An On-Line, Business-Oriented Optimization of Performance and Availability for Utility Computing

Joseph Hellerstein
Kaan Katircioglu and
Maheswaran Surendra
{hellers, kaan, suren}@us.ibm.com
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

April 29, 2004

Abstract

Utility computing provides a pay-as-you-go approach to information systems in which Application Providers (e.g., web sites) can better manage their costs by adding capacity in response to increased demands and shedding capacity when it is no longer needed [24]. This paper develops a framework for Application Providers using a cluster design to determine the number of servers that minimizes the sum of quality of service (QoS) costs resulting from service level penalties and server holding costs. The server characteristics considered are service rate, failure rates, repair rates, and costs. We further devise a simple approach for owners of computing utilities to select among servers with different characteristics and for server manufacturers to address design trade-offs between server characteristics. Our approach uses a performance and availability model that extends the $M/M/m/K/M$ queueing system with considerations for failures and repairs. The model is validated with data from a product level testbed with an eCommerce workload. We develop a closed-form approximate solution for the optimal number of servers that works well for a wide range of system parameters. This result indicates that the optimal number of servers m^* is the effective traffic intensity (the number of servers consumed by the workload with considerations for failures and repairs) with an adjustment for QoS costs and server holding costs. Last, we show that the Application Provider cost at m^* is an increasing function of $\hat{v}$, the effective server price performance (which considers failures and repairs). Thus, we propose that utility owners should consider $\hat{v}$ in their purchasing decisions, and manufacturers of servers should assess design trade-offs in terms of their impact on $\hat{v}$.

1 Introduction

Utility computing provides a pay-as-you-go approach to information systems in which Application Providers (e.g., web sites) deliver services to their customers using hardware and software provided by the utility that charges based on the resources consumed. For Application Providers, computing utilities provide a way to better manage costs, especially adding capacity in response to increased demands so as to avoid penalties for violating service level agreements and shedding capacity when it is no longer needed so as to reduce costs. This paper develops an approach for Application Providers to determine the number of servers that minimizes the combined cost of service level penalties and server holding costs based on server characteristics (e.g., service rates, server failure rates, repair rates, and costs). We further show how these results can be

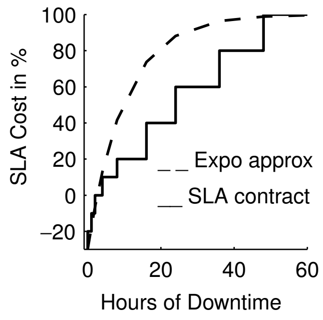


Figure 1: *Staircase function of costs in a service level agreement as reported in [27]. The staircase is approximated by $b(1 - e^{-0.1x}) - C$, where b is the range of the staircase, x is downtime, and C is an offset constant.*

used by utility owners to select servers for the computing utility and by server manufacturers to assess design trade-offs. The performance model that underlies our approach is validated by studies of a product level testbed with an eCommerce workload.

Service level agreements (SLAs) are often used to govern the relationship between Application Providers and their customers (e.g., [14], [27]). SLAs may specify penalties for poor performance, which we refer to as **quality of service (QoS) costs**. The solid line in Figure 1 is an example that is cited in [27] in which there is a penalty for excessive downtime (with a reward, or negative cost, for greatly reducing downtime) that is expressed in terms of the percent of the customer's monthly service charge (e.g., 20 hours of down time results in a rebate to the customer of 40% of their monthly service charge). In general, SLAs are specified in terms of an SLA metric, which might be response time or customer-seconds in system (or other possibilities) [11]. For both availability and response time, the service provider can reduce SLA penalties by using more servers, and/or more efficient/reliable servers and software. In general, doing so results in greater costs for holding servers and software licenses, which we refer to as **server holding costs** (or sometimes just holding costs).

One way to balance QoS costs and holding costs is to acquire and release hardware resources and/or software licenses in response to changes in workloads. Indeed, Hewlett Packard, IBM, Microsoft, and others have recently announced capabilities to do this (e.g., [20], [8]). Figure 2 depicts a computing utility that implements such capabilities. The solid circles represent **servers**, a term we use to refer to the combination of hardware and software resources supplied by the computing utility. The utility may have servers with different characteristics, which are indicated by the different size circles. The **server characteristics** of most interest to us are cost (e.g., dollars charged per second of usage), service rate (speed of processing),

