Static and Dynamic Hosting of Cloud Servers

Paul Ezhilchelvan and Isi Mitrani

School of Computing Science, Newcastle University, NE1 7RU, UK
e-mail: paul.ezhilchelvan@ncl.ac.uk, isi.mitrani@ncl.ac.uk

Abstract. The problem of maximizing the profit achieved by hiring servers from a Cloud and offering virtual machines to paying customers is examined. A number of VMs, each running a user job, can share a server. Hiring a server incurs an initial set-up cost, as well as running costs proportional to the duration of hire. New jobs that cannot start immediately may be lost, or they may be queued. It may or may not be possible to move running VMs from server to server. The effect of these different conditions on several hiring policies, both static and dynamic, is analyzed and evaluated.

1 Introduction

This paper addresses a problem that arises in the market for computer services. A host gains income by running user jobs on servers that it hires from a Cloud provider. To run a job, a Virtual Machine (VM) is instantiated on one of the servers. However, there is a limit on the number of VMs, and hence jobs, that can run in parallel on one server without unduly degrading each other's performance. When that limit is reached for all currently hired servers, the host may
(a) reject newly arriving jobs, thus losing revenue;
(b) queue newly arriving jobs, possibly having to pay penalties mandated by a Service Level Agreement (SLA);
(c) hire more servers, incurring more costs.

The cost of hiring a server may include a fixed initial set-up component, plus a cost proportional to the duration of hire. In the case of queued jobs, the SLA may guarantee a bound on waiting, with a penalty payable when that bound is exceeded. In all cases, the problem is to decide what actions to take so as to maximize the long-term average profit (revenues minus costs) obtained per unit time.

We analyze, evaluate and compare several server hiring policies. Some of these are static, choosing a fixed number of servers and keeping them for as long as the input parameters remain the same. Others are dynamic, hiring and releasing servers in response to changes in the number of jobs in the system. The majority of policies reject incoming jobs which cannot start immediately. However, the possibility of queueing such jobs subject to waiting time guarantees is also considered.

In a practical application, these policies would have to be combined with some monitoring and parameter estimation technique that would detect when the

loading parameters change. We do not dwell on that aspect because it has already been covered quite extensively in the literature (see below). Our assumption is that the system reaches steady state during a period where the parameters stay the same.

It should be pointed out that the behaviour of a dynamic policy depends on whether a running VM can be moved from one server to another or not. In the former case, jobs can be packed into the smallest number of servers required, whereas in the latter one may need to keep an unnecessarily large number of partially filled servers.

A special queueing model is analyzed and solved for the case of a dynamic hiring policy with non-movable VMs.

The general conclusion reached after a number of numerical experiments comparing the different hiring policies is that a static policy can perform really well, provided that it is chosen optimally (this proviso is important!). Dynamic policies do tend to produce higher profits, but the improvements rarely exceed 10%.

There has been quite a lot of work on server allocation, often in the context of the trade-off between performance and energy consumption. In most cases the focus has been on static policies, with an emphasis on estimating the traffic and reacting to changes in the parameters. Such studies were carried out by Mazzucco et al. [7, 8], using models and empirical observations. Bodík et al. [2] use statistical machine learning to estimate the workload during the next period.

Chaisiri et al. [3] attempt to exploit the lower costs of future reservations in order to minimize the overall cost of hiring Cloud servers. They use stochastic and deterministic programming techniques, coupled with approximations. This study has some dynamic features. However, the actual demand process is not modelled and therefore neither losses nor waiting can be taken into account.

A dynamic optimization using Markov decision theory was carried out by McGough and Mitrani [9] for a model with batch arrivals and also when hiring decisions are made at fixed intervals. Gandhi et al [5], and Mitrani [12] analyzed certain dynamic server allocation policies with set-up costs. In these studies jobs are queued but there are no SLAs, and the possibility of rejections is not considered.

More distantly related work concerns the maximization of throughput and the minimization of waiting or response time in different scheduling contexts, e.g. Urgaonkar et al. [13], Chandra et al. [4] and Bennani and Menascé [1]. A deterministic example of job scheduling with migration in order to minimize the number of servers was considered by Ghribi et al [6].

In all of the above papers, servers are assumed to serve one job at a time (VMs are mentioned in [8] for the purpose of parameter estimation, but are not included in the analysis). Where a dynamic policy has been compared to a static one (e.g. in [9]), the latter has been chosen in an ad-hoc manner, rather than optimally. The effect of not being able to move VMs between servers has not been examined.

Section 2 introduces a number of static and dynamic policies and evaluates the profit they achieve. The models involve job losses and also queueing. The dynamic policies assume that VMs can move instantaneously from one server to another. The model of a dynamic policy that does not move VMs is analyzed in section 3. Section 4 summarizes the conclusions and outlines some directions for further research.

