policy.

The system structure is illustrated in Figure 1.

Service times for jobs of type j are assumed to be i.i.d. random variables distributed exponentially with mean $1/\mu_j$ $(j = 1, 2, \ldots, T)$. The insensitivity property of the Erlang model does not hold when jobs arrive in batches (see [13]).

Under the above assumptions, the vector $(n_1(t), n_2(t), \ldots, n_T(t))$, where $n_j(t)$ is the number of type j jobs present in the system at time t, is a finite-state, irreducible Markov process. It therefore always has a stationary distribution (see [18]).

Suppose that each unit of resource of type r costs c_r to provide, and each job of type j that is run brings in a revenue of v_j. Denote by $\alpha_{j,k}$ the steady-state probability that an incoming batch of type j and size k is accepted. The total average profit, V, that the

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

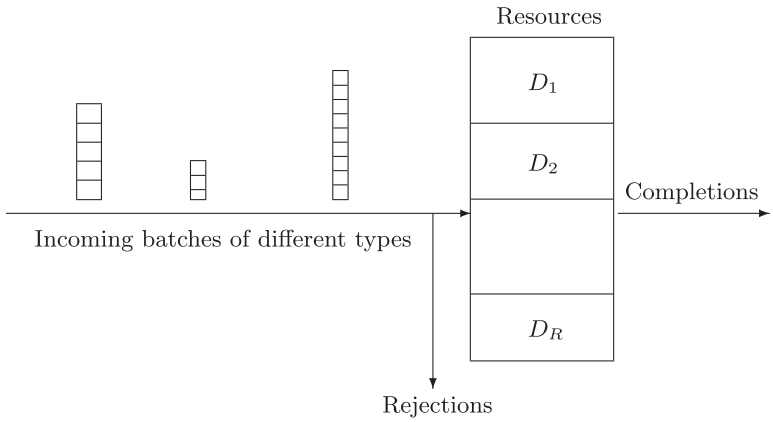


FIGURE **1.** Resource sharing with batch arrivals.

system generates per unit time in the steady state can be expressed as

$$V = \sum_{j=1}^{T} \lambda_j \sum_{k=1}^{K_j} q_{j,k} \alpha_{j,k} k v_j - \sum_{r=1}^{R} c_r D_r. \tag{2}$$

Clearly, increasing the resource capacities D_r leads to higher revenues, but also higher costs. The profit optimization problem is to choose D_r so as to maximize V. One could also consider a Quality-of-Service problem, which is to find the minimum values of D_r for which the acceptance probabilities $\alpha_{j,k}$ exceed certain pre-defined targets. In both cases, the evaluation of the objective function depends on being able to determine $\alpha_{j,k}$.

3. AN ERRONEOUS SOLUTION

The technique used in Choudhury et al. [3] is to replace an incoming batch of type j and size k (when $q_{j,k} > 0$), by a single *macro job* that goes through a series of k queues in tandem: in the first queue, it uses $kd_{j,r}$ units of type r resource and is served at rate $k\mu_j$, regardless of how many other such macro jobs are present; in the second queue it uses $(k-1)d_{j,r}$ units of resource and is served at rate $(k-1)\mu_j$; this goes on until queue k, where it uses $d_{j,r}$ units of resource and is served at rate μ_j. After that, the macro job departs.

Another interpretation of these queues is to think of them as having infinitely many servers. Each ordinary job is served on a separate server and as soon as one of them completes, the remaining macro job moves to the next queue.

In this formulation, which is equivalent to the one in terms of batches of ordinary jobs, the system state is a vector of integers $[n_{j,k,s}]$, specifying the numbers of macro jobs of type j and size k that are now in queue s of their series of queues (i.e., they have $k+1-s$ ordinary jobs remaining). The authors find that, for both partial and complete blocking, the probabilities of those vectors satisfy local balance, and therefore the steady-state distribution has product form:

$$\pi(\mathbf{n}) = G \prod_{j=1}^{T} \prod_{k=1}^{K_j} \prod_{s=1}^{k} \frac{(\lambda_j q_{j,k})^{n_{j,k,s}}}{((k+1-s)\mu_j)^{n_{j,k,s}} n_{j,k,s}!}, \tag{3}$$

where $\mathbf{n}$ is the state vector $[n_{j,k,s}]$ and G is a normalization constant.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

To demonstrate that this product form does not hold in the case of complete blocking, we offer the following simple counter-example. Take a system with a single resource type, a single job type, and a single batch size. The resource capacity is 4, the resource requirement per job is 1, and the incoming batch size is 3. The arrival and service rates are both 1.

There are now 3 queues in series, so the system state is a triple (n_1, n_2, n_3). The feasible states are (0,0,0), (1,0,0), (0,1,0), (0,0,1), (1,0,1), (0,1,1), and (0,0,2). The state (0,2,0), for instance, is not feasible because if there was a macro job in queue 2, consuming 2 resource units, then a new macro job requiring 3 units would not have been admitted.

Consider the two states (0,0,1) and (1,0,1). If (3) is correct, then their stationary probabilities are $\pi(0,0,1) = G$ and $\pi(1,0,1) = G/3$. Hence, $\pi(0,0,1) = 3\pi(1,0,1)$. On the other hand, the only way of leaving state (1,0,1) is by a service completion, either at queue 1, at rate 3, or at queue 3, at rate 1. The total completion rate is 4. The only way of entering state (1,0,1) is by an arrival of a new batch, at rate 1, when the system is in state (0,0,1). Therefore, $\pi(0,0,1) = 4\pi(1,0,1)$. This contradiction demonstrates that the solution (3) is not correct.

The failure of the product form is due to the fact that, contrary to the assertion in [3], local balance does not hold in the case of complete blocking. In the above example, in states (1,0,1) and (0,1,1), a service completion at queue 3 (leading to states (1,0,0) and (0,1,0) respectively), cannot be balanced by an arrival because in either case the new batch would be rejected.

In principle, there might exist a different, as yet undiscovered product form solution for the complete blocking model. For example, one might explore reversibility arguments, via the RCAT theorem (see Harrison, [11]). However, given the strong connection that is known to exist between reversibility and local balance (e.g., see [2]), we believe that such a search is very unlikely to succeed.

In the absence of a tractable exact solution, we now turn to the task of finding an accurate approximation.