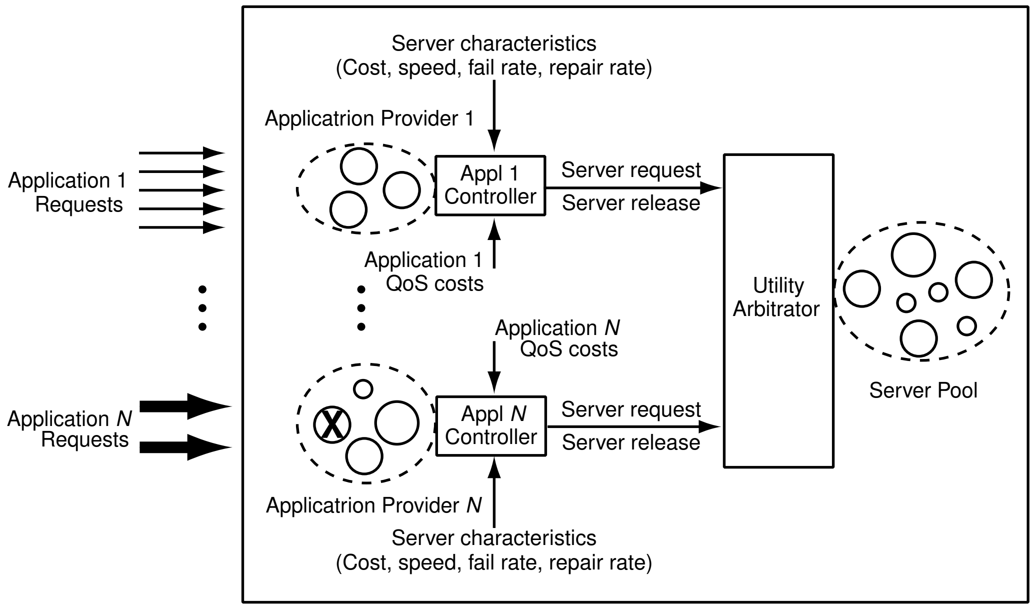


Figure 2: *Architecture of a computing utility. Applications request and release servers (the solid circles) based on server characteristics so as to minimize total cost. The Utility Arbitrator handles the request/release protocol and arbitrates between competing requests. Servers may fail (as indicated by the circle with an "x") and be repaired (e.g., by re-booting) while being assigned to an Application Provider.*

failure rate, and repair rate. We focus on failures that can be detected and corrected automatically, such as disk transient errors, memory leaks, and process failures. Application Providers negotiate SLAs with their customers to establish the price that should be charged for the service quality (e.g., response time) delivered. The Application Controller of the Application Provider uses information about the expected request rates to determine the number and type of servers to request from the Utility Arbitrator (which manages the Server Pool). If a request is granted, then the Application Provider is said to own the server. If an Application Provider no longer needs a server, its ownership is released to the Utility Arbitrator. We consider a cluster-oriented design for the Application Providers (as is common in eCommerce systems) in which customer requests can be handled by any server owned by the Application Provider. Ideally, the Application Controller requests servers in a way that minimizes the sum of QoS and server holding costs, a quantity we refer to as **total cost**. One example of the architecture depicted in Figure 2 is the HotRod multi-system eCommerce testbed that dynamically allocates WebSphere Application Servers in response to flash events that can change application loads by a factor of twenty or more [17].

This paper addresses a new set of capacity management issues that arise as a result of operating computing utilities for application clusters. Traditionally, capacity planning is done every few months using off-line tools. With the advent of computing utilities, Application Providers may want to add or remove servers many times in an hour in order to minimize total cost. Our work develops a simple criteria for making these decisions. We also want to assist utility owners who desire to be cost competitive by buying servers (and server software) that minimize the total cost of Application Providers (e.g., a less expensive middleware package may be a poor choice if it runs slower or fails more often). And we want to aid manufacturers of server hardware and software in making design decisions with trade-offs between cost, service rate, failure rate, and repair rates. An example of such a trade-off is the frequency with which "heart-beats" are done to detect hardware and software failures. Faster heart-beats increase the repair rate, but they also increase overheads and hence reduce service rates.

There is considerable work related to this paper. Computing utilities rely heavily on dynamic allocation of resources, a topic that has been studied extensively (e.g., [21] for a self-tuning resource management system, [30] for video QoS, [4] for multimedia networks, [12] for TCP adaptive queue management, and [19] for storage devices). However, none of these approaches address failures and repairs in combination with QoS costs and server holding costs. Performability provides a way that performance and reliability analysis can be considered in the same framework by having a reward associated with system states [22]. Many researchers have described approaches to performability modeling (e.g., [28] uses Markov models, [6] employs layered queueing models, and [29] leverages fluid stochastic Petri nets). While providing a useful analytic context, these approaches, in themselves, do not address the problem of how Application Providers should acquire and release servers to minimize total cost. In particular, the existing performability literature does not consider SLA costs of the form depicted in Figure 1. More directly related to our problem are efforts that address manufacturing lines (e.g., [7] addresses repairable assembly lines, [2] studies a just-in-time approach to manufacturing, [18] addresses configurations of continuous production systems, and [1] investigates the effects of stochastic market demands). All of these consider some element of optimization and reliability, but the configuration parameters, performance metrics, and cost metrics are quite different from those used in computing systems. There has also been work that seeks to optimize computer configurations based on performance and reliability considerations. [15] describes a conceptual framework for QoS optimization, but many specifics are missing (e.g., cost model, validations for a real system). [25] provides performance and availability optimizations for cellular networks, but this work does not address QoS or holding costs. [3] considers a problem very close to our own in which server capacity is dynamically allocated. However, many factors are not considered, especially server holding costs, failures, and repairs.

This paper makes several contributions to the management of enterprise services based on utility computing:

1. We develop a business-oriented cost model for resource allocation for Application Providers. The cost model is based on SLA costs and server costs.

2. We construct a model for the performance and availability of an eCommerce system and show that the model is consistent with data from a multi-system testbed with an eCommerce workload. Our technical approach is based on the $M/M/m/K/M$ queueing system in which failures and repairs are considered. While others (e.g., [23]) have developed the general notion of queues with breakdowns, we develop and assess a model appropriate for a real world system.

3. We formulate a simple (closed form) approximation for the optimal allocation of servers for an Application Provider based on SLA and server holding costs. An algorithm is developed and assessed that can be implemented easily by an Application Controller.

4. We develop a simple criteria for utility owners and server manufacturers to make design trade-offs. Our approach is based making utility owners making buying decisions and manufacturers making design

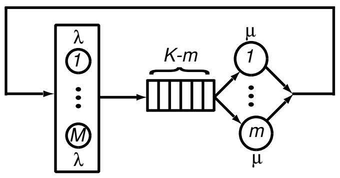


Figure 3: *$M/M/m/K/M$ queueing system.*

decisions that reduce the total cost for Application Providers if they operate at their optimal server allocations.