2 Static and dynamic policies

A host hires servers from a cloud provider in order to offer services to paying customers, Servers can be hired and released instantaneously and at any time. Similarly, VMs can be initiated and terminated instantaneously and at any time. In this section we also assume that VMs can be moved from server to server without delay and without incurring costs.

The service provided by a VM during its lifetime is referred to as a 'job'. A server can run efficiently up to m parallel VMs, so if there are n active servers at a given moment, there is room for a maximum of nm jobs.

The cost of a server which is used for a period of length t is $c_1 + c_2 t$. The first term, c_1, if non-zero, may be considered as a 'set-up' cost, or it may be introduced by the provider in order to discourage short-term hire. The coefficient c_2 reflects the cost of operating a server per unit time.

Jobs arrive into the system in a Poisson stream at rate λ. Their lifetimes may have arbitrary distribution with mean $1/\mu$. The offered load is thus $\rho = \lambda/\mu$. The values of these parameters are assumed to remain constant long enough so that the system can be treated as being in steady state.

The assumption of easily movable VMs implies that jobs can be 'packed' efficiently. Suppose that the servers currently hired are numbered $1, 2, \ldots$. When accepting an incoming job, allocate it to the server with the lowest index that has room for it. When a job is completed and its VM is terminated, move a job from the non-empty server with the highest index (if different) to the vacated place. This ensures that if there are j jobs present, they can be run on $\lceil j/m \rceil$ servers, where $\lceil x \rceil$ is the smallest integer exceeding or equal to x.

The problem that needs to be addressed in this context is: When, and how many, servers should be hired or released? One possibility is to employ a static policy whereby a fixed number of servers, n, is hired and kept for as long as the parameters λ and μ retain their values. An incoming job that finds all servers full, i.e. nm jobs present, is rejected. If the policy is static, the question of moving jobs between servers does not arise.

In such a system, the number of jobs present behaves like an $M/M/K/K$ queue, i.e. an Erlang loss model where $K = nm$ is the maximum number of jobs that can be accepted. The steady-state probability, q_j, that there are j jobs present is equal to (e.g., see [11])

$$q_j = \frac{\rho^j}{j!} \left[\sum_{k=0}^{nm} \frac{\rho^k}{k!} \right]^{-1} \quad ; \quad j = 0, 1, \ldots, nm \; . \tag{1}$$

The decision on what value of n to choose depends on the objective function to be optimized. If, for example, the aim is simply to avoid job losses, one could fix a desirable value, ε, and hire the smallest number of servers which ensures that the probability of rejection does not exceed ε:

$$n^* = \min\{n \ : \ q_{(nm)} \leq \varepsilon\} \ . \tag{2}$$

This will be referred to as the 'fix-ε' policy.

Alternatively, one could attempt to maximize profit. Suppose that every accepted job brings in a revenue of r. Then the average long-run profit produced by n servers per unit time, $R(n)$, is given by

$$R(n) = r\lambda(1 - q_{(nm)}) - c_2 n \ , \tag{3}$$

The long-run set-up costs incurred per unit time are zero, because after the initial moment there are no new hiring events.

It is known that the Erlang loss probability, $q_{(nm)}$, is convex in n (see [10]). Hence, the profit function $R(n)$ is concave in n and has a single maximum. The optimal number of servers, and the corresponding maximum achievable profit, can therefore be computed quite easily by evaluating $R(n)$ for $n = 1, 2, \ldots$, and stopping as soon as $R(n+1) < R(n)$. The resulting hiring policy will be referred to as 'fix-opt'.

Now consider the possibility of hiring and releasing servers dynamically, in response to changes in the system state. A rather general policy of this type would work as follows: Hire a block of k_1 servers; if there are $k_1 m$ jobs present and a new job arrives, hire a new block of k_2 servers. This goes on up to a maximum of b blocks with a total of $n = k_1 + k_2 + \ldots + k_b$ servers. When a job completes, a job from a block with a higher index (if any) is moved into its place so as to maintain optimal packing. If, as a result of this completion, a block empties, all the servers in it are released. One may also decide to keep block 1 permanently hired. This policy will be referred to as 'blocks-b', with a bound of n.

The number of jobs in the system under the blocks-b policy with bound n has the same distribution, given by (1), as under the static policy with n servers. In particular, the probability that an incoming job is accepted is the same. However, the dynamic policy incurs set-up costs, while reducing the operating costs.