4. A FIXED-POINT APPROXIMATION

We propose treating each job type as if it was an isolated, one-dimensional Markov process taking place within a static environment determined by the other job types. More precisely, when considering jobs of type j, assume that all jobs of other types in the system are consuming a fixed total amount, $Z_{j,r}$, of type r resource ($r = 1, 2, \ldots, R$). In other words, type j operates in an environment where the available resource of type r is $D_r - Z_{j,r}$. Hence, the maximum number of type j jobs that can be admitted, m_j, is given by (1), with D_r replaced by $D_r - Z_{j,r}$.

Under the above assumption, the type j Markov process evolves on the state space $\{0, 1, \ldots, m_j\}$. Let $\pi_j(n)$ be the stationary probability of state n, that is, the probability that there are n type j jobs present. We can write a set of recurrence relations for these probabilities by examining the flows across a cut separating states $0, 1, \ldots, n-1$ from states $n, n+1, \ldots, m_j$. The relevant balance diagram is illustrated in Figure 2.

The downward (i.e., to the left) flow across the cut contains just a single transition, from state n, due to a departure; the corresponding rate is $n\mu_j$. On the other hand, if an arrival occurs in any state, i ($i < n$), and the size of the incoming batch is at least $n - i$ but does not exceed $m_j - i$, then it contributes to the upward (to the right) flow across the

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

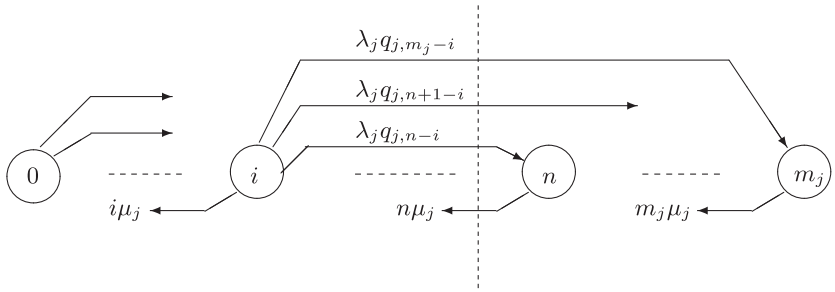


FIGURE **2.** Balance diagram for type j.

cut. Equating the two flows provides a balance equation for each n:

$$n\mu_j\pi_j(n) = \lambda_j \sum_{i=0}^{n-1} \pi_j(i) \sum_{k=n-i}^{m_j-i} q_{j,k}; \quad n = 1, 2, \ldots, m_j. \tag{4}$$

The simplest way to solve these equations is to set $\pi_j(0) = 1$, evaluate $\pi_j(n)$ for $n = 1, 2, \ldots, m_j$ according to (4), and then re-normalize by dividing each of them by their sum.

Having computed the probabilities $\pi_j(n)$, the probability that an incoming batch of type j and size k is accepted, is given by

$$\alpha_{j,k} = \sum_{n=0}^{m_j-k} \pi_j(n). \tag{5}$$

The average number, L_j, of type j jobs present is equal to

$$L_j = \sum_{n=1}^{m_j} n\pi_j(n). \tag{6}$$

Consequently, the average amount, $z_{j,r}$, of type r resource consumed by jobs of type j, is given by

$$z_{j,r} = d_{j,r}L_j. \tag{7}$$

In common with most models of this type, the above solution depends only on the ratio $\rho_j = \lambda_j/\mu_j$, and not on the individual values of those parameters.

We can now approximate the effect that type j has on the other job types by treating its *average* consumption of resource r, $z_{j,r}$ as a *fixed* consumption of resource r. This will form part of the environment in which another given job type operates.

Suppose that we have somehow obtained an estimated vector, $\mathbf{L} = (L_1, L_2, \ldots, L_T)$, of the average numbers of jobs of different types in the system. Carry out the following procedure.

1. For every j, compute the type r resource, $Z_{j,r}$, consumed by job types other than j:

$$Z_{j,r} = \sum_{i=1, i\neq j}^{T} z_{i,r}, \tag{8}$$

with $z_{i,r}$ is given by (7).

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

2. For every j, use equations (4)–(7) in order to compute new values for L_j, $\alpha_{j,k}$ and $z_{j,r}$.

This procedure implements a mapping, $f(\cdot)$, from one vector of averages, call it $\mathbf{L}^{old}$, to another vector of averages, $\mathbf{L}^{new}$. Our approximate solution consists in finding a vector, $\mathbf{L}^*$, whose 'new' image is the same as the 'old' one. That is, $\mathbf{L}^*$ is a fixed point of the mapping $f(\cdot)$:

$$\mathbf{L}^* = f(\mathbf{L}^*). \tag{9}$$

At the fixed point $\mathbf{L}^*$, the environments in which the different job types operate are consistent with each other. That is, for every job type j, the resources it consumes, $z_{j,r}$, do not alter the resources consumed by the other job types, $Z_{j,r}$. In that sense, this fixed point is like a Nash equilibrium in multi-person games.

The acceptance probabilities corresponding to $\mathbf{L}^*$ are substituted into (2) in order to compute the profit V, or are used to see whether the QoS targets have been met.

To compute $\mathbf{L}^*$, we use an iterative schema of the form

$$\mathbf{L}^{(i+1)} = f(\mathbf{L}^{(i)}). \tag{10}$$

These iterations start with some initial vector such as $\mathbf{L}^{(0)} = (0, 0, \ldots, 0)$, and stop when two consecutive vectors are sufficiently close to each other. To reduce the number of iterations, it is advisable that the evaluations are of the Gauss–Seidel type, that is, as soon as a new value for some L_j is obtained, the corresponding value of $z_{j,r}$ is used in determining the environment for other job types.

Note. Fixed-point approximations of queuing systems have been used in the past, mainly in the context of open or closed networks. There, the decomposition is in terms of nodes and the fixed point equations attempt to capture the interactions between them (e.g., see Sadre et al. [21], Whitt [23]). Kelly's fixed-point approximation for multi-class circuit-switched networks [15] is concerned with shared resources but does not model batch arrivals. The decomposition is with respect to offered loads for individual units of resource.

As far as we are aware, a decomposition by job type, where the fixed point equations capture different contributions to a shared environment, has not been used before.