The remainder of this paper is organized as follows. Section 2 develops a Markov model for QoS metrics that considers failures and repairs; the model is consistent with measurements of the HotRod multi-system eCommerce testbed. Section 3 develops a cost model that incorporates holding costs and QoS costs based on SLAs whose metrics can be obtained from the Markov model. Section 4 develops a closed form expression for an approximation of total cost from which we estimate the optimal number of servers; we also use this model to address the server selection and server design problems. Our conclusions are contained in Section 5.

2 Performance and Availability Model

This section develops a performance and availability model for Application Providers using a cluster design in which any of the servers it owns can respond to an application request. The model is quite general in that it addresses clusters with heterogeneous server characteristics (although we do assume that service times and inter-arrival times are exponentially distributed). The model is consistent with data from the HotRod multi-system eCommerce testbed.

Our starting point is the $M/M/m/K/M$ queueing model, which is depicted in Figure 3. This queueing system has M customers (i.e., requestors of services from the Application Provider). A customer is either "thinking", has a request waiting for service, or that request is being served. The rectangle that encloses the circles labelled $1, \cdots, M$ indicate the time spent thinking, which is exponentially distributed with a mean of $\frac{1}{\lambda}$. The circles on the right labelled $1, \cdots, m$ are the servers that process requests. The waiting area has $K-m$ spaces for requests. Of most interest is the case where $K = M$ so that no request is lost (although there is no technical difficulty with considering $K < M$). We denote this system by $M/M/m/M/M$. This queueing system can be easily described as a Markov chain [16]. Let $p_n(t)$ be the probability of having n requests in the system (either waiting for or receiving service) at time t. Then $\lambda(M-n+1)p_{n-1}(t) = \mu \min\{m, n\} p_n(t)$. From this relationship, it is easy to construct the transition rate matrix (which has a tridiagonal form). Let

$\mathbf{p}(t) = (p_0(t), \cdots, p_M(t))$ be the probability state vector, and let A be the $(M+1) \times (M+1)$ transition rate matrix. From [9], we know that $\mathbf{p}(t) = \mathbf{p}(0)e^{\mathbf{A}t}$. In this way, we can obtain the transient solution to the system. However, if t is large enough, then $\mathbf{p}(t) = \mathbf{p}_{ss} = (p_0, \cdots, p_M)$, the steady state probability. From [16], we know that the steady state solution is

$$\begin{aligned} p_n &= p_0 \left(\frac{\lambda}{\mu}\right)^n \left(\begin{array}{c} M \\ n \end{array}\right), 0 \le n \le m \\ &= p_0 \left(\frac{\lambda}{\mu}\right)^n \left(\begin{array}{c} M \\ n \end{array}\right) \frac{n!}{m!} m^{m-n}, m+1 \le n \le M \end{aligned} \qquad (1)$$

$$p_0 = \left(\sum_{k=0}^{m-1} \left(\frac{\lambda}{\mu}\right)^n \left(\begin{array}{c} M \\ n \end{array}\right) + \sum_{n=m}^{\infty} \left(\frac{\lambda}{\mu}\right)^n \left(\begin{array}{c} M \\ n \end{array}\right) \frac{n!}{m!} m^{m-n} \right)^{-1}$$

$M/M/m/M/M$ turns out to provide a very good description of the performance of the HotRod system [17], a product level eCommerce testbed that uses multiple HTTP servers, application servers, and a database server. We studied HotRod's performance using a modified version of the SpecJAPPServer2002 eCommerce workload [5] that generates time varying transaction rates, and collected measurements of response time and throughput in 10 second intervals. We obtained the parameters of the $M/M/m/M/M$ model for $m = 3$ servers as follows: $\frac{1}{\mu}$ is 30 msec, the average time to process a single transaction; $M\lambda$ is set to throughput, which is in units of business operations per second or BOPS; and M is a free parameter that we evaluate at different values. The foregoing allows us to estimate $\mathbf{p}(t)$, but not $R(t)$, the response time at time t. Unfortunately, we must estimate $R(t)$ to make comparisons with measurements of the HotRod system. One approach to estimating $R(t)$ is to use Little's Result [16], which equates number in system to throughput times response time if the system is in steady state. However, this is inappropriate in our case since: (a) we cannot assume that the system is in steady state; and (b) throughput is a function of response time in closed queueing systems. It turns out that our assumption of exponential service times provides a way to circumvent these difficulties. A request arriving at time t finds n requests in the system with probability $p_n(t)$. If $n < m$, then the request's expected response time is $\frac{1}{\mu}$ (since there is a free server). Otherwise, the request must wait for $(n - m + 1)$ requests to leave the system before entering service, where the time for the next request to depart is $\frac{1}{m\mu}$. Thus,