Let K_i be the total number of servers in the first i blocks: $K_i = k_1 + k_2 + \ldots + k_i$; $i = 1, 2, \ldots, b$; $K_b = n$ and, by definition, $K_0 = 0$. Since block $i+1$ is hired whenever an incoming job finds exactly $K_i m$ jobs present, the average number of hiring events per unit time is

$$S = \lambda \sum_{i=0}^{b-1} q_{(K_i m)} k_{i+1} \ . \tag{4}$$

For a given number, j, of jobs present ($1 \leq j \leq nm$), denote by $K(j)$ the number of servers currently hired. That is the smallest K_i such that $K_i \geq \lceil j/m \rceil$.

With that notation, the average number of servers hired, L, can be written as

$$L = \sum_{j=1}^{nm} q_j K(j) \ . \tag{5}$$

Hence, the average long-term profit obtained per unit time under the blocks-b policy is equal to

$$R(n) = r\lambda(1 - q_{(nm)}) - c_1 S - c_2 L \ . \tag{6}$$

To determine the best blocks-b policy with a bound n, one would have to search not only with respect to n, but also with respect to b and k_i. That search may be quite expensive. However, two special cases can be handled quite easily. At one extreme is the policy which we shall call 'one-by-one': it hires and releases servers one at a time ($b = n$ and $k_i = 1$ for all i). The best one-by-one policy can be found by a simple search with respect to n. At the other extreme is blocks-2, where only two blocks are used; $b = 2$, $k_1 + k_2 = n$. The search in this case is with respect to n and k_1.

Figure 1 illustrates the performance of the above policies in the context of a system where each server runs up to 5 VMs in parallel. The average residence of a VM is taken as the unit of time, $\mu = 1$. The revenue per job is $r = 1$, while the server costs are $c_1 = 0.1$ and $c_2 = 3$. Thus, a server can make a profit by running 5 jobs, but the margin is not large.

For the policies fix-opt, one-by-one and blocks-2, the profit produced by the best server bound n (and, in the case of blocks-2, the best block sizes k_1 and k_2, all determined by a search), is plotted against the offered load by increasing the arrival rate. For the fix-eps policy, the value of n is chosen so that no more than one job in a thousand is lost, $\varepsilon = 0.001$.

We observe that the fix-ε policy has the worst performance, actually losing money unless the arrival rate exceeds 30. This is not surprising, since the value of ε is quite small, and the costs of servers are disregarded when choosing n. Of the dynamic policies, one-by-one is better than blocks-2, but not by much. Both are better than the static fix-opt policy, but again not by much.

The server allocations for each value of λ are shown in the table below (the values of λ now go up to 100). n^* is the best number of servers found for the fix-opt policy; $n1^*$ is the best upper bound for the one-by-one policy; k_1 and k_2 are the best block sizes for the blocks-2 policy. Note that $n1^*$ is always larger than n^*, while k_1 (at least in this example) is the same as n^*. The improvement in profit produced by the blocks-2 policy is due to the small second block which is brought into play when the first block is full.

The above behaviour is observed for other parameter values, as long as the set-up costs are quite small compared to the operating costs. When that is not the case, the comparison is less clear-cut. This is illustrated in Figure 2, where the performance of the fix-opt, one-by-one and blocks-2 policies is plotted against the set-up cost c_1. The job arrival rate is $\lambda = 40$ and the other parameters are as in figure 1.

Since the fix-opt policy is not affected by the set-up cost, its plot is a horizontal line. The one-by-one policy starts off as the best of the three, but eventually

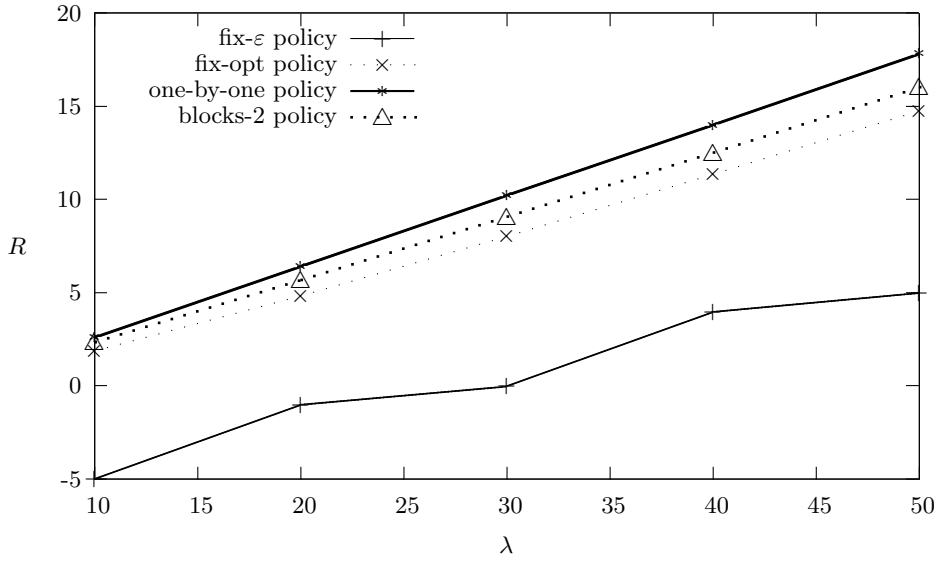

Fig. 1. Comparison of hiring policies
$m = 5,\ \mu = 1,\ r = 1,\ c_1 = 0.1\ c_2 = 3$

