A Resource Leasing Policy for On-demand Computing

Darin England and Jon Weissman
Department of Computer Science and Engineering
University of Minnesota, Twin Cities
{england,jon}@cs.umn.edu

Abstract

Leasing computational resources for on-demand computing is now a viable option for providers of network services. Temporary spikes or lulls in demand for a service can be accommodated by flexible leasing arrangements. From the service provider's perspective the problem is how many resources to lease and for how long. In this work we formulate and solve the resource leasing problem for the case of a single service. The objective is to minimize the cost of leasing resources while still maintaining an adequate quality of service, which we measure by the average wait time of requests. Demand for the service and execution times of service requests are modelled as random variables. The problem is formulated as continuous-time, infinite-horizon Markov decision problem. We use the dynamic programming method of value iteration for its solution and we characterize the resulting optimal cost function. We find that the cost of providing a service is convex-like in the number of resources leased and nondecreasing in the number of requests in the system. Close examination of the optimal cost function shows that the cost of providing a service is more sensitive to under-deployment than to over-deployment. Thus, when demand for the service is known to exist, but is unpredictable, it is better to lease more resources than fewer resources.

1 Introduction

Creators of software applications are increasingly making use of network services that provide specific functionality to their applications. Mathematical programming, graphics rendering, and gene sequence comparison represent some important computation-intensive applications that are now being offered as on-line services. In order to accommodate this new computing paradigm the research community has created several service-oriented architectures [11, 17, 19–21]. Researchers have focused particular attention on the area of grid services [1, 8, 16, 18] wherein the operating environment is known to be heterogeneous and unpredictable. Perhaps not coincidentally, makers of high-performance computers have recently begun to offer on-demand computing solutions, also known as utility computing. The idea is that these companies act as resource providers, owning the computational resources and performing all system administration tasks, while charging service providers nominal fees for access to computing cycles. Such a scenario is attractive to service providers that experience temporary spikes in demand for their service. On-demand computing obviates the need for purchasing computational infrastructure since leasing arrangements provide the needed computing capacity at low cost and low risk.

In this work we consider the resource leasing problem from the perspective of the service provider. The nature of the problem is such that the demand for the service and the processing times of individual requests are unknown, but can be modelled with appropriate probability distributions. Still, the service provider is faced with the conflicting goals of leasing enough computational resources to provide an adequate level of service and keeping the cost of leasing to a minimum. Leasing too many resources to host a service incurs unnecessary cost. However, leasing too few resources results in long wait times and client dissatisfaction. In our model we place a cost on the average wait time of client requests. Such a cost can represent a loss in revenue or a loss in goodwill. Due to its stochastic nature we model and solve the resource leasing problem in a dynamic programming framework. Given the cost structure of the lease arrangement, the resulting policy provides state-dependent rules for service providers to acquire on-demand resources and to terminate those leases when it is cost-effective to do so. We characterize the optimal cost function and we find that the cost is more sensitive to leasing too few resources than to leasing too many resources. This indicates that it is better for a service to be slightly over-deployed than under-deployed.

In the next section we describe a mechanism for on-demand, dynamic deployment of network services, one with which we have experimented. Section 3, which constitutes the bulk of the paper, is the dynamic programming formulation of the resource leasing problem. We then describe the solution method in section 4. Section 5 contains experimental results for an example problem. We discuss related work in section 6 , and give some concluding remarks in section 7.

2 On-demand Deployment of Network Services

A key assumption for this work is the ability for a service provider to dynamically deploy a service onto the network. This is the nature of on-demand computing: a quasi-real-time response to increased demand for a service. One architecture that allows such dynamic deployment of services is described in [18]. Figure 1 shows a simplified logical view of the architecture. Requests for the service form a queue at the front-end and are dispatched to leased computing nodes. Initially the network service is not present on the leased resources. When demand for the service increases to the point where a new lease is warranted, an archive file that contains the service code and required libraries is shipped to the node and deployed into a running servlet engine. The front-end may then route service requests to the newly leased node. Thus only basic service-oriented infrastructure is required to be in place on the computing nodes, i.e. a running servlet engine. The package sizes and transfer times for dynamic deployment of network services are discussed in [18]. We mention here that the overhead associated with such deployment, i.e. package transfer and configuration, is acceptable. For example, packages sizes less than 1MB require less than 1 second for configuration in a running servlet engine. Configuration times for package sizes over 1MB on are the order of seconds. However, note that each deployment of the service is long-lived. Thus the overhead will be amortized over multiple requests.

As discussed in the introduction, the decision to be made by the service provider is the number of nodes to lease. If too few resources are leased then waiting time increases and clients become dissatisfied. The cost associated with this dissatisfaction could be monetary if clients are paying for the service or it could be a loss in goodwill if the service is being provided by an institution that has committed to serving its user base. Finally, we note that the method used to assign service requests to computing nodes is orthogonal to this work. We address the issue of resource requirements. The only assumption regarding the scheduling discipline is that it is work-conserving, i.e. there are no idle nodes under lease whenever there are requests in the queue.