$$\hat{R}(t) = \frac{1}{\mu} + \sum_{n=m}^{M} \frac{n-m+1}{m\mu} p_n(t)$$

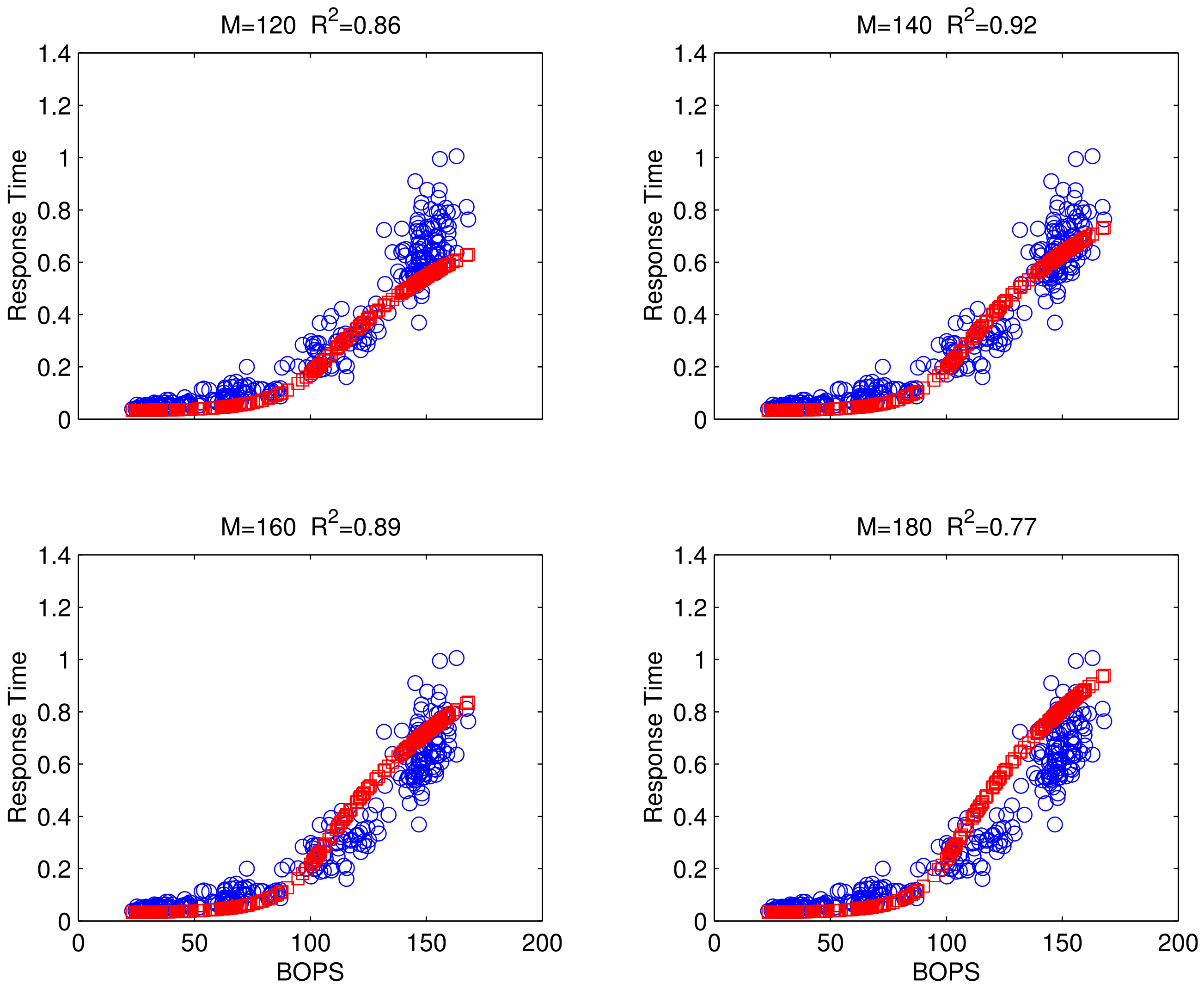


Figure 4: *Assessment of $M/M/m/K/M$ queueing model ($K = M$) for data collected from a three server cluster. μ = 30/sec. BOPS is business transactions per second, which is related to λ.*