Table 1. Server allocations

λ	n^*	$n1^*$	k_1	k_2
20	4	14	4	1
40	8	20	8	1
60	12	27	12	1
80	16	33	16	2
100	20	39	20	2

becomes the worst. This is because it keeps hiring and releasing servers even when that is not warranted by the high set-up costs. The blocks-2 policy is more conservative. It yields slightly lower profits than one-by-one when c_1 is low, but on the other hand it never becomes worse than fix-opt as c_1 increases. What happens is that for high values of c_1, the best blocks-2 policy is of the form $k_1 = n$, $k_2 = 0$. In other words, it becomes identical to fix-opt.

The conclusions that can be drawn from these results, as well as from others derived with different parameter values and cost coefficients, can be summarized as follows:

The optimally chosen static policy fix-opt, which does not incur repeated set-up costs and does not require moving VMs from server to server, performs very well. The dynamic policies from the blocks-b family can achieve 10% – 15%

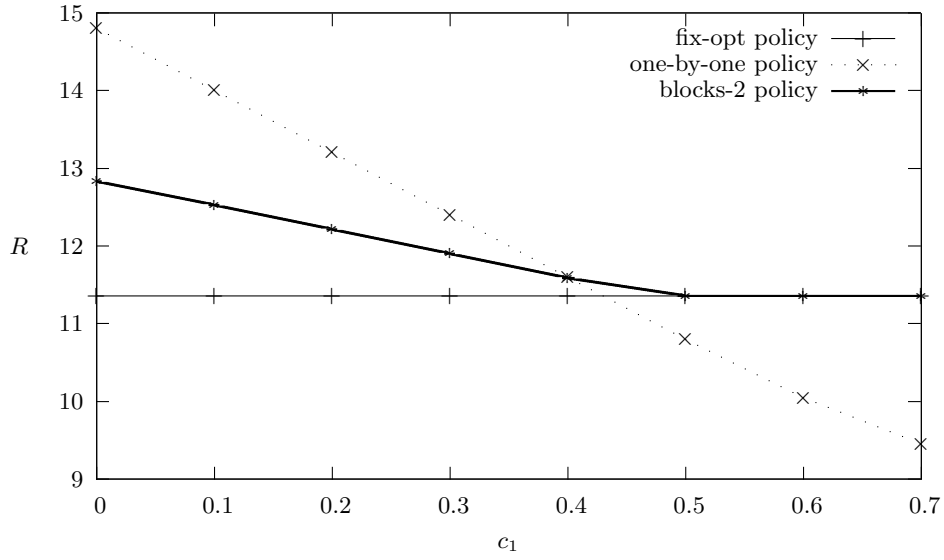

Fig. 2. Comparison of hiring policies; different set-up costs
$m = 5,\ \mu = 1,\ r = 1,\ \lambda = 40\ c_2 = 3$

higher profits when the set-up costs are low. The best blocks-2 policy is always at least as good as the fix-opt policy and is never much worse than one-by-one.

2.1 Queued jobs

Instead of rejecting the jobs that find all available VMs occupied, it may be possible to queue them. That would avoid the revenue loss due to rejections, but would raise the question of quality of service. Since customers do not like waiting, the host would normally offer some *Service Level Agreement* (SLA), e.g. promising to pay a penalty u for any job whose waiting time exceeds a given threshold, v.

Would that be worth doing? To evaluate the trade-off, assume that the service times are distributed exponentially. Consider the static policy that hires a fixed number of servers, n, and queues jobs. The long-run average profit produced per unit time is now

$$R(n) = \lambda[r - uP(w > v)] - c_2 n \ , \tag{7}$$

where $P(w > v)$ is the steady-state probability that the waiting time in the $M/M/(nm)$ queue exceeds v. For a stable queue ($\rho < nm$), that probability is given by (e.g., see [11])

$$P(w > v) = qe^{-\mu(nm-\rho)v} \ , \tag{8}$$

where q is the steady-state probability that an incoming job would have to wait:

$$q = \frac{\rho^{nm}}{(nm-1)!(nm-\rho)} \left[\sum_{j=0}^{nm-1} \frac{\rho^j}{j!} + \frac{\rho^{nm}}{(nm-1)!(nm-\rho)} \right]^{-1} . \qquad (9)$$

This last expression is known as the 'Erlang-C formula', or 'Erlang's delay formula'.