3 Problem Formulation

Due to the uncertainty associated with demand and processing times, the problem of how many resources to lease for hosting a network service is essentially a stochastic decision problem. Therefore, we chose to model the problem in a dynamic programming context. In particular, we formulate an infinite-horizon, continuous-

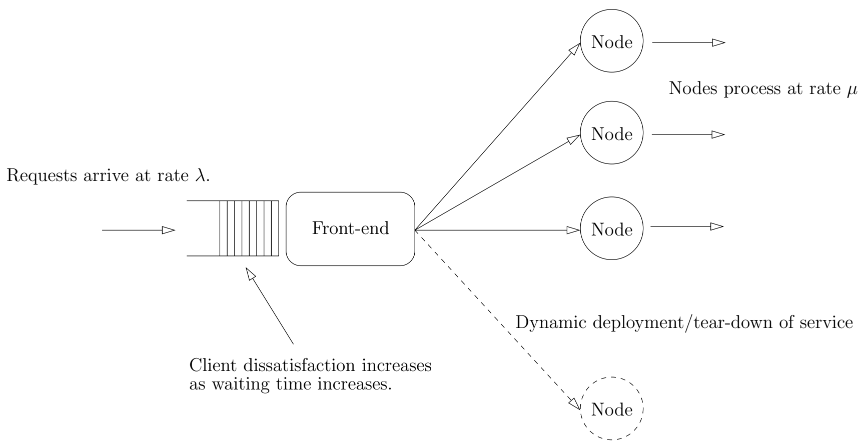
Node
Node
Node
Node
Nodes process at rate μ
Requests arrive at rate λ.
Front-end
Dynamic deployment/tear-down of service
Client dissatisfaction increases as waiting time increases.

Figure 1: Dynamic Deployment of a Network Service

time Markov decision problem with nonuniform transition rates among the states. Infinite-horizon methodology is used because the network service is to be deployed for an indefinite period of time: we assume that the service will be utilized for a long time or until it is replaced by a newer version. A finite-horizon version of the problem in which a service is to be deployed for a known, relatively short period of time is modelled and solved in [7]. In this work we assume that the service provider may initiate a lease at any point in time. Hence, the time between leasing decisions is random and depends on the demand for the service and the computational load imposed by the service requests. For this reason the problem is modelled as a continuous-time problem. The rate of transition from one state to another is nonuniform because it depends on the state and the leasing decision. We employ the well-known technique of uniformization [2, 3, 5, 14, 15] to convert the problem into a discrete-time Markov decision problem. We may then employ the standard dynamic programming method of value iteration to solve the problem.

3.1 State Space

The state of the system at any point in time is described by a pair of variables (x, m). x represents the number of service requests that are currently in the system, which includes both requests in process and requests in the queue. This component of the state is random and is a function of the arrival rate of service requests and the processing rate of the computational resources. m represents the number of resources currently under lease and on which the service is currently deployed. This component of the state is not random and is under the control of the decision-maker. Indeed, given that the service is currently deployed on m computational resources, the control variable u is the number of *additional* resources that should be leased and on which the service should be deployed. We note here that the value of u could be negative, in which case the decision is to tear-down the service and terminate the leases on $|u|$ resources. The inclusion of m as part of the state is necessary because there is a cost for setting up and taking down the service on a newly leased resource. In effect, including the number of currently leased resources as part of the state prevents the policy from thrashing between deployment and tear-down of the service. We discuss the components of cost in Section 3.3.

We place a limit K on the maximum number of service requests that can be in the system simultaneously. If the system is handling K requests at the time of a request arrival, then the request is rejected. In practice K may be set to a large number. We also assume a maximum number of resources, R, are available for lease. Thus the number of possible states is finite.

3.2 State Transition Probabilities

3.2.1 One-step Transition Probabilities

Transitions of the random component of the state x occur when a new request arrives or when a request finishes execution. We model the arrival of requests as a Poisson process with rate λ. The processing time of a single request is exponentially distributed with parameter μ. In effect, the system can be modelled as an $M/M/m/K$ queueing system; however, the number of servers m may change at each decision period. Given that the service is deployed on m resources, the one-step transition rates for the random component of the state, x, are shown in Figure 2. Let $p_{i,j}(u)$ be the probability of going from i requests in the system to j requests in the system when the leasing decision u is taken. Then the one-step transition probabilities are given by

$$p_{i,j}(u) = \begin{cases} \frac{\lambda}{\lambda+i\mu} & \text{if } i = j-1 \text{ and } i < m, \\ \frac{i\mu}{\lambda+i\mu} & \text{if } i = j+1 \text{ and } i < m, \\ \frac{\lambda}{\lambda+m\mu} & \text{if } i = j-1 \text{ and } i \geq m, \\ \frac{m\mu}{\lambda+m\mu} & \text{if } i = j+1 \text{ and } i \geq m, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

When writing transition probabilities we use the traditional notation of i and j (instead of x) for specifying the number of requests in the system.

3.2.2 Uniformization

From Figure 2 and our knowledge of the $M/M/m/K$ queue we note that the time between state transitions is exponentially distributed and depends on the number of requests currently in the system. In addition, the number of currently leased resources is itself part of the state and changes with the control u. Thus, the time from one state transition to the next is exponentially distributed with a parameter that depends on both the state and the control. Let the rate of transition out of a state (x, m) when control u is applied be denoted by $\nu_{x,m}(u)$. Furthermore, denote the time of the nth transition by t_n and the number of requests in the system just after the nth transition by $x(t_n)$. Then the distribution of transition times between states is given by

$$P\{t_{n+1} - t_n < \tau \mid x, m, u\} = 1 - e^{-\nu_{x,m}(u)\tau},$$

and the one-step transition rates are given by

$$\nu_{x,m}(u) = \begin{cases} \lambda & \text{if } x = 0, \\ \lambda + x\mu & \text{if } 0 < x < m, \\ \lambda + m\mu & \text{if } x \geq m. \end{cases}$$