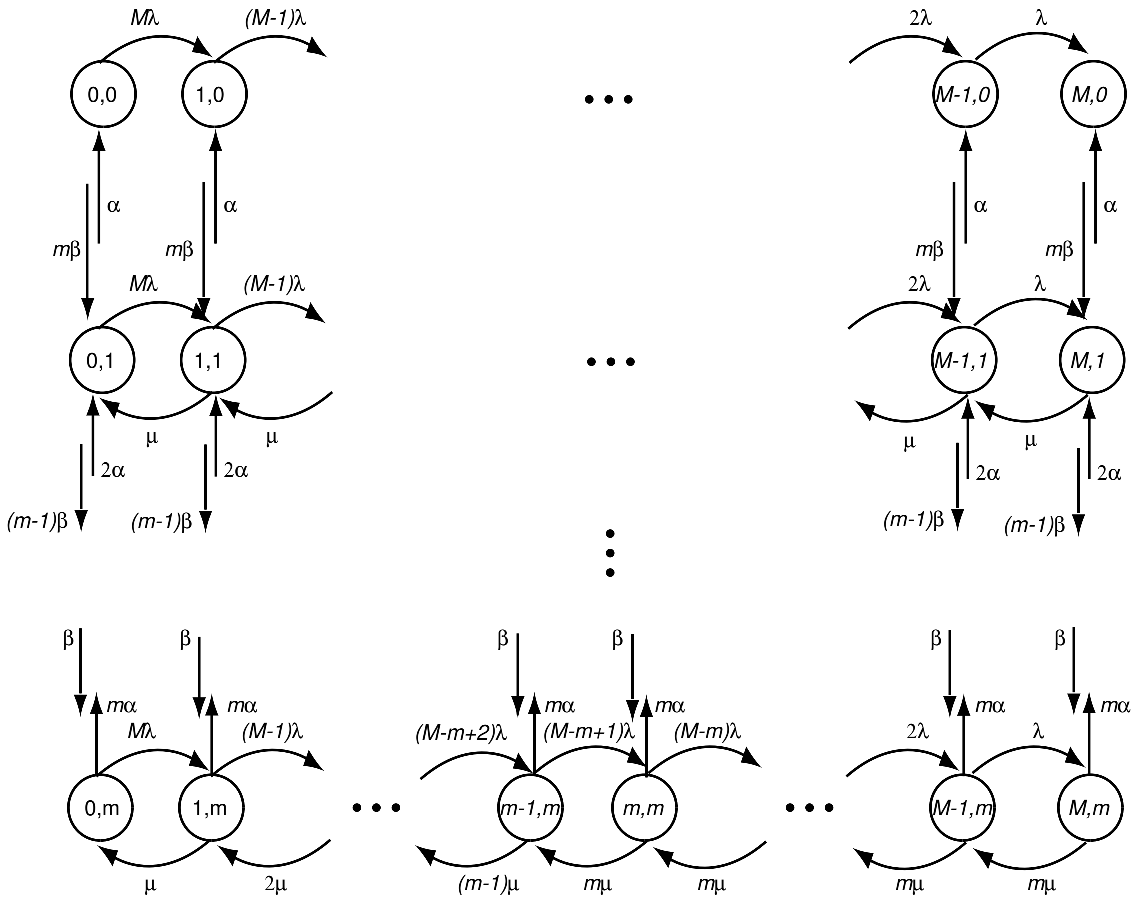


Figure 5: *Markov chain for $M/M/m/M/M$ with failures and repairs. The state is a tuple $n, \tilde{m}$ where n is the number of customers requesting or receiving service and $0 \leq \tilde{m} \leq m$ is the number of non-failed servers.*

Figure 4 plots results for a three server HotRod system. The plots display response time versus BOPS at different values of M. The circles are measurements from the HotRod system, and the squares are the $\hat{R}(t)$ estimated response times using $M/M/m/M/M$ transient probabilities. The R^2 value approximates the variability explained by $\hat{R}(t)$, where $R^2 = 1$ is a perfect fit. We see that $R^2 = 0.92$ at $M = 140$, which is an excellent fit.

Having justified the use of $M/M/m/M/M$, we generalize it in two ways. First, we consider server failures and repairs. When a failure occurs, the request being processed on the failed server is placed at the head of the Application Provider's queue. We simplify matters and assume that the service time of the request after the server failure is drawn from the same exponential distribution as the original service request. This assumption provides a way to gain insight into system characteristics without detailed data that may be costly or impossible to obtain. We assume that the time between failures is exponentially distributed, as is the time to repair. Both assumptions are common in performability analysis (e.g., [23] and [25]) and reliability engineering (e.g. [13]) (although we note in passing that a wide range of distributions may be used to characterize failure data, such as the Weibull, normal, and Raleigh distributions [13]). We use α to denote the rate at which a server fails, and β to denote its repair rate.

We model $M/M/m/M/M$ with failures and repairs using a two dimensional state model $(n, \tilde{m})$, where: (1) n is the number of requests in the system and (2) $\tilde{m}$ is the number of non-failed servers owned by the Application Provider. The Markov chain must include four types of transitions. The first two are the same as for $M/M/m/M/M$ in that we must handle arrivals and departures (service completions). The third type of transition is for failures; that is, the transition $(n, \tilde{m}) \rightarrow (n, \tilde{m}-1)$, which occurs at a rate $\tilde{m}\alpha$. The fourth is for repairs; that is, the transition $(n, \tilde{m}) \rightarrow (n, \tilde{m}+1)$, which occur at a rate $(m-\tilde{m})\beta$. Figure 5 depicts the state transition diagram for this Markov chain. Note that, with one exception, transitions along the horizontal dimension are identical to those for a $M/M/\tilde{m}/M/M$ queueing system. The one exception is when $\tilde{m} = 0$, in which case there are no transitions that reduce the number of requests in the system. Transitions along the vertical dimension are for failures (moving up) and repairs (moving down).

A second generalization of $M/M/m/M/M$ is to consider different types of servers. Suppose there are $\tilde{m}_i$ servers of type i, where $1 \le i \le I$. The i-th server has service rate μ_i, failure rate α_i, and repair rate β_i. To avoid tedious details, we only outline how to construct this Markov chain. We assume that the Application Provider employs a scheduling policy whereby requests are assigned to the fastest server it owns that is idle and not failed (an approach that is consistent with maximizing service rates that has been proposed in [31]). Thus, we can express the state of an Application Provider with a heterogeneous set of servers as $(n, \tilde{m}_1, \cdots, \tilde{m}_I)$, where the types are ordered from fastest to slowest. There are four types of events that can cause a state transition: the arrival of a request, the departure of a request, a server failure, and a server repair. Suppose that the initial state is $(n, \tilde{m}_1, \cdots, \tilde{m}_i, \cdots, \tilde{m}_I)$. Table 1 specifies the new state for each type of transition, the transition rate, and the states for which the transition applies.

The transition rate for the departure event requires some explanation. Let $\bar{m}_i$ be the number of requests served at rate μ_i if we are in state $(n, \tilde{m}_1, \cdots, \tilde{m}_i, \cdots, \tilde{m}_I)$. Then,

$$\bar{m}_i = \min\{\sum_{k=0}^{i} \tilde{m}_k, n\} - \min\{\sum_{k=0}^{i-1} \tilde{m}_k, n\} \tag{2}$$

where $\tilde{m}_0 = 0$. We explain Equation (2) for $I = 2$ servers. If $n \le \tilde{m}_1$, the transition rate is $n\mu_1 = \bar{m}_1\mu$. If $n = \tilde{m}_1 + 1$, then the first $\tilde{m}_1$ requests are served at the rate μ_1, and the last is served at the rate μ_2. So the transition rate is $\tilde{m}_1\mu_1 + \mu_2 = \bar{m}_1\mu_1 + \bar{m}_2\mu_2$. Thus, $\bar{m}_i$ is the number of requests that are served at rate μ_i.