The policy that uses the value of n which maximizes the right-hand side of (7) will be referred to as fixQ-opt. The trade-off between fix-opt and fixQ-opt would clearly depend on the SLA. Intuitively, queueing jobs is likely to be advantageous if customers are willing to put up with waiting, otherwise rejections would be better.

In figure 3, the fixQ-opt policy is compared with fix-opt for three different thresholds v. To make the queueing policy more directly comparable to the rejection one, the penalty u is taken to be equal to r. In other words, customers whose jobs wait longer than v get their money back. From the profit perspective, it is as if they had been rejected. The other parameters are as in figure 1. The threshold values chosen are $v = 0.2$, $v = 0.4$ and $v = 0.6$. In other words, the penalty is payable if the customer's waiting time is more than 20%, 40% or 60% of their residence time. In all cases, the achieved average profit is plotted against the arrival rate, λ.

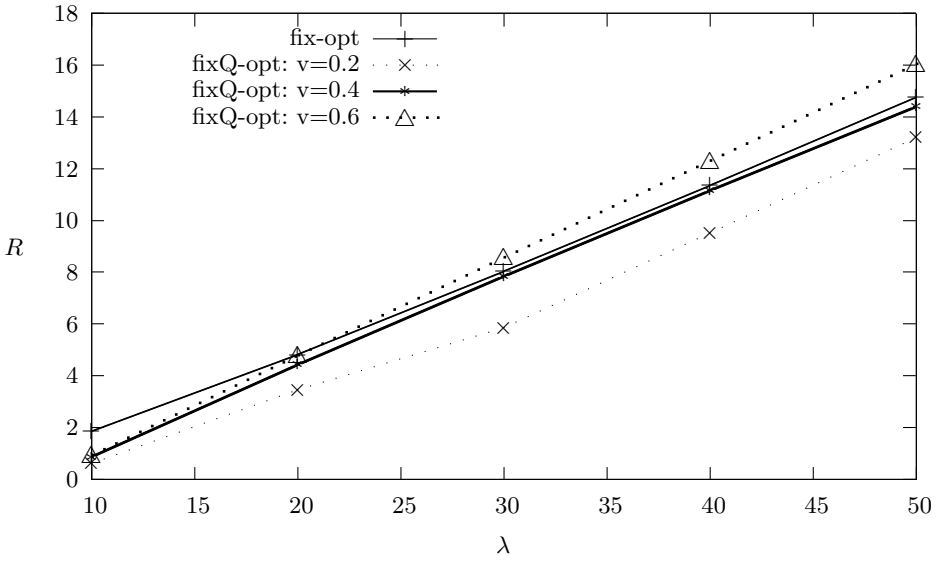

Fig. 3. Rejection vs queueing; different thresholds v
$m = 5,\ \mu = 1,\ r = u = 1,\ c_2 = 3$

We observe that when customers are impatient ($v = 0.2$), it is better to reject jobs than to queue them. The situation is reversed when customers are quite tolerant of waiting ($v = 0.6$) and the load exceeds 20. However, the differences are not large in either case. For the intermediate threshold ($v = 0.4$), there is even less to choose between queueing and rejecting. Moreover, increasing v beyond 0.6 does not improve the profits achieved by fixQ-opt significantly.

In summary, one can say again that, as long as the number of servers is chosen optimally, the management of the jobs is of secondary importance.

N.B. The Markovian assumptions of Poisson arrivals and exponential service times can be relaxed, at the price of replacing the exact results with approximations. There are approximate results for the $G/GI/n/n$ loss model, as well as for the $G/GI/n$ queue. To use those expressions one would need to estimate not only the average interarrival and service times, but also their second moments.

3 Virtual machines do not move

The dynamic policies examined in the last section relied for their operation on the ability to move virtual machines from server to server instantaneously and without cost. It would be interesting to evaluate the extent to which profits are affected when that flexibility is removed. In order to do that, we shall assume that the job service times are distributed exponentially, with mean $1/\mu$.

Consider the dynamic policy blocks-2, where n servers are divided into two blocks, 1 and 2, of sizes k_1 and k_2 respectively ($k_1 + k_2 = n$). A VM, once started in a server, remains there until its job completes. Let us assume that the servers in block 1 are permanently hired. Incoming jobs are allocated VMs in block 1 whenever there are fewer than $k_1 m$ jobs present there; otherwise they go to block 2, if there are fewer than $k_2 m$ jobs there; when all nm VMs are busy, jobs are rejected. The servers in block 2 are released when a departure leaves all of them empty; they are re-hired at the next arrival instant which finds block 1 full.