The times between state transitions are exponentially distributed and therefore these times possess the memoryless property. That is, there is no benefit in knowing how much time has elapsed since the last transition. Thus, we do not include the time since the last transition as a part of the state, which makes the problem tractable. Indeed, modelling the random execution times with any distribution other than the Exponential distribution coupled with the fact that we have multiple resources would lead to a very difficult infinite state space problem. However, we note that suboptimal techniques [4, 13] can be used on problems of this type and they are the focus of future work. The memoryless property of the Exponential distribution saves us a lot of work, but in order to employ the standard dynamic programming algorithm of value iteration to a continuous-time problem, we need to convert it into a discrete-time problem using the technique of uniformization.

The idea behind uniformization is to choose a new uniform transition rate which is the same for all states and controls with the caveat that sometimes a transition will leave the state unchanged. Statistically, the uniform process is identical to the original process in that the amount of time spent in each of the states is the same. We select a new uniform transition rate ν such that $\nu \geq \nu_{x,m}(u)$ for every state (x, m) and control u. For our resource leasing problem we choose the rate $\nu = \lambda + R\mu$, where R

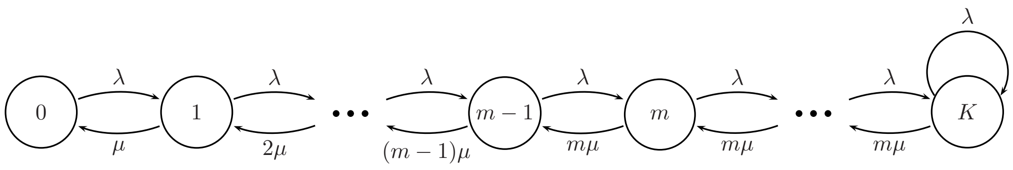

Figure 2: One-step Transition Rates for Original Problem

is the maximum number of resources that may be leased. The transition rates for the uniform version of the problem, given in terms of the original parameters, are presented in Figure 3. In order to specify the transition probabilities corresponding to the uniform rate, we observe that a transition leads to a new state with probability $\nu_{x,m}(u)/\nu$, and remains in the same state with probability $1 - \nu_{x,m}(u)/\nu$. Thus, the one-step transition probabilities for the uniform version of the problem are given by

$$\tilde{p}_{ij}(u) = \begin{cases} \frac{\nu_{i,m}(u)}{\nu} p_{ij}(u) & \text{if } i \neq j, \\ \frac{\nu_{i,m}(u)}{\nu} p_{ij}(u) + 1 - \frac{\nu_{i,m}(u)}{\nu} & \text{if } i = j. \end{cases} \tag{3.2}$$

3.2.3 n-step Transition Probabilities

Given that state transitions occur whenever service requests enter or depart the system and the assumption that inter-arrival times and execution time are exponentially distributed, we could choose to impose the leasing decision at any point in time, and not necessarily in coordination with arrivals and departures. The question then arises: when should leasing decisions be made, and how often should they be made? Naturally, we want to make leasing decisions in such a way that demand is accommodated while deployments don't happen too often (due to cost concerns). In our model the deployment/tear-down decisions are made at the times of service request arrivals and departures, but not after every single arrival or departure event. We let the number of events that pass in between leasing decisions be specified by a parameter n. Since arrivals and departures are independent events, computing the n-step transition probabilities is accomplished by multiplication of the one-step probabilities. If we denote the matrix of one-step transition probabilities by $P^1(u)$, the entries of which are specified by Equation 3.1, then the n-step transition probabilities are computed by raising $P^1(u)$ to the power n:

$$P^n(u) = \{P^1(u)\}^n. \tag{3.3}$$

3.3 Cost per Period

Our objective is to minimize cost, but more specifically we want to find the optimal tradeoff between the cost of leasing resources and the cost of exorbitant waiting time. Naturally, the more resources that are leased for the network service, the shorter the waiting time. Leasing fewer resources saves money but effects longer waiting times. In our model the cost of leasing has two components:

1. a deployment cost s, and
2. a resource holding cost h.

The deployment cost s is a one-time cost that is charged whenever the service is deployed onto a resource for the first time, or whenever it is removed from a resource. Intuitively it is undesirable for a service provider to rapidly deploy and then take-down a network service in quick succession. This would serve no purpose for users of the service and would cause unnecessary overhead. The deployment cost prevents this type of deploy/take-down thrashing effect. The second component of the cost of leasing is the resource holding cost, h, which is the cost per unit time for keeping a lease active. Once

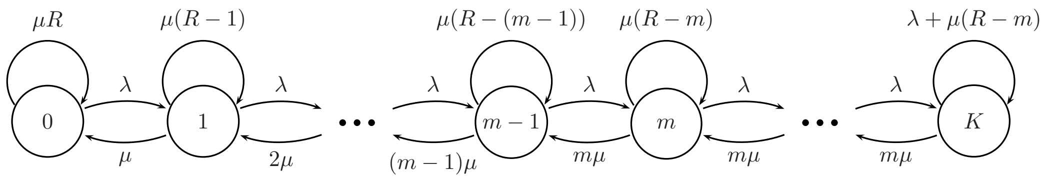

Figure 3: One-step Transition Rates for Uniform Version

the service is deployed, the service provider may hold the lease for as long as desired.