We are unaware of others who have formulated a Markov model for $M/M/m/M/M$ with failures and repairs, either for homogeneous or heterogeneous servers. The most closely related work is [23] who provides a general matrix geometric solution for open queueing systems. [31] specifically analyzes $M/M/m/K/M$ using simulation, but does not formulate a Markov model and so provides little analysis insight.

Table 1: Summary of state transitions for a closed queueing system with heterogeneous service, failure, and repair rates. The initial state is $(n, \tilde{m}_1, \cdots, \tilde{m}_i, \cdots, \tilde{m}_I)$. $\bar{m}_i$ is defined in Equation (2).

Event	**New State**	**Transition Rate**	**Constraints**
Arrival	$(n+1, \tilde{m}_1, \cdots, \tilde{m}_i, \cdots, \tilde{m}_I)$	$(M-n)\lambda$	$n < M$
Departure	$(n-1, \tilde{m}_1, \cdots, \tilde{m}_i, \cdots, \tilde{m}_I)$	$\sum_{i=1}^{I} \bar{m}_i\mu_i$	$n > 0$
Repair	$(n, \tilde{m}_1, \cdots, \tilde{m}_i + 1, \cdots, \tilde{m}_I)$	$(m_i - \tilde{m}_i)\beta_i$	$\tilde{m}_i < m_i$
Failure	$(n, \tilde{m}_1, \cdots, \tilde{m}_i - 1, \cdots, \tilde{m}_I)$	$\tilde{m}_i\alpha_i$	$\tilde{m}_i > 0$

3 Cost Model

This section introduces the cost model we use to quantify the trade-off between QoS costs and serving holding costs.

We begin by considering I types of servers, and assume that costs and measurements are computed over intervals of duration T. Let $c_k(m_1, \cdots, m_I)$ be the total cost of the Application Provider owning $m_1, \cdots, m_I$ servers of the various types during the k^{th} interval. Since the total cost over multiple time periods is obtained by summing the appropriate $c_k(m_1, \cdots, m_I)$, we drop the subscript k. Let $c_Q(m_1, \cdots, m_I)$ be the QoS costs and $c_H(m_1, \cdots, m_I)$ be the server holding costs. Then,

$$c(m_1, \cdots, m_I) = c_Q(m_1, \cdots, m_I) + c_H(m_1, \cdots, m_I) \tag{3}$$

We use SLA costs such as the staircase function in Figure 1 to describe $c_Q(m_1, \cdots, m_I)$. We approximate the staircase with a negative exponential, such as the dashed line in Figure 1. Let $x(m_1, \cdots, m_I)$ be a QoS metric such as response time or customer-seconds in system for which a larger value indicates a lower service quality. Thus, we expect that $x(m_1, \cdots, m_I)$ is non-increasing in each m_i. We use

$$c_Q(m_1, \cdots, m_I) = b(1 - e^{-ax(m_1, \cdots, m_I)}) \tag{4}$$

The parameter b is a normalization constant that determines the maximum value of $c_Q(m_1, \cdots, m_I)$. We refer to a as the **QoS cost rate** in that it determines the rate of cost increase. In essence, this parameter relates to the steepness of the SLA staircase function. If $x(m_1, \cdots, m_I)$ is a QoS metric such as throughput for which a larger value indicates better quality of service (and hence we expect $x(m_1, \cdots, m_I)$ is non-decreasing in each m_i), then we use $c_Q(m_1, \cdots, m_I) = be^{-ax(m_1, \cdots, m_I)}$.

We assume that server holding costs are proportional to the time that the server is used. In practice, there may also be a fixed cost for allocating a server. However, we assume that fixed costs are incorporated into the per time unit costs. Let h_i be the cost of holding a server of type i for a second, then the holding cost for the Application Provider during an interval in which it owns $m_1, \cdots, m_I$ servers is

$$c_H(m_1, \cdots, m_I) = \sum_{i=1}^{I} h_i T m_i \tag{5}$$

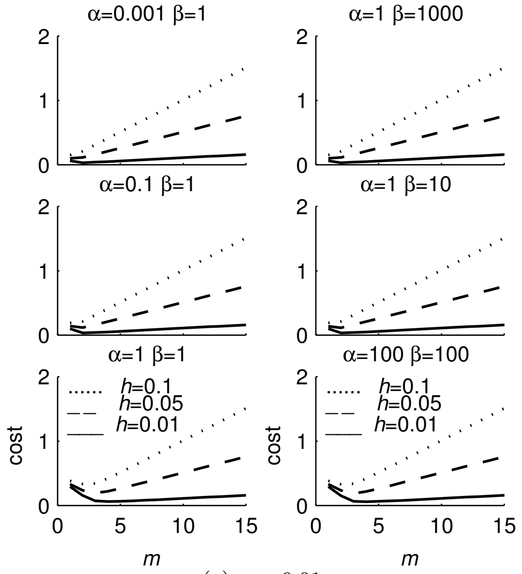
We combine Equation (4) and Equation (5) and then divide by b to normalize the h_i. This results in server holding costs that are in units of the maximum QoS cost. Thus, the total cost for I server types is

$$c(m_1, \cdots, m_I) = 1 - e^{-ax(m_1, \cdots, m_I)} + \sum_{i=1}^{I} h_i T m_i \tag{6}$$