A single integer - the total number of jobs present - is no longer enough to describe the state of the system. One needs to specify the number of jobs, I, present in block 1, and the number of jobs, J, present in block 2. Those two random variables are not independent: J can increase only when $I = k_1 m$. Let $p_{i,j}$ be the joint steady-state probability that $I = i$ and $J = j$ ($i = 0, 1, \ldots, k_1 m$; $j = 0, 1, \ldots, k_2 m$).

The servers in block 2 are being used whenever $J > 0$. Hence, the average number of servers hired, L, is given by

$$L = k_1 + k_2 \left[1 - \sum_{i=0}^{k_1 m} p_{i,0} \right] . \tag{10}$$

The hiring of block 2 occurs whenever an incoming job finds the system in state $I = k_1 m$, $J = 0$. Hence, the average number of server hiring events per unit time, S, is

$$S = \lambda k_2 p_{k_1 m, 0} . \tag{11}$$

Jobs are rejected when both blocks are full, which occurs with probability p_{k_1m,k_2m}. The average profit obtained per unit time is thus given by an expression similar to (6):

$$R(n) = r\lambda(1 - p_{k_1m,k_2m}) - c_1S - c_2L \ . \tag{12}$$

It now remains to determine the joint distribution $p_{i,j}$. The instantaneous transition diagram for the Markov process (I, J) is illustrated in figure 4.

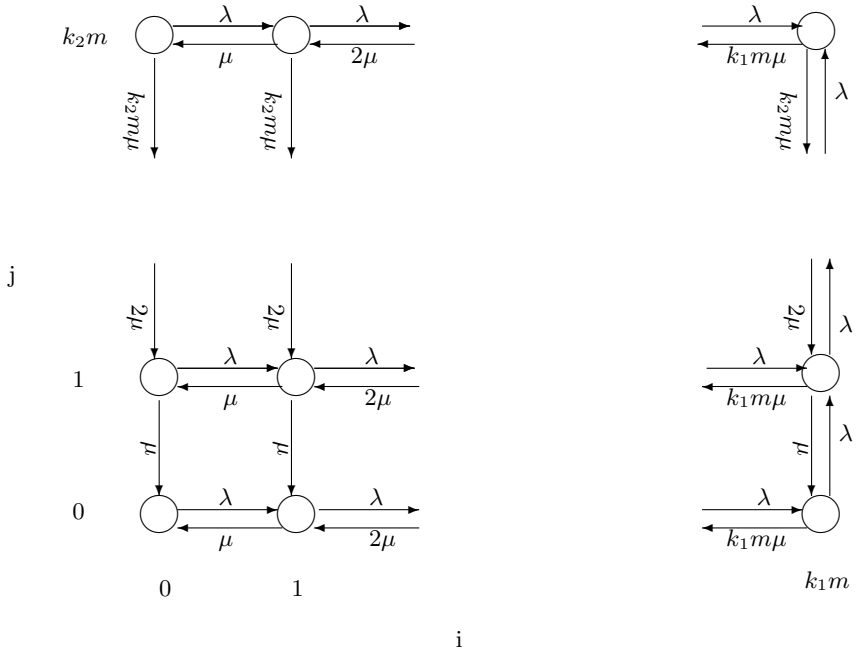

Fig. 4. Transition diagram

For $i = 0, 1, \ldots, k_1m - 1$ and $j = 0, 1, \ldots, k_2m$, the probabilities $p_{i,j}$ satisfy the following balance equations:

$$(\lambda + i\mu + j\mu)p_{i,j} = \lambda p_{i-1,j} + (i+1)\mu p_{i+1,j} + (j+1)\mu p_{i,j+1} \ , \tag{13}$$

where $p_{-1,j} = 0$ and $p_{i,k_2m+1} = 0$ by definition. When $i = k_1m$, the equations become:

$$(\lambda + k_1m\mu + j\mu)p_{k_1m,j} = \lambda p_{k_1m-1,j} + \lambda p_{k_1m,j-1} + (j+1)\mu p_{k_1m,j+1} \ , \tag{14}$$

where $p_{k_1m,k_2m+1} = 0$ by definition.

The numerical complexity of solving this set of simultaneous equations, plus the normalizing equation, by Gaussian elimination, can be high. It is on the order

of $O[(k_1m+1)^3(k_2m+1)^3]$. Fortunately, we can exploit the special structure of this Markov process in order to reduce that complexity considerably.