If leasing costs were the only concern, then the service provider would lease as few resources as possible; however, we assume that quality of service is important. Therefore, we impose a cost c per unit of average waiting time W. For the dynamic programming algorithm we use an approximation to average waiting time that is based on Little's Law [12]. First we compute the one-step and the n-step transition matrices, $P^1(u)$ and $P^n(u)$, as described in Sections 3.2.1 and 3.2.3. Given the number of requests currently in the system, x, we can then compute the expected number of requests in the system after n arrivals or departures have taken place, i.e. after n steps. Denote this number by L. Then

$$L = E\{x(t_n) \mid x(t_0) = i\} = \sum_{j=0}^{K} p_{ij}^n(u) \times j.$$

To get the expected average waiting time W, we invoke Little's Law, which states

$$W = \frac{L}{\lambda}.$$

We note that using this method is only an approximation to the average waiting time and becomes more accurate for large values of n. Combining the three cost components we specify the total cost per period as

$$g(x, m, u) = s|u| + h(m + u) + cW. \qquad (3.4)$$

4 Solution Method

In the previous section we described the state space, transition probabilities, and cost per period of a continuous-time, infinite-horizon formulation of the resource leasing problem. Using the technique of uniformization we converted the problem into an equivalent discrete-time formulation which may be solved by employing the dynamic programming method of value iteration. We refer the interested reader to the monograph by Bertsekas [3] for detailed information on the theory of dynamic programming and its solution methods. Below we provide an outline of the solution method that we employed.

The solution is the optimal cost function and the associated decisions that achieve the minimum costs. Given an initial state the dynamic programming cost function, denoted by J, is the minimum over all admissible decisions of the cost of the current period plus the discounted cost of all future periods starting from the subsequent state. The theory of dynamic programming states that the optimal cost function satisfies the following set of equations, which are known as Bellman's equations.

$$\begin{aligned} J(i,m) = \min_u \Bigg[& g(i,m,u) \\ & + \alpha \sum_{j=0}^{K} \tilde{p}_{ij}(u) J(j, m+u) \Bigg], \\ & i = 0, \ldots, K, \quad m = 0, \ldots, R, \qquad (4.1) \end{aligned}$$

where the cost per period g is given by Equation 3.4, the uniform transition probabilities $\tilde{p}_{ij}(u)$ are given by Equation 3.2, and α is a discount factor applied to future costs ($0 < \alpha < 1$). Aside from having nice theoretical properties regarding convergence, discounting future costs are appropriate in our case for two reasons: 1) the future is more uncertain than the present; therefore, we should place less emphasis on costs that are further into the future since those estimates are less accurate, and 2) the time value of money — money is simply worth more today than it will be in the future.

We mention here that since we are using the technique of uniformization, a portion of the cost per period g is also discounted by an amount $1/(\beta + \nu)$, where $0 < \beta < 1$. In the interest of space we state without justification the fact that $\alpha = \nu/(\beta+\nu)$ (see [3].) The portion of g that is discounted is the holding cost per unit time and the waiting cost per unit time. The deployment cost is a one-time cost that does not accumulate over time and therefore is not discounted. Thus, Bellman's equation for our problem becomes

$$J(i,m) = \min_u \left[s|u| + \frac{h(m+u) + cW}{\beta + \nu} + \alpha \sum_j \tilde{p}_{ij}(u) J(j, m+u) \right]. \quad (4.2)$$

Rewriting Equation 4.2 in terms of the transition probabilities of the original problem and eliminating α gives

$$J(i,m) = \frac{1}{\beta + \nu} \min_u \Bigg[(\beta+\nu)s|u| + h(m+u) + cW + (\nu - \nu_i(u))J(i, m+u) + \nu_i(u) \sum_j p^n_{ij}(u) J(j, m+u) \Bigg], \quad (4.3)$$

which is the form that we use in the implementation of the value iteration method. Note that we use the n-step transition probabilities as computed in Equation 3.3.

The method of value iteration is the application of Bellman's equation. Beginning with an initial cost function $J(i, m)$ for every state (the zero function), the costs are updated at each iteration according to Equation 4.3. The algorithm terminates when the difference in the cost functions between successive iterations becomes small enough, i.e. the algorithm converges. Convergence is guaranteed under the assumptions of a finite state space and a bounded cost function, both of which apply to our problem. The method of value iteration for the resource leasing problem is presented as Algorithm 1.

5 Experimental Results

5.1 Example Problem

In this section we characterize the optimal cost function and the leasing policy obtained from the solution of an example problem. The parameters for the problem, which are the input to Algorithm 1, are presented in Table 1. We implemented Algorithm 1 in Matlab. For our example problem the value iteration method converged after 255 iterations[1]. The resulting optimal cost function is presented in Figure 4 and the associated leasing policy is shown in Figure 5. The shape of the optimal cost function reveals some intuitive but interesting properties: 1) the cost is nondecreasing in the number of requests in the system, and 2) the cost is convex-like in the number of leased resources, i.e. the cost increases when the service is under-deployed or over-deployed. How to choose the correct (minimum cost) level of deployment is determined from the optimal leasing policy shown in Figure 5. Recall that the leasing policy is state-dependent. After n events have occurred, the decision-maker, i.e. the scheduling software, observes the current number of requests in the system and the current number of leased resources and consults the policy to obtain the leasing decision u. Referring to Figure 5, more resources are leased as the system becomes busier or when the service is underdeployed. The policy is to terminate leases when the service is over-deployed. However, there is a large "flat area" in the policy where the decision is to leave the service deployed at its current level. This is partly a result of in-

[1] To speed convergence we implemented the Gauss-Seidel method for updating costs in Algorithm 1. This technique is not shown in order to improve the clarity of the presentation. We refer interested readers to [3].