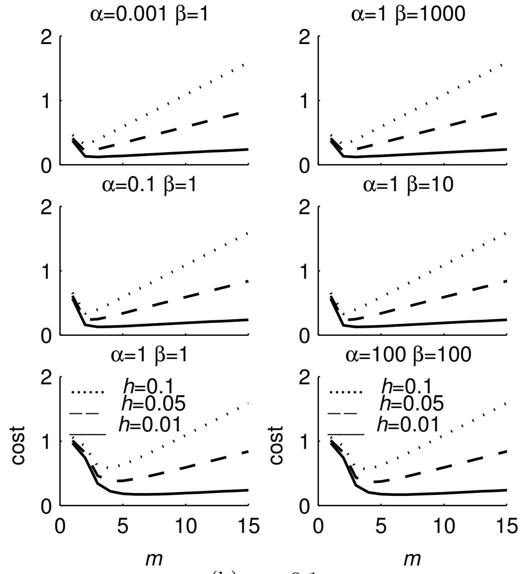
(if $x(m_1, \cdots, m_I)$ is non-increasing in each m_i). Note that the h_i in Equation (6) are normalized by dividing by b.

One appeal of Equation (6) is that it is fairly easy to obtain values for the cost parameters a and h_i. a expresses how rapidly QoS costs increase with the QoS metric. For example, the exponential fit in Figure 1 has $a = 0.1$. A more gradual rise might be $a = 0.01$. To obtain a range for the h_i, we use the QoS metric "customer seconds in the system". Intuitively, this is the sum of the response times observed during the interval T (if interval boundary effects are ignored), which can be estimated as the number in system times T. This metric can be easily related to people costs. By so doing, we can obtain a range for the (normalized) h_i by considering the ratio of server prices to people costs. For example, the cost of a skilled professional employee (including benefits) may exceed \$200,000 per year, and the price of a high performance workstation might be \$20,000. If the server is replaced annually, $h = 20,000/200,000 = 0.1$. Alternatively, suppose that a less expensive server is used, and it is replaced less frequently. This results in a smaller ratio of server costs to people costs, and so h_i may be 0.05 or even 0.01.

Before plotting total cost, we must also consider the values of α and β, the server failure and repair rates. These parameters may vary widely. For example, disk drives may have a mean time between failures of 1,000,000 hours, with repairs on the order of a tens hours (e.g., replace the disk drive). However, the situation for software components is much different. Some authors report that the repair rate for software failures is only twenty times that of the component failure rate [26]. In other cases, the scale of the application creates an even more extreme relationship between failure and repair rates, such as the Google file system in which “failures are common” [10]. Our conclusion is that we should consider a wide range of parameter values. We use $\alpha/\beta \in \{0.001, 0.1, 1\}$.



(a) $a = 0.01$



(b) $a = 0.1$

Figure 6: Cost curves for $M = 50$, $\lambda = 0.02$, $\mu = 1$, $T = 1$.

The sequel considers a single server type (i.e., $I = 1$) and uses number in system as the QoS metric $x(m)$. The motivation for using number in system is that it can be interpreted as people seconds, and the latter can be directly related to costs. We obtain $x(m)$ by observing it every T seconds. Recall that number in system is the first component of a state in the Figure 5 model. $c(m)$ is computed using Equation (6).

Figure 6 plots cost $c(m)$ on the y-axis and m on the x-axis while varying h, a, α, and β using the parameter ranges discussed above. M and λ are fixed, although they are varied in the next section. To reduce visual clutter, line plots are used, even though we only consider integer values of m.

We explore of the effects of the parameters of $c(m)$. First consider the effect of m. Observe that in almost all cases, $c(m)$ initially decreases as m increases until a minimum is reached. To understand why, note that the metric number in system increases exponentially with utilization. Since, adding a server dramatically reduces utilization, it is often the case that number in system drops precipitously when one server is added, which causes $c_Q(m)$ to decline. If h is not too big, then $c(m)$ declines. However, there is a decreasing marginal return from adding more servers. So at some m^*, adding a server increases $c_H(m^*)$ more than it decreases $c_Q(m^*)$. Thus, m^* is a local minimum. Note that $c(m)$ increases linearly from m^*. This is because $c_Q(m)$ is small for $m \geq m^*$, and so $c(m) \approx c_H(m) = hTm$.

Now consider the effect of the cost parameters h and a. Clearly, $c_Q(m)$ is unaffected by h, and $c_H(m)$ is increasing in h. Thus, $c(m)$ is increasing in h, which is consistent with the plots. The parameter a affects the rate with which QoS costs decline, as can be seen by comparing Figure 6(a) in which $a = 0.01$ with Figure 6(b) in which $a = 0.1$. We see that a larger value of a causes the $c(1)$ to be larger and results in a steeper slope for the first part of the curve. This effect is readily explained in terms of $c_Q(1)$ in that a smaller a makes $c_Q(1)$ closer to 0.

Next, we focus on the failure rate α and repair rate β. Figure 6(a) is organized so that the first and second columns change α and β but preserve their ratio. Note that the two columns are identical. The same is true in Figure 6(b). This effect can be explained in terms of Figure 5. Let $p_{n,\tilde{m}}$ be the probability of having n customer requests in the system and $\tilde{m}$ servers available (not failed) to the Application Provider. The columns in Figure 5 represent transitions for failures (moving bottom to top) and repairs (moving top to bottom). If we look at a column in isolation, then $\tilde{m}\alpha p_{n,\tilde{m}} = (m - \tilde{m} + 1)\beta p_{n,\tilde{m}-1}$, or $p_{n,\tilde{m}} = \frac{(m-\tilde{m}+1)}{\tilde{m}} \frac{\beta}{\alpha} p_{n,\tilde{m}-1}$. Thus, just the ratio of β to α matters.