Note first that the total number of jobs present, $I+J$, behaves like the number of calls in an Erlang loss system $M/M/nm/nm$ with offered traffic $\rho = \lambda/\mu$. In particular, the rejection probability, p_{k_1m,k_2m}, is equal to the probability q_{nm}, given by expression (1) for $j = nm$.

Examining equations (13) for $j = k_2m$ and $i = 0, 1, \ldots, k_1m-1$ in turn, we see that they can be transformed into recurrence relations. Denoting k_2m by K, these can be written as

$$p_{i,K} = a_{i+1}p_{i+1,K} \;\; ; \;\; i = 0, 1, \ldots, k_1m-1 \; , \tag{15}$$

where $a_0 = 0$ and

$$a_{i+1} = \frac{i+1}{(1-a_i)\rho + i + K} \;\; ; \;\; i = 0, 1, \ldots, k_1m-1 \; . \tag{16}$$

After evaluating p_{k_1m,k_2m} and the coefficients a_i, the recurrences (15) enable the probabilities, $p_{k_1m-1,K}$, $p_{k_1m-2,K}$, ..., $p_{0,K}$ to be computed by successive substitution. Denote the sum of those probabilities, i.e. the marginal probability that block 2 is full, by $p_{\cdot,K}$. Equating the rate at which the number of jobs in block 2 decreases from K to $K-1$ with that at which it increases from $K-1$ to K, we obtain

$$K\mu p_{\cdot,K} = \lambda p_{k_1m,K-1} \; . \tag{17}$$

This equation determines $p_{k_1m,K-1}$. In general, if the marginal probability, $p_{\cdot,j+1}$, that there are $j+1$ jobs in block 2 is known, then the probability $p_{k_1m,j}$ is determined from

$$(j+1)\mu p_{\cdot,j+1} = \lambda p_{k_1m,j} \;\; ; \;\; j = 0, 1, \ldots, K-1 \; . \tag{18}$$

Now consider the balance equations corresponding to row j in the diagram, for $j = 0, 1, \ldots, K-1$. They can be written in the form

$$p_{i,j} = a_{i+1,j}p_{i+1,j} + b_{i+1,j} \;\; ; \;\; i = 0, 1, \ldots, k_1m-1 \; , \tag{19}$$

with $a_{0,j} = 0$,

$$a_{i+1,j} = \frac{i+1}{(1-a_{i,j})\rho + i + j} \;\; ; \;\; i = 0, 1, \ldots, k_1m-1 \; , \tag{20}$$

and $b_{0,j} = 0$,

$$b_{i+1,j} = \frac{\rho b_{i,j} + (j+1)p_{i,j+1}}{(1-a_{i,j})\rho + i + j} \;\; ; \;\; i = 0, 1, \ldots, k_1m-1 \; . \tag{21}$$

Having determined the probabilities in row $j+1$, and consequently $p_{k_1m,j}$ from (18), these relations determine all the other probabilities in row j.

Proceeding in this manner through rows $K-1$, $K-2$, ..., 0, one can compute all unknown probabilities. The numerical complexity of that procedure is on the order of $O[(k_1m+1)(k_2m+1)]$, i.e. it is linear in the number of unknowns.

It may be expected that the inability to move VMs and pack them into the smallest number of servers would reduce the effectiveness, and hence the profits, of a dynamic hiring policy. The extent of that reduction is illustrated in figure 5. The blocks-2 policy of this section is compared with the fix-opt and blocks-2 policies of section 2, where VMs could move instantaneously from server to server. For both versions of the dynamic policy, the block sizes k_1 and k_2 are chosen optimally, by performing a search. In all cases, the achieved profit is plotted against the arrival rate λ, while the other parameters are as in figure 1.