4.1. Existence and Uniqueness of the Fixed Point

The mapping $f(\cdot)$ is not continuous, because equation (1) involves the 'floor' function $\lfloor x \rfloor$. Consequently, we cannot invoke Brouwer's theorem and assert the existence of the fixed point $\mathbf{L}^*$. There are other results that apply to discontinuous functions. For example, Hering et al. [12] proved the existence of a fixed point for functions that satisfy a property called 'locally gross direction preserving'. In Cromme [4], an approximate fixed point is shown to exist, given certain bounds on the discontinuities. In all those cases, checking the conditions of the relevant theorem requires an explicit characterization of the function whose fixed point is sought.

Our mapping is defined only by the algorithm that is used to compute it. We know that a small change in $Z_{j,r}$ may cause the maximum number of jobs of type j, m_j, to increase or decrease by 1. However, evaluating the effect of such a change in the average number of jobs of type j, L_j, is non-trivial. One has to solve equations (4), and the resulting change may be small or large, depending on the traffic parameters. For that reason, we have been unable to use existing results in order to prove the existence of a fixed point.

However, it is possible to give an intuitive reason for the convergence of iterations (10), by following the evolution of resource allocations. In order to do that, we need to make some general remarks about average numbers of jobs and offered loads.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

The average offered load of type j is given by $\sigma_j = \lambda_j b_j/\mu_j$, where b_j is the average batch size of type j. That would also be the average number of type j jobs present if all resource capacities were infinite (e.g., see [18]). Hence, the elements of the vectors $\mathbf{L}$ are bound by

$$L_j \leq \sigma_j; \quad j = 1, 2, \ldots, T. \tag{11}$$

Moreover, the more resources are allocated to type j, the closer L_j is to σ_j.

It is important to bear in mind that real-life clouds are not, on the average, overloaded. That is, more resources are provided than are required by the average offered loads:

$$\sum_{j=1}^{T} \sigma_j d_{j,r} < D_r; \quad r = 1, 2, \ldots, R. \tag{12}$$

This, together with (11), implies that there are some unconsumed resources throughout the iteration process.

Consider, for simplicity, a single resource system with two job types. Start the first iteration with $Z_1 = 0$, that is, the entire resource, D, is allocated to type 1. The resulting m_1 and L_1, take their largest possible values. Type 2 now operates in an environment where $Z_2 = L_1 d_1$, and the available resource is $D - Z_2$. The corresponding values for m_2 and L_2 are small. In the second iteration, the resource available to type 1 decreases, causing m_1 and L_1 to stay the same or decrease, and m_2 and L_2 to stay the same or increase. This goes on until, in two consecutive iterations, m_1 remains the same and m_2 also remains the same. Then so do L_1 and L_2, and the process terminates.

In general, with T job types, if in one iteration the allocations of resources to certain job types are too generous while the allocations to others are not generous enough, there will be a transfer of unconsumed resources from the former to the latter. In the next iteration, the first set of averages will stay the same or decrease, while the second will stay the same or increase.

Thus the iterative process naturally pushes the vector $\mathbf{L}$ towards a point of equilibrium.

The fixed point reached by these iterations is not necessarily unique. To see that, consider the above two-type example under very heavy overload conditions. Suppose that both arrival rates are close to infinity, and assume that all batches are of size 1 and all resource requirements are 1. If the entire resource, D, is initially allocated to type 1, then type 1 will consume it all, and the process will terminate after one iteration, reaching the fixed point $(D, 0)$. If, on the other hand, the entire resource is initially allocated to type 2, then type 2 will consume it all and the process will terminate after one iteration with the fixed point $(0, D)$. In fact, any initial split of the resource will remain unchanged and will become the final split.

This example is of course unrealistic. We wish to re-emphasize that the systems of practical interest are not overloaded. Indeed, in the experiments that follow, the optimal resource capacity that should be provided turns out to be more than twice as large as the requirement of the average offered load. Under such conditions, any fixed point is likely to be close to the true vector $\mathbf{L}$, and therefore even if there are multiple fixed points, they are likely to be close to each other.

4.2. Numerical and Simulation Experiments

To examine the quality of the proposed approximation, consider an example system with four job types, 1, 2, 3 and 4, or "small", "medium", "large" and "very large". There is a single resource type and the individual resource requirements of the four job types are are

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

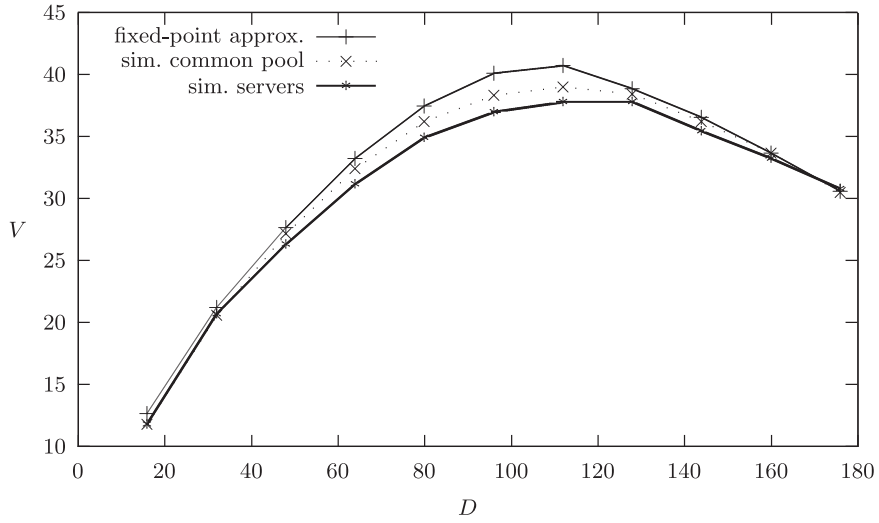


FIGURE **3.** Estimated and simulated average profit.

$d_1 = 1$, $d_2 = 2$, $d_3 = 4$, and $d_4 = 8$. These numbers are motivated by similarities with the T2 family of VM instances offered by the Amazon EC2 (Elastic Computing Cloud) service (see [24]). The resource that is being shared in this context is vCPUs (virtual CPUs).

Type 1 jobs arrive singly, at a rate of $\lambda_1 = 6$ jobs per unit time. Their average service times are $1/\mu_1 = 1$. For type 2, the possible batch sizes are 1 or 2, with equal probability ($q_{2,1} = q_{2,2} = 0.5$). The traffic parameters are $\lambda_2 = 4$ and $1/\mu_2 = 1$. Jobs of type 3 and type 4 arrive in batches of size 1, 2, or 3, with probabilities 0.4, 0.3 and 0.3, respectively. Their arrival and service parameters are $\lambda_3 = 2$, $1/\mu_3 = 0.5$, $\lambda_4 = 1$, $1/\mu_4 = 0.5$.