Algorithm 1: Value Iteration Method for the Resource Leasing Problem

```
Data: $\lambda, \mu, R, K, \beta, n, c, h, s, \epsilon$
Initialize: $k \leftarrow 0,\ J_k \leftarrow \mathbf{0},\ \nu \leftarrow \lambda + R\mu$
 1: repeat
 2:     for $i \leftarrow 0$ to $K$ do                                ▷ For every possible state
 3:         for $m \leftarrow 0$ to $R$ do
 4:             for $u \leftarrow -m$ to $R - m$ do            ▷ For every admissible leasing decision
 5:                 Compute $P^1$
 6:                 Compute $P^n$
 7:                 $L \leftarrow \sum_{j=0}^{K} p_{ij}^n \times j$                 ▷ Expected number in system
 8:                 $W \leftarrow L/\lambda$                                ▷ Expected waiting time
 9:                 $g \leftarrow (\beta + \nu)s|u| + h(m + u) + cW$
10:                 if $i < m + u$ then
11:                     $C \leftarrow g + (\nu - \lambda - i\mu)J_k(i, m + u) + (\lambda + i\mu)\sum_j p_{ij}^n J_k(j, m + u)$
12:                 else
13:                     $C \leftarrow g+(\nu-\lambda-(m+u)\mu)J_k(i,m+u)+(\lambda+(m+u)\mu)\sum_j p_{ij}^n J_k(j,m+u)$
14:                 end if
15:                 Store the minimum cost $C_{\min}$ found so far and the associated decision $u$
16:             end for
17:             $J_{k+1}(i, m) \leftarrow (1/(\beta + \nu)) \times C_{\min}$          ▷ Update entry for state $(i, m)$
18:         end for
19:     end for
20:     $k \leftarrow k + 1$
21: until $\|J_{k+1} - J_k\| < \epsilon$
```

cluding the deployment cost s in the cost function. Ideally the system operates in this range and avoids deployment/tear-down cycles.

Table 1: Parameters for Example Problem

Arrival rate λ	4
Processing rate μ	1
Max number of resources R	12
Max system capacity K	50
Discount factor β	.9
Number of events in a period n	20
Wait cost per unit time c	10
Lease cost per unit time h	5
Deployment cost s	5
Convergence criterion ϵ	10^{-3}

Taking a closer look at the optimal cost function, the cost versus number of resources is presented in Figure 6 for five different values of i (number of requests). The figure shows the convex-like appearance of the cost. A minimum cost is achieved when the service is deployed on between five and seven resources. An interesting characteristic is that the optimal cost is more sensitive to under-deployment than to over-deployment, as evidenced by the spread of cost function at the left end of the graph and the closeness at the right end of the graph. This indicates that when demand for the service is known to exist but is highly irregular, it is better to be slightly over-deployed than under-deployed. Knowledge of the sensitivity is also beneficial in the case where consistency and predictability of the leasing cost is of concern, e.g. service providers who budget for leasing costs.

Regarding cost versus number of requests in the system, Figure 7 shows the optimal cost for five different levels of m (number of resources). We see that the cost is nondecreasing in the number of requests. The lowest costs are natu-