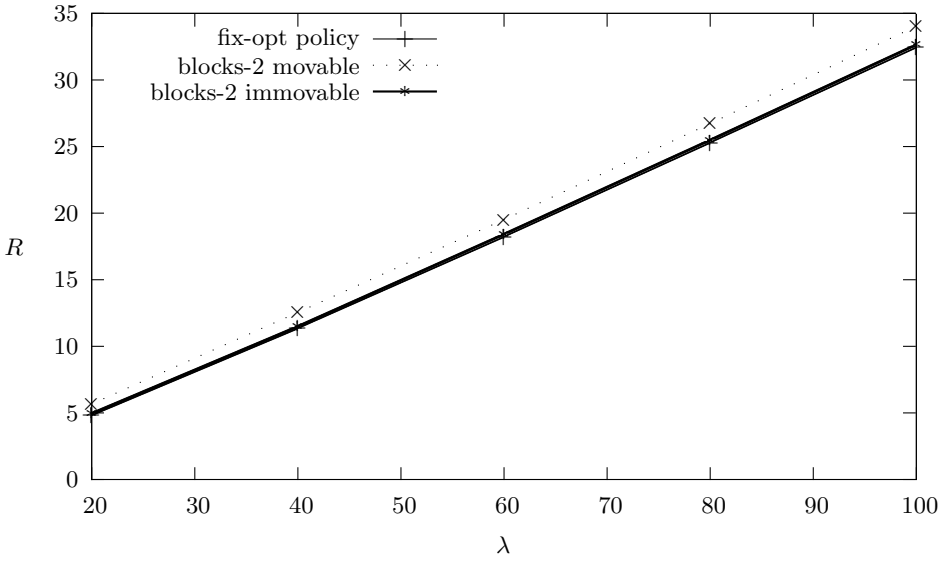

Fig. 5. Static and dynamic policies with movable and immovable VMs
$m = 5,\ \mu = 1,\ r = 1,\ c_1 = 0.1\ c_2 = 3$

The figure shows that when VMs cannot be moved, the advantage of dynamic hiring is almost eliminated. The blocks-2 policy with immovable VMs still produces slightly higher profits than the fix-opt policy, but the improvements are on the order of 1%.

4 Conclusion

We have evaluated and compared the performance of several static and dynamic server hiring policies. The results of a number of numerical experiments lead

to the rather general conclusion that a well-chosen static policy can be nearly as good as a dynamic one. Moreover, either an increase in set-up costs, or a restriction in the movement of VMs, tends to reduce the gains achieved by a dynamic policy. That will be the case, for example, if VMs can be moved, but not instantaneously. The costs and durations of moving VMs were examined in more detail in Voorsluys et al [14].

Among the dynamic hiring policies, the simple blocks-2 policy may be recommended. The best block sizes are quite easily determined, and it is robust with respect to rising set-up costs. However, its advantages are significantly reduced if VMs cannot move from server to server.

If the incoming jobs belong to different classes, with different parameters, and if servers are hired and dedicated to separate classes, then our results would apply to each individual class. The situation would be more complicated if VMs of different classes were allocated to the same server. That would require further work.

What if jobs do not arrive in a Poisson stream? Some answers may be obtained by applying approximate results, but those would probably need to be validated by simulations.

References

1. M.N. Bennani and D. Menascé, "Resource allocation for autonomic data centers using analytic performance methods", Procs., 2nd IEEE Conf. on Autonomic Computing (ICAC-05), pp 229-240, 2005.
2. P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters", Conf. on Hot Topics in Cloud Computing (HotCloud'09), Berkeley, CA, USA, 2009.
3. S. Chaisiri, B.S. Lee and D. Niyato, "Optimization of resource provisioning cost in cloud computing", IEEE Transactions on Services Computing, 5(2), pp. 164–177, 2012.
4. A. Chandra, W. Gong and P. Shenoy, "Dynamic resourse allocation for shared data centers using online measurements", Procs., 11th ACM/IEEE Int. Workshop on Quality of Service (IWQoS), pp 381-400, 2003.
5. A. Gandhi, M. Harchol-Balter and I. Adan, "Server farms with setup costs", Performance Evaluation, 67, 11, pp. 1123-1138, 2010.
6. C. Ghribi, M. Hadji and D. Zeghlache, "Energy Efficient VM Scheduling for Cloud Data Centers: Exact Allocation and Migration Algorithms", 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 671-678, 2013.
7. M. Mazzucco, D. Dyachuk, and M. Dikaiakos, "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, 2010.
8. M. Mazzucco, M. Vasar, and M. Dumas, "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, 2012.
9. A.S. McGough and I. Mitrani, "Optimal Hiring of Cloud Servers", Procs, European Performance Engineering Workshop (EPEW), Florence, 2014.

10. E.J. Messerli, "Proof of a convexity property of the Erlang B formula", *Bell System Technical Journal* 51, pp. 951-953, 1972.
11. I. Mitrani, *Probabilistic Modelling*, Cambridge University Press, 1998.
12. I. Mitrani, "Trading Power Consumption Against Performance by Reserving Blocks of Servers", Procs, European Performance Engineering Workshop (EPEW), Munich, 2012.
13. R. Urgaonkar, U. C. Kozat, K. Igarashi and M. J. Neely, "Dynamic Resource Allocation and Power Management in Virtualized Data Centers", IEEE/IFIP NOMS 2010, Osaka, Japan, 2010.
14. W. Voorsluys, J. Broberg, S. Venugopal and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation", LNCS, vol. 5931, pp. 254-265, 2009.