Thus far, we have examined the effects of individual parameters. There are combined effects as well. Consider the combined effect of α/β and a. In Figure 6(a), a is small, and we see very little difference between the first and second row of plots where α/β changes by a factor of 100. In Figure 6(b), a is ten times larger than in Figure 6(a). Here, we see a noticeable change in $c(m)$ between the first and second row of plots.

The effects of λ, M can be understood by inspecting $c_Q(m)$. For number in system (and response time), $x(m)$ increases with λ, M, and $c_Q(m)$ increases with $x(m)$. Thus, $c(m)$ increases with λ, M.

One final observation is that the cost curves do not always have a "U" shape. For example, if h is large enough, $c_H(1)$ dominates $c(1)$. Thus, $c(m) \approx c_H(m) = hTm$, and the global minimum occurs when $m = 1$.

4 Optimizing the Number of Servers

This section develops a closed form approximation of the optimal number of servers using $c(m)$ in Equation (6) where the QoS metric is customer seconds and the system is described by Figure 5. Our primary motivation is to provide a simple decision criteria for the Application Controller to select the number of servers. In addition, we want to give utility owners a basis for selecting the types of servers to purchase for the computing utility, and help server manufacturers with design trade-offs between server characteristics.

Our approach is based on four **assumptions**:

1. the system is in steady state;
2. the state space distribution is "tight" in that only a few, adjacent states have high probability and so the expected cost is approximately the same as the cost of the expected number in system;
3. the number of customers M is sufficiently large so that $M - \bar{N} \approx M$, where $\bar{N}$ is the expected number of customers in the system; and
4. the system is sufficiently loaded so that all servers are busy.

Assumption (1) is reasonable if T is sufficiently large. Our studies of the HotRod system reveal that the steady state assumption is reasonable for $T \geq 1$ second. Indeed, repeating the analysis in Figure 5 for $T = 1$, we obtain results that are almost identical with $T = 10$. However, a poor fits is obtained for $T = 0.1$. Assumption (3) is consistent with the findings in Figure 4 where $M = 140$ provides the best fit and $\bar{N}$ is small compared to 140. Assumption (4) is, most likely, consistent with optimizing $c(m)$ with respect to m in that idle servers are costly (assuming that h is not very small).

Our strategy is to use these assumptions to approximate the system in Figure 5 by an $M/M/1$ queueing system. We begin by defining the effective arrival rate.

$$\hat{\lambda} = \lambda M \tag{7}$$

$\hat{\lambda}$ will be used as the external arrival rate in $M/M/1$. Note that the definition of effective arrival rate is consistent with assumption (3) above.

The effective service rate must consider failures and repairs. Consider the $M/M/m/K/M$ queueing system in Figure 3. We can interpret this as a machine repair model in which the customers are machines that fail and the servers are repair stations. Thus, if $M = m = K$, the number in system is the expected number of non-failed machines. Substituting into Equation (1) and doing some straight-forward calculations, we determine that the effective number of servers is $m\frac{\beta}{\alpha+\beta}$. So, the effective service rate is

$$\hat{\mu} = \mu \frac{\beta}{\alpha + \beta} \tag{8}$$

The effective traffic intensity is

$$\hat{r} = \frac{\hat{\lambda}}{\hat{\mu}} \tag{9}$$

which is the number of servers required to carry the offered load. Effective utilization is

$$\hat{\rho} = \frac{\hat{r}}{m} \tag{10}$$

From this, we use $M/M/1$ to approximate the number in system for m servers

$$\begin{aligned} \hat{N}(m) &= \frac{\hat{\rho}}{1-\hat{\rho}} \\ &= \frac{\hat{r}}{m-\hat{r}} \end{aligned} \tag{11}$$

Next, we approximate the QoS cost using $1 - e^{-a\hat{N}(m)T}$. To obtain a closed form solution, we further simplify matters by using a linear approximation to the exponential near $\bar{N}$, an arbitrary value of $\hat{N}(m)$. (We will show shortly how to determine $\bar{N}$.) Thus,

$$\hat{c}_Q(m) = 1 - e^{-a\bar{N}T} + aTe^{-a\bar{N}T}\hat{N}(m) \tag{12}$$

and so

$$\begin{aligned} \hat{c}(m) &= 1 - e^{-a\bar{N}T} + aTe^{-a\bar{N}T}\hat{N}(m) + hTm \\ & 1 - e^{-a\bar{N}T} + aTe^{-a\bar{N}T}\frac{\hat{r}}{m-\hat{r}} + hTm \end{aligned}$$

Treating m as continuous, we have

$$\frac{d\hat{c}(m)}{dm} = aTe^{-a\bar{N}T}\frac{-\hat{r}}{(m-\hat{r})^2} + hT$$

Setting this to 0 and solving for m^*, the local minimum we have

$$m^* = \hat{r} \pm \sqrt{\frac{\hat{r}ae^{-a\bar{N}T}}{h}} \tag{13}$$

In theory, there are two solutions for m^*. However, incorporating the constraint that $m^* > 0$ often eliminates the solution $m^* = \hat{r} - \sqrt{\frac{\hat{r}ae^{-a\bar{N}T}}{h}}$. When there are two feasible solutions, we choose the one that provides the smallest total cost.

Equation (13) is consistent with our intuition. We see that m^* increases with effective traffic intensity $\hat{r}$ since more servers are needed to address the QoS costs at a higher loads. Also, m^* decreases with h (at least for the more common case of $m^* = \hat{r} + \sqrt{\frac{\hat{r}ae^{-a\bar{N