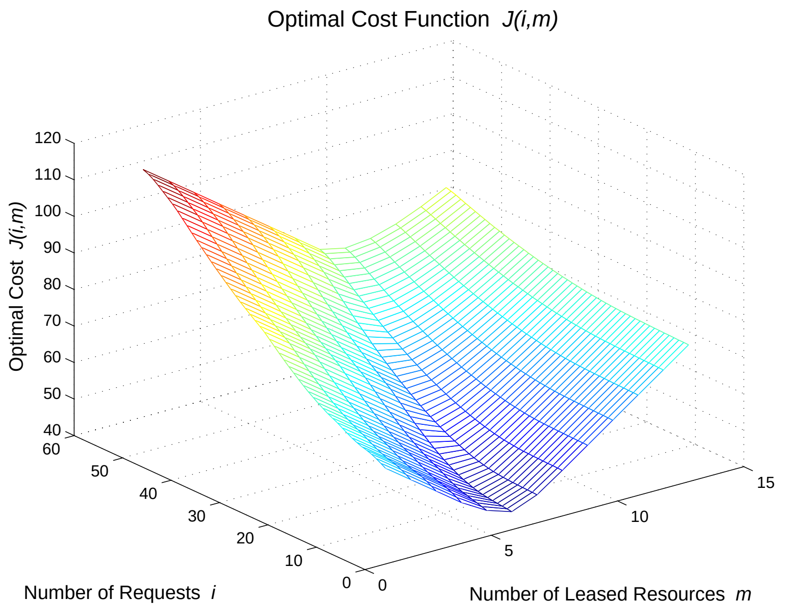

Figure 4: Optimal Cost Function

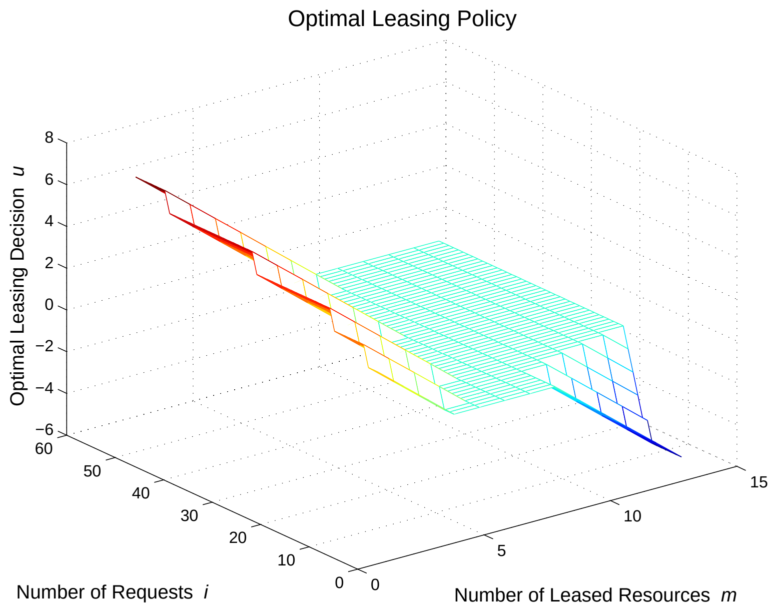

Figure 5: Optimal Leasing Policy

rally achieved when there are very few requests in the system since there is no waiting cost. Figure 7 shows that the optimal cost function is not very sensitive to the number of requests in the system. This can perhaps be taken as good news since, unlike the number of leased resources, the number of requests is random and cannot be directly controlled. We note, however, that for higher levels of deployment, $m = 8$ and $m = 11$, the optimal cost does not vary as much from one end of the graph to the other, which corresponds with our earlier discussion of Figure 6.

5.2 Sensitivity of the Model

The values of the parameters in the example problem were chosen not for their absolute values, but rather for their relative values. For example, we assume that the arrival rate of requests, $\lambda = 4$, is considerably greater than a single resource can handle, $\mu = 1$, so that the policy will result in the lease of multiple resources. The choice of n, the number of events in a period, indirectly controls the length of a decision period. Note the difference between the two alternatives to determine the length of a decision period: 1) a fixed-length decision period that lasts for t time units, and 2) letting the length of the decision period be determined by some number of events. Option 2 is more responsive to abrupt increases in demand for the service. A flash crowd of arrivals would cause the number of events to reach n very quickly and thus a new leasing decision would be made to increase the number of resources. On the other hand, if demand trailed off significantly, then it could take a long time for the system to recognize that it should terminate the leases on some resources, whereas leasing decisions that were made on fixed-length time intervals would recognize this situation sooner. Note that since both the inter-arrival times and the execution times are exponentially distributed, and therefor possess the memoryless property, it is possible to build a hybrid model wherein the length of a decision period is determined by the minimum of a number of events n and a fixed length of time t.

The cost parameters were also chosen for their relative values. In the example problem we assume that the cost of waiting time, $c = 10$, is relatively more important than the cost of holding a lease, $h = 5$. This corresponds to most situations, otherwise there is not much incentive to host the service at all. We ran additional experiments with increasing values of c and h. Holding h at 5, we increased the value of c to 15, 20, and 25. Similarly, holding c at 10, we increased the value of h to 10, 15, and 20. Although the overall cost of hosting the service naturally increased, and the values of the state (i, m) that trigger leasing decisions are different, the general shape and trend of the optimal cost function and the leasing policy did not change. For this reason we omit those figures.

The parameters that have the largest impact on the model are those that affect the size of the state space: the size of the resource pool, R, and the system capacity, K. It is well-known that explosion in the size of the state space is a drawback of using dynamic programming methodology [2, 4]. In Algorithm 1, we compute the cost of every every possible leasing decision for every possible state. This leads to $(K+1) \times (R+1)^2$ cost computations for each iteration of the algorithm. Figure 8 shows the increase in the size of the state space for increasing values of R (with $K = 50$). For comparison purposes, our example problem with $K = 50$ and $R = 12$ results in a state space of size 8,619. Recall that the algorithm terminated after 255 iterations. A larger problem with $K = 100$ and $R = 24$, which has a state space of size 63,125, terminated after 321 iterations. So not only are there more computations per iteration, but more iterations are required for convergence because the matrix of optimal costs, J, is larger. It is important to note, however, that the algorithm is not an "online" algorithm. All of the computations are performed before the system begins operation. At the beginning of each decision period, only the (pre-computed) policy is consulted, which is just a simple look-up. Whenever there is a