The average offered load requirement for this example, that is, the left-hand side of (12), is 33.2 vCPUs.

The cost incurred per unit of resource is $c = 0.2$, and the revenues brought in by the different job types increase with the resource consumed: $v_1 = 1$, $v_2 = 3$, $v_3 = 6$, $v_4 = 10$.

In Figure 3, the estimated average profit is plotted against the offered resource capacity, D. The three curves correspond to (a) the fixed-point approximation, (b) a simulation of the model as described in Section 2, and (c) a simulation of a system where the resource is not provided as a common pool, but is contained within servers. In this last case, each server has 16 vCPUs. Thus, in order to offer 128 vCPUs, one has to provide 8 servers. The discrete server system is less efficient than the common pool, because, in order to instantiate a VM on a server, all its resource requirement must be available on that server. It is not enough to have some of the requirement available on one server and some on another.

The two simulation graphs may be considered exact. Their confidence intervals are too narrow to plot. The figure shows that the fixed-point approximation tends to overestimate the average profit slightly. This is not surprising since replacing a random environment with a fixed one reduces the variance of the process. The relative differences between the approximated and simulated (common pool) points are on the order of 5% or less.

The discrete server simulation confirms the intuition that that system is less efficient, but again the differences from the common pool are on the order of 5% or less. In particular, all three graphs agree that the optimal resource capacity that should be provided is 112 vCPUs, or 7 servers. This is just over three times greater than the requirement of the average offered load.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

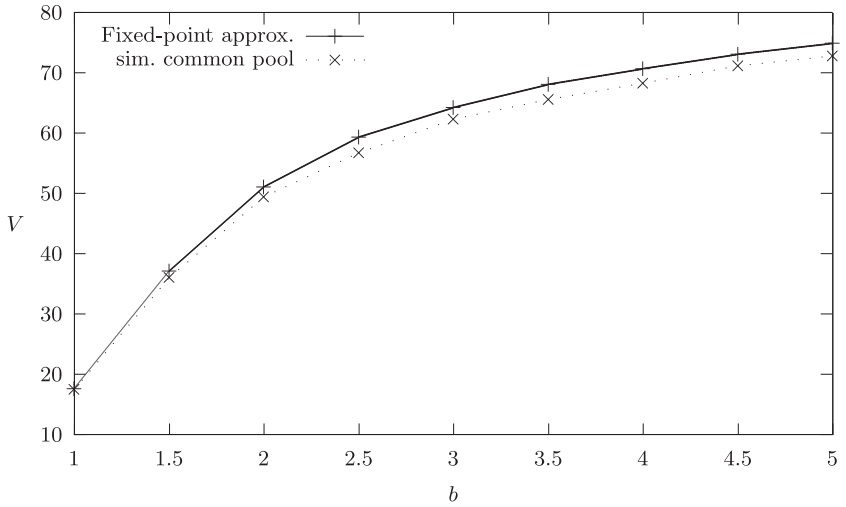


FIGURE **4.** Increasing average batch sizes.

The convergence of iterations (10) to the fixed point is very fast. The termination criterion in this example was that $|L_j^{(i+1)} - L_j^{(i)}| < 10^{-6}$ for all job types j. The entire approximation graph took less than tenth of a second to compute, and no point required more than 4 iterations.

We have also simulated a discrete server system where each server contains 8 vCPUs instead of 16. The differences between that system and the one shown in the figure are negligible.

In order to observe the behavior of the approximation under different loading conditions, a single resource system with four job types has been evaluated for increasing average batch sizes. Specifically, the batch sizes of all four job types are uniformly distributed between 1 and K, and K increases from 1 to 9. In other words, the average batch sizes, b, increase from 1 to 5. The resource provided is fixed at 112 vCPUs. All other parameters are the same as in the first example.

When $b = 5$, the average offered load requirement is 110 vCPUs, so at that point the system is quite heavily loaded.

In Figure 4, the estimated and simulated profits are plotted against b. The results confirm that the quality of the fixed-point approximation is excellent at light loads (this is not surprising), but they also show a good agreement with the simulation at heavy loads. The largest relative errors are on the order of 5%.

In the next example, all arrival rates are scaled up by a factor of 100: $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (600, 400, 200, 100)$. The resource is again provided in a common pool, with scaled capacities ranging from 7000 to 10000 vCPUs. The other parameters, revenues and costs are as in example 1. The aim of this exercise is to see whether the fixed-point approximation is sufficiently robust to cope with the much larger state space and offered loads. The estimated and simulated average profits are displayed in Figure 5.

The accuracy of the approximation has increased. Not only does the model predict the optimal resource allocation correctly, but the relative differences between estimated and simulated profits are now less than 1%. This is not really surprising since the flows of traffic of various job types through the system behave more and more like deterministic fluids when capacities and offered loads increase.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

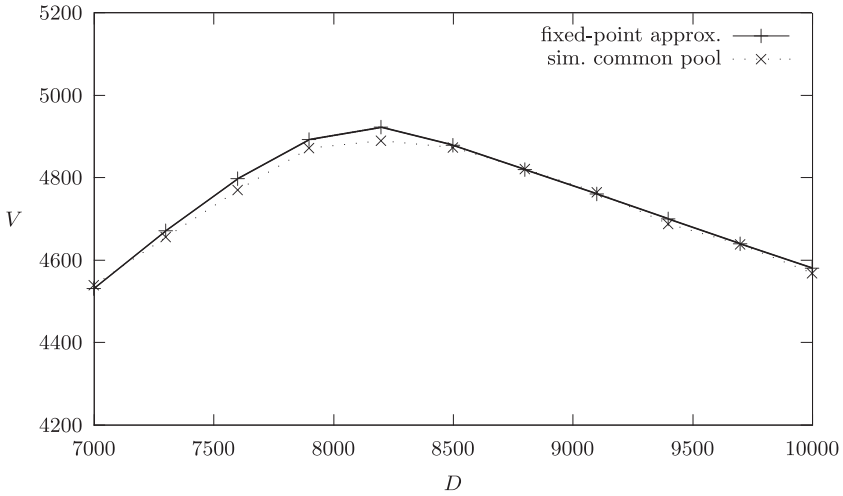


FIGURE **5.** System scaled up by a factor of 100.