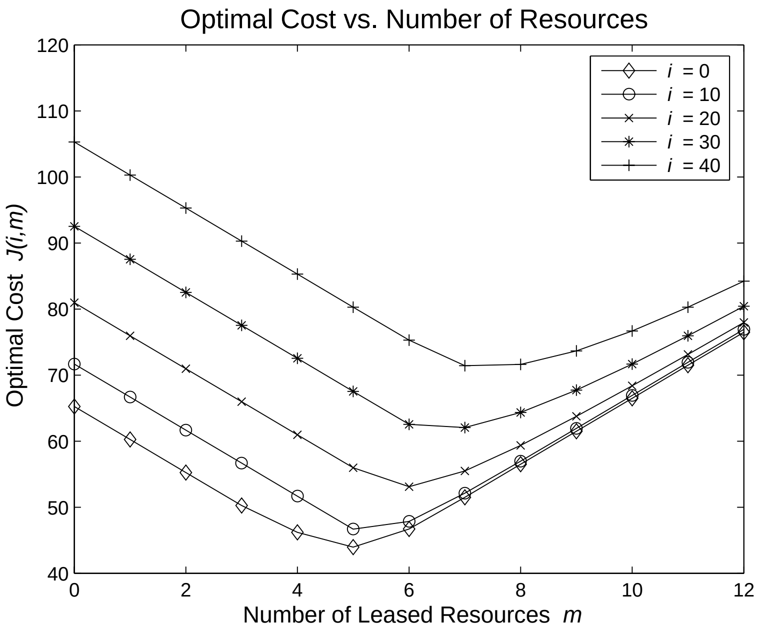

Figure 6: Cost vs. Number of Leased Resources

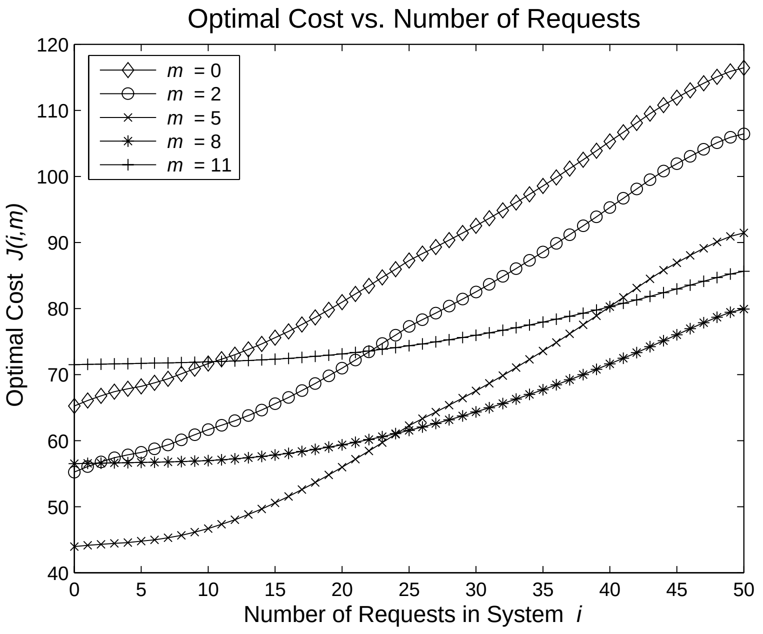

Figure 7: Cost vs. Number in System

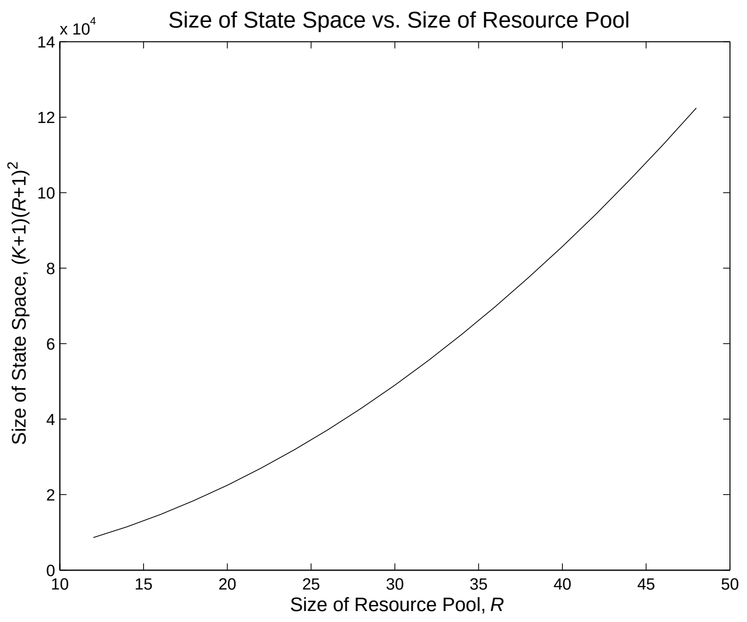

Figure 8: Size of State Space

permanent change in one of the underlying parameters, e.g. arrival rate or cost parameter, then the algorithm is executed using the new parameters and a new leasing policy is generated.

6 Related Work

In [7] the authors formulate and solve a finite-horizon version of the resource leasing problem. Our previous work is appropriate for the case where the service is to be deployed for a known, relatively short period of time. There are, however, some important limitations that we address in this article. First, this work is targeted toward services that are to be maintained for a long period of time, hence the infinite-horizon formulation. The solution methods for these types of problems are different from the finite-horizon case. In addition, the work presented in [7] is for a discrete-time problem wherein the lengths of the decision periods are constant (and somewhat arbitrarily determined). In contrast, in this work we solve a continuous-time problem (by conversion to a discrete-time representation) in which the lengths of the decision periods are determined by the frequency of events, i.e. arrivals and departures. A final important difference is that our previous work did not consider queueing. Every request required a resource upon its arrival, triggering a new lease whenever no resources were available. More realistically, in this work requests are queued to the point where the cost of leasing outweighs the cost of waiting time. The objective is to find the optimal balance between these two conflicting costs.

A number of works have proposed service-oriented architectures and have tested high-performance applications in those environments [8, 18–20]. In [20], Weissman and Lee present an architecture and middle-ware for dynamic replica selection and creation in response to service demand. Their work answers the questions of *when* and *where* to deploy a service. In contrast, this work focuses on the question of *how many* resources are required to host a network service in the presence of random demand and execution times.

Buyya et. al. [6] and Wolski et. al. [22, 23]

examine the use of supply and demand-based economic models for the purpose of pricing and allocating resources to the consumers of grid services. In this work we assume a supply and demand-based economy in which both software services and computational resources are in demand. In particular, we assume a separation of interests between the service provider and the resource provider. The service provider obtains the necessary computational resources at a cost. The user then, is only concerned with the software services that are required for the application, rather than negotiating directly with a resource owner for computing time.

7 Conclusion

In this work we present a formulation and a solution method for the resource leasing problem for on-demand computing. We view the problem from the perspective of the service provider: how many resources should be leased in the presence of random demand for the service and random execution times of service requests. The objective is to minimize the cost of leasing while maintaining quality of service. Due to its stochastic nature we model the problem in a dynamic programming framework. Specifically, we formulate an infinite-horizon, continuous-time Markov decision problem. Using the technique of uniformization we convert the problem into an equivalent discrete-time representation, which may be solved by the dynamic programming method of value iteration. The output of the algorithm is an optimal cost function and its associated leasing policy. We present these results for one example problem and we find that the shape of the optimal cost function is convex-like in the number of leased resources and is nondecreasing in the number of service requests in the system. Furthermore, the optimal cost is sensitive to the number of leased resources. Examination of the contour of the cost function reveals that it is better for a service provider to slightly over-deploy than to under-deploy in the case where demand for the service is known to exist but has large fluctuations. This knowledge is important whenever consistency and predictability of the leasing cost are of particular concern to the service provider.

A model such as the one presented in this article provides a useful abstraction for the resource leasing problem. In addition there are some important direct uses for the model. A service provider may want to predict the cost of leasing for budget purposes. One may also employ the model to choose among alternative leasing arrangements. For example, given the cost structures of different resource providers, the model may be used to select the lowest-cost provider. Even in the case where leasing is not an option and the service provider owns the computational resources, the cost of deployment (disk space, memory) and the cost of waiting time (client satisfaction) may still pose a trade-off that requires analysis. In this case the model can aid in determining the resource requirements for the network service.

Acknowledgements

The authors would like to acknowledge the support of the National Science Foundation under grants CNS-0305641 and ITR-0325949, the Department of Energy's Office of Science under grant DE-FG02-03ER25554, and the Minnesota Supercomputing Institute for Digital Simulation and Advanced Computation and the Digital Technology Center at the University of Minnesota.

References

[1] D. Arnold et al. Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[2] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, second edition, 2000.

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, second edition, 2001.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[5] Gunter Bolch et al. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley and Sons, Inc., 1998.

[6] Rajkumar Buyya et al. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[7] Darin England and Jon Weissman. A stochastic control model for deployment of dynamic grid services. In *Fifth IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, pages 192–199, 2004. November 8, Pittsburgh, Pennsylvania.

[8] Ian Foster et al. Grid services for distributed system integration. *Computer*, 35(6), 2002.

[9] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*, volume 2. Elsevier, 2004.

[10] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[11] Armando Fox et al. Cluster-based scalable network services. In *16th ACM Symposium on Operating Systems Principles, SOSP'97*, 1997.

[12] Donald M. Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. John Wiley and Sons, second edition, 1985.

[13] Peter Marbach, Oliver Mihatsch, and John N. Tsitsiklis. Call admission control and routing in integrated services networks using neuro-dynamic programming. *IEEE Journal on Selected Areas in Communications*, 18(2):197–208, February 2000.

[14] Sidney I. Resnick. *Adventures in Stochastic Processes*. Birkhäuser, 1992.

[15] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, Inc., fourth edition, 1989.

[16] Yoshio Tanaka et al. Ninf-g: A reference implementation of rpc-based programming middleware for grid computing. *Journal of Grid Computing*, 1(1):41–51, June 2003.

[17] Robert von Behren et al. Ninja: A framework for network services. In *Proceedings of the 2002 USENIX Annual Technical Conference*, 2002. Monterey, CA.

[18] Jon B. Weissman, Seonho H. Kim, and Darin England. Supporting the dynamic grid service lifecycle. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid2005)*, May 2005. Cardiff, UK.

[19] Jon B. Weissman and Byoung-Dai Lee. The service grid: Supporting scalable heterogenous services in wide-area networks. In *IEEE Symposium on Applications and the Internet*, 2001. San Diego, CA.

[20] Jon B. Weissman and Byoung-Dai Lee. The virtual service grid: An architecture for delivering high-end network services. *Concurrency: Practice and Experience*, 14(4):287–319, April 2002.

[21] Matt Welsh, David Culler, and Eric Brewer. Seda: An architecture for well-conditioned, scalable internet services. In *18th ACM Symposium on Operating Systems Principles, SOSP'01*, 2001.

[22] Rich Wolski et al. G-commerce: Market formulations controlling resource allocation on the computational grid. In *International Parallel and Distributed Processing Symposium*, 2001.

[23] Rich Wolski et al. Grid resource allocation and control using computational economies. In Francine Berman, Geoffrey Fox, and Anthony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 32, pages 747–769. John Wiley and Sons, 2003.