TABLE 1. vCPU and Storage Requirements

Type	vCPUs	GB storage
1	$d_{1,1} = 1$	$d_{1,2} = 4$
2	$d_{2,1} = 2$	$d_{2,2} = 32$
3	$d_{3,1} = 4$	$d_{3,2} = 80$
4	$d_{4,1} = 8$	$d_{4,2} = 160$

We conclude that, for systems of this and similar sizes, it would be preferable to use the fixed-point approximation even if an exact product-form solution was available. The reason is that the complexity and numerical problems associated with computing the relevant normalization constants increase very quickly with the number of job types, the incoming batch sizes and the offered resource capacities.

Let us now extend the last example by adding a second shared resource, storage. The resource requirements of the four job types are specified in Table 1.

The traffic and batch size parameters are the same as before, as are the revenues per job and the unit cost per vCPU. The unit cost per GB of storage is 0.01.

The total vCPU provision, D_1, is varied between 7000 and 9700 in increments of 300, while the total storage capacity, D_2, varies between 80 and 280 terabytes (TB), in increments of 20. The resulting profits, V, computed by applying the fixed-point iterations, are plotted in Figure 6.

The corresponding simulation results are not shown in the figure, because the two surfaces would be too close to distinguish.

In this example, the optimal resource provision is $D_1 = 7900$ vCPUs and $D_2 = 140$ TB storage. The corresponding maximum profit is $V = 3404$ (subject to the granularity of the increments). Note that the introduction of a second resource constraint has led to a significant reduction in the achievable profits. This was to be expected since more incoming batches are now rejected.

In the general case of R resource types, one would have to explore an R-dimensional surface in order to search for the optimal resource provision. There are various existing

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

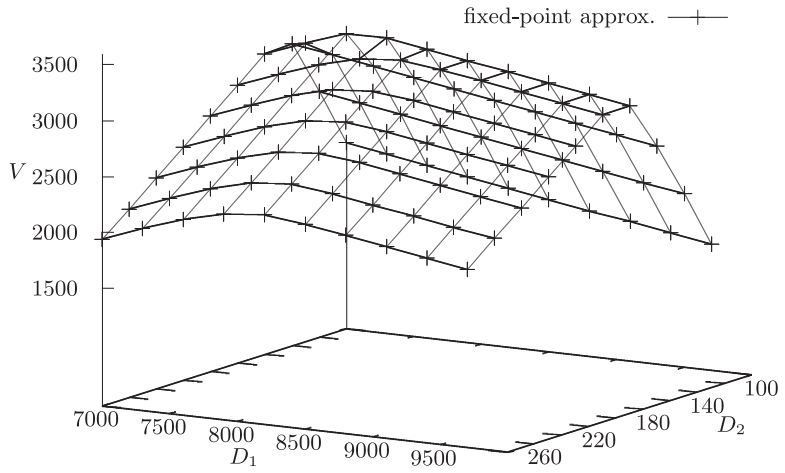


FIGURE **6.** Two types of shared resources: D_1 vCPUs and D_2 TB storage.

methods for performing such searches. One way of reducing the effort, at the cost of accepting a possibly sub-optimal result would be to carry out R one-dimensional searches. In step 1, search for the best D_1 value, ignoring $D_2, D_3, \ldots, D_R$. In step 2, search for the best D_2 value, fixing D_1 as computed already, and ignoring $D_3, \ldots, D_R$. Carry on until step R, which searches for the best D_R value, with $D_1, D_2, \ldots, D_{R-1}$ fixed. This will not necessarily find the optimum but is likely to produce a reasonable resource allocation. In our two-dimensional example, the two 1-dimensional searches yield $D_1 = 8200$ vCPUs and $D_2 = 140$ TB of storage. The corresponding profit is $V = 3344$, which is close to optimal.

Although the fixed-point approximation is accurate and efficient, there are limits to its applicability. For example, if the system in Figure 5 is scaled up by another factor of 10, bringing the arrival rates to $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (6000, 4000, 2000, 1000)$ and the vCPU resource capacities in the region of 100000, our solution breaks down. The failure is not in the fixed-point iterations, but in the one-dimensional solutions (4). When the maximum number of jobs, m_j, becomes too large, the implementation of (4) starts to experience numerical overflows.

It is thus desirable to develop another approximation which can be applied to very large systems and produce reasonable estimates, albeit with some loss of accuracy.

5. VERY LARGE-SCALE SYSTEMS

We have observed that the solution of an isolated job type ceases to work when the resource capacities and the offered loads are on the order of tens of thousands or more. For such large systems, a simpler and more robust approximation is required.

With this in mind, we propose to represent the various batch arrivals of type j by single 'macro' jobs with appropriately chosen resource requirements. Then, for the purpose of the fixed-point solution, the isolated type j model becomes a classic Erlang loss process. The benefit of this simplification is that the Erlang B function, which provides the rejection probability, can be computed in a stable manner for large values of the parameters.

The arrival rate and average service time of type j macro jobs are λ_j and $1/\mu_j$, respectively. To define the resource requirement of type r for a macro job of type j, $\delta_{j,r}$, we take

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

the average over the possible type j batch sizes:

$$\delta_{j,r} = d_{j,r} \sum_{k=1}^{K_j} k q_{j,k}. \tag{13}$$

Hence, if all other job types consume a fixed amount, $Z_{j,r}$, of type r resource, then the maximum number of type j macro jobs that can be admitted into the system is

$$m_j = \min_r \left\{ \left\lfloor \frac{D_r - Z_{j,r}}{\delta_{j,r}} \right\rfloor \right\}. \tag{14}$$

The probability, β_j, that an incoming macro job of type j will be rejected, is given by the Erlang-B function (e.g., see [18])

$$\beta_j = B(m_j, \rho_j) = \frac{\rho_j^{m_j}}{m_j!} \left[\sum_{i=0}^{m_j} \frac{\rho^i}{i!} \right]^{-1}. \tag{15}$$

A numerically stable procedure for computing the function $B(m, \rho)$ is provided by the recurrence relation

$$B(m, \rho) = \frac{(\rho/m) B(m-1, \rho)}{1 + (\rho/m) B(m-1, \rho)}, \tag{16}$$

starting with $B(0, \rho) = 1$ (e.g., see [8]).

The average number, L_j, of type j macro jobs in the system is then given by Little's result:

$$L_j = \rho_j (1 - \beta_j). \tag{17}$$

The average amount of type r resource that those jobs consume is $z_{j,r} = L_j \delta_{j,r}$.

We now have the necessary elements for carrying out the iterations described in the previous section and finding the fixed-point vectors of average numbers of macro jobs present, L_j^*, and corresponding rejection probabilities, β_j^*. The average profit achieved per unit time is given by

$$V = \sum_{j=1}^{T} \lambda_j v_j (1 - \beta_j) \sum_{k=1}^{K_j} k q_{j,k} - \sum_{r=1}^{R} c_r D_r. \tag{18}$$

To illustrate the efficacy and accuracy of this approximation, we have scaled up the example with four job types introduced in the previous section. The arrival rates are now $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (6000, 4000, 2000, 1000)$, and the single resource capacity is varied in the range $D \in (70000, 90000)$. All other parameters are as before.

In Figure 7, the average profits predicted by the macro fixed-point approximation are compared with those produced by simulation runs in each of which a total of ten million batches of all types and sizes arrived into the system. Computing the fixed-point was very fast and free from numerical problems. The simulation runs were several orders of magnitude slower because we wanted narrow confidence intervals.

The figure shows that the simplified approximation is remarkably accurate at this scale. The two plots are almost indistinguishable. This confirms the tendency observed earlier, that as the scale of the system increases, its behavior agrees more closely with the deterministic assumptions that underlie the fixed-point approach.

The above observation applies also when more than one type of resource is being shared.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

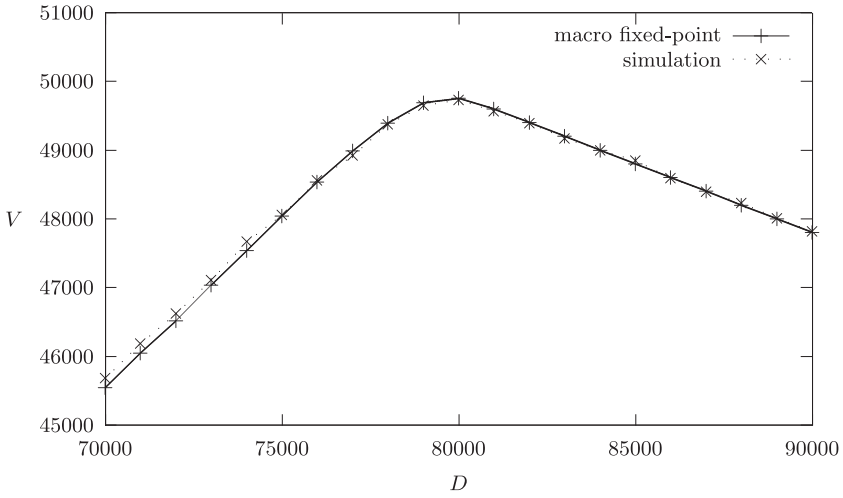


FIGURE **7.** System scaled up by a factor of 1000.

6. PARTIAL BLOCKING

Under the partial blocking policy, the model has a product-form solution (e.g., see [14]). The system state is described by the vector $\mathbf{n} = (n_1, n_2, \ldots, n_T)$, where n_j is the number of jobs of type j present. The feasible states are those satisfying all resource bounds, that is,

$$\sum_{j=1}^{T} n_j d_{j,r} \leq D_r; \quad r = 1, 2, \ldots, R. \tag{19}$$

The stationary probabilities, $\pi(\mathbf{n})$, are then given by

$$\pi(\mathbf{n}) = G \prod_{j=1}^{T} p_j(n_j), \tag{20}$$

where G is a normalization constant. The quantities $p_j(n_j)$ satisfy one-dimensional recurrences whose only difference from (4) is the range of batch sizes admitted:

$$n\mu_j p_j(n) = \lambda_j \sum_{i=0}^{n-1} p_j(i) \sum_{k=n-i}^{K_j} q_{j,k}; \quad n = 1, 2, \ldots, m_j. \tag{21}$$

In addition, [14] established that when a single resource with total bound D is shared, the marginal probabilities, $\psi(s)$, that s resource units are currently allocated, also satisfy a set of one-dimensional recurrences:

$$s\psi(s) = \sum_{j=1}^{T} \rho_j d_j \sum_{i=1}^{\lfloor s/d_j \rfloor} \psi(s - id_j) \sum_{k=i-1}^{K_j} q_{j,k}; \quad s \leq D, \tag{22}$$

where d_j is the resource requirement for type j.

The problem with these exact solutions is that they do not scale well. The normalization constant appearing in (20) is difficult to calculate even for moderately-sized systems. The recurrences (22) are more efficient, but they too are of limited utility. As well as only

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

applying to a single shared resource, we have observed that those recurrences break down numerically when D is greater than about 3000.

In view of these drawbacks, it is advisable to apply a fixed-point approximation to the partial blocking policy. The iterative procedure is as described in Section 4. The recurrences (21) are used to compute the one-dimensional distribution of type j jobs present, given the resource capacities available to them. This provides a mapping, $\mathbf{L}^{new} = f(\mathbf{L}^{old})$, from an old vector of average numbers of jobs present to a new one. That mapping is iterated until it converges to a fixed point.

At this point, one needs to consider the reward structure associated with the partial blocking policy. In particular, there may be penalties for being unable to allocate all the jobs in an incoming batch. Suppose that, if a batch of type j and size k is fully allocated, then the revenue received is kv_j, but if only i of the k jobs are accepted, then the revenue is iw_j, for some w_j which would typically be less than v_j. Then the contribution of type j to the revenues received, V_j, has two terms: revenues received from fully accepted batches and revenues received from partially accepted batches.

$$V_j = \lambda_j \sum_{k=1}^{K_j} q_{j,k} \left[kv_j \sum_{n=0}^{m_j-k} p(n) + w_j \sum_{i=1}^{k-1} ip(m_j - i) \right]. \tag{23}$$

It seems intuitively clear that the differences between the complete and partial blocking policies would become more pronounced when the batch sizes are larger and the resource capacities are less generous. To quantify that observation, we shall take the single resource, 4-type example of Section 4 and modify it by increasing the upper batch limits by a factor of 5: $(K_1, K_2, K_3, K_4) = (3, 10, 15, 15)$. Within those limits, the batch sizes are distributed uniformly, which means that the variances have also been increased. The arrival rates are now $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (120, 80, 40, 20)$. Consequently, in terms of offered loads, this example is similar to the one in Figure 5.

The resource requirements, revenues from fully accepted batches and unit cost per vCPU are as before. However, we now assume that if a batch is only partially accepted, then the revenue from each of its jobs is half of what it would be if the batch was fully accepted: $w_j = 0.5v_j$, $j = 1, 2, 3, 4$.

The three plots in Figure 8 show the profits achieved according to (a) fixed-point approximation of the complete blocking policy, (b) fixed-point approximation of the partial blocking policy, and (c) simulated partial blocking where each point corresponds to a run containing one million batch arrivals.

We see that the partial blocking policy is also very accurately approximated by the fixed-point iterations. The relative errors, as compared with the simulations, do not exceed 2%, and are mostly lower than 1%. It is worth remarking that the optimal resource allocation (7000 vCPUs, subject to the granularity of the increments) is the same for the complete and partial blocking policies.

As expected, when the resource provision is plentiful, there is very little difference between the two policies. That would be true even if the revenue from partially accepted batches was 0. Significant differences begin to emerge when the resource provided is not enough. With our cost structure, the complete blocking policy is more profitable than the partial blocking one. This is because the acceptance of partial batches reduces the probability of accepting full ones, which reduces revenues.

Devising a simple approximation for very large systems is more problematic in the case of the partial blocking policy. Because of the possibility of accepting parts of batches, and the attendant revenue complications, we need to consider both macro jobs (aggregated

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

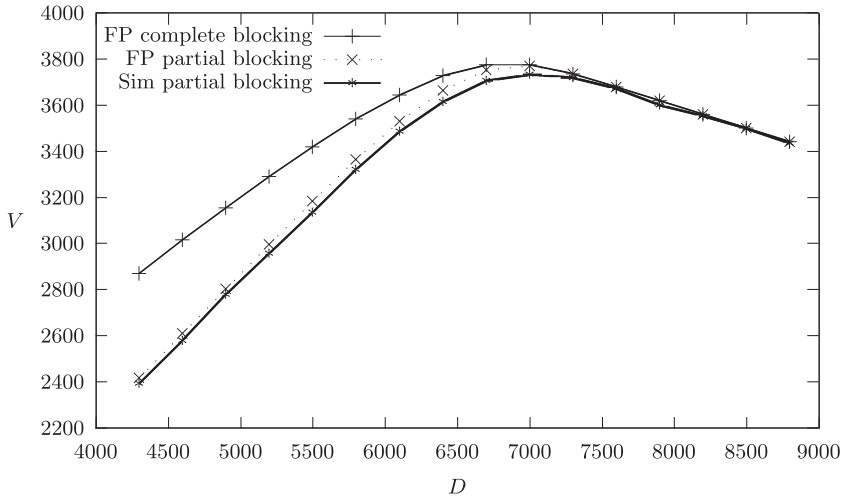


FIGURE **8.** Complete and partial blocking; larger batches.

batches) and individual jobs. The latter will be treated as arriving in Poisson streams where the rate for type j, γ_j, is

$$\gamma_j = \lambda_j \sum_{k=1}^{K_j} k q_{j,k}. \tag{24}$$

The offered load for individual jobs of type j is $\sigma_j = \gamma_j/\mu_j$. For a given resource capacity of type r available to type j, $D_r - Z_{j,r}$, the maximum number of type j jobs that can be present in the system, n_j, is equal to

$$n_j = \min_r \left\{ \left\lfloor \frac{D_r - Z_{j,r}}{d_{j,r}} \right\rfloor \right\}, \tag{25}$$

where $d_{j,r}$ is the type r resource requirement of a type j job.

The fixed-point iterations applied to individual jobs provide the probabilities, β_j, that a job of type j is rejected:

$$\beta_j = B(\sigma_j, n_j). \tag{26}$$

Also, the average number of type j jobs present, and the resources of type r that they use, are given by

$$L_j = \sigma_j(1 - \beta_j); \quad z_{j,r} = L_j d_{j,r}.$$

It remains to assess the average revenue, u_j, that an accepted job of type j would bring. We propose to use a linear combination

$$u_j = \alpha_j v_j + (1 - \alpha_j) w_j, \tag{27}$$

where α_j is the probability that the job belongs to a fully accepted batch. An approximation for α_j is provided by the probability that an aggregated 'macro' job of type j is accepted:

$$\alpha_j = 1 - B(\rho_j, m_j), \tag{28}$$

where $\rho_j = \lambda_j/\mu_j$ is the offered load of type j macro jobs and m_j is their maximum number, given by (14). Note that the available resources have already been computed.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

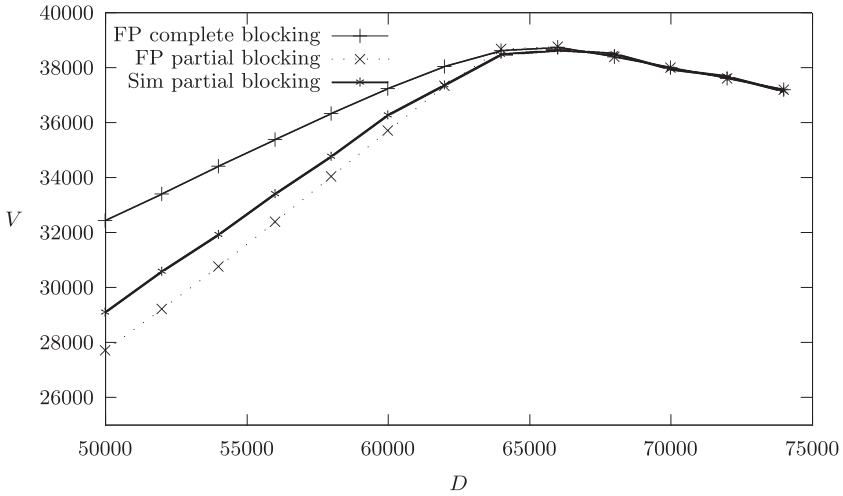


FIGURE **9.** Large system, complete and partial blocking.

The average profit achieved per unit time is now obtained from

$$V = \sum_{j=1}^{T} \gamma_j (1 - \beta_j) u_j - \sum_{r=1}^{R} c_r D_r. \tag{29}$$

In Figure 9, the large-scale approximations are applied to a single resource system where the batch arrival rates are ten times larger than in Figure 8 $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (1200, 800, 400, 200)$. All other parameters, including the batch size distributions, are the same. In terms of offered loads, this example is similar to the one in Figure 7. When the blocking policy is partial, the revenue per job in a partially accepted batch is again half of that in a fully accepted batch. The allocated resource is varied in the range (50000,74000) vCPUs, in increments of 2000.

In the region of insufficient resource allocation, the Erlang-B fixed-point approximation of the partial blocking policy is now a little less accurate, compared with the simulations. However, it is orders of magnitude faster, and the relative errors are still about 5% or less. Moreover, all three plots agree that the optimal resource allocation is 66000 vCPUs.

It is worth emphasizing that when resources are allocated near, or above, their optimal values, the differences between the complete blocking and partial blocking policies are negligible.

7. CONCLUSION

We have addressed a practically relevant problem concerned with service provisioning in public clouds. The multi-class model with batch arrivals and complete blocking appeared to be solved, but we have shown that the existing solution is incorrect. An alternative solution based on fixed-point iterations is introduced. This replaces the multi-dimensional stochastic process with a number of single-dimensional ones, using averages to model the interactions between them. The accuracy of the fixed-point solution is good for small systems, and gets better when the system size increases. A simplified version of that solution is shown to apply to very large-scale systems.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

The partial blocking model can be solved exactly, but that solution does not scale well. We have therefore applied the approximation methodology and shown it to produce accurate results.

The exact solution of the complete blocking model is still an open problem, as is also the solution of the model where resources are provided in discrete servers, rather than in common pools. Those problems are interesting and worthy of further study. However, we feel that even if the exact solutions were available, it would be better to tackle the task of optimizing a real-life system by using the proposed approximations. They are easily implementable and sufficiently accurate.

Other, more general models may be tackled by the methods described here. For example, the complete blocking and partial blocking policies need not be mutually exclusive. One may wish to operate a mixed policy where some job types are subject to complete blocking, while others are accepted under partial blocking. The fixed-point approach and the large-scale simplifications would still apply.

References

1. Bodík, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., & Patterson, D. (2009). Statistical machine learning makes automatic control practical for internet datacenters. Conf. on Hot Topics in Cloud Computing (HotCloud'09), Berkeley, CA, USA.
2. Chao, X. & Miyazawa, M. (1998). On Quasi-reversibility and local balance: an alternative derivation of the product-form results. *Operations Research,* 46(6): 927–933.
3. Choudhury, G.L., Leung, K.K., & Whitt, W. (1995). Resource-sharing models with state-dependent arrivals of batches. In W.J. Stewart, (ed.), *Computations with Markov Chains*. Springer, Boston, MA, pp. 225–282.
4. Cromme, L.J. (1997). Fixed point theorems for discontinuous functions and applications. *Nonlinear Analysis: Theory, Methods and Applications,* 30(3): 1527–1534.
5. Dean, J. & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM,* 51(1): 107–113.
6. van Doorn, E.A. & Planken, F.J.M. (1993). Blocking probabilities in a loss system with arrivals in geometrically distributed batches and heterogeneous service requirements. *ACM/IEEE Transactions on Networking,* 1: 664–667.
7. Ezhilchelvan, P. & Mitrani, I. (2015). Static and dynamic hosting of cloud servers. In M. Beltran, W. Knottenbelt & J. Bradley, (eds.), *Computer Performance Engineering LNCS 9272*. Springer, Cham, pp. 19–31.
8. Ezhilchelvan, P. & Mitrani, I. (2017). Optimal provisioning of servers for hosting services of multiple types. *Simulation Modelling Practice and Theory,* 75: 17–28.
9. Ezhilchelvan, P. & Mitrani, I. (2017). Multi-class resource sharing with batch arrivals and complete blocking. 14th Int. Conf. on Quantitative Evaluation of SysTems (QEST 2017), Berlin.
10. Hampshire, R.C., Massey, W.A., Mitra, D., & Wang, Q. (2003). Provisioning for bandwidth sharing and exchange. In Anandalingam G., Raghavan S. (eds) Telecommunications Network Design and Management, vol. 23 of series *Operations Research/Computer Science Interfaces*, Springer, Boston, MA, pp. 207–225.
11. Harrison, P.G. (2004). Reversed processes, product forms and a non-product form. *Linear Algebra and its Applications,* 386: 359–381.
12. Herings, P.J.-J., van der Laanb, G., Talmanc, D., & Yang, Z. (2008). A fixed point theorem for discontinuous functions. *Operations Research Letters,* 36(1): 89–93.
13. Kaufman, J.S. (1981). Blocking in a shared resource environment. *IEEE Trans. Commun.,* 29: 1474–1481.
14. Kaufman, J.S. & Rege, K.M. (1996). Blocking in a shared resource environment with batched Poisson arrival processes. *Performance Evaluation,* 24: 249–263.
15. Kelly, F. (1986). Blocking probabilities in large cirquit switched networks. *Advances in Applied Probability,* 18: 473–505.
16. Mazzucco, M., Dyachuk, D., & Dikaiakos, M. (2010). Profit-aware server allocation for green internet services. IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 277–284.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press

17. Mazzucco, M., Vasar, M., & Dumas, M. (2012). Squeezing out the cloud via profit-maximizing resource allocation policies. IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 19–28.
18. Mitrani, I. (1998). *Probabilistic Modelling*. Cambridge University Press, Cambridge, England, UK.
19. Roberts, J.W. (1981). A service system with heterogeneous user requirement. In G. Pujolle, (ed.), *Performance of Data Communications Systems and Their Applications*. North-Holland, pp. 423–431.
20. Ross, K.W. (1995). *Multiservice Loss Models for Broadband Telecommunication Networks*. Springer-Verlag, Springer London.
21. Sadre, R., Haverkort, B.R., & Ost, A. (1999). An efficient and accurate decomposition method for open finite- and infinite-buffer queueing networks. Proceedings of the 3rd International Workshop on Numerical Solution of Markov Chains (Eds. W. Stewart and B. Plateau), pp. 1–20.
22. Tan, Y., Lu, Y., & Xia, C.H. (2012). Provisioning for large scale loss network systems with applications in cloud computing. *ACM SIGMETRICS Performance Evaluation Review,* 40(3): 83–85.
23. Whitt, W. (1983). The Queueing Network Analyzer. *Bell System Technical Journal*, 62(9): 2779–2815.
24. https://aws.amazon.com/ec2/instance-types/, Amazon Web Services, 2016.

https://doi.org/10.1017/S0269964818000323 Published online by Cambridge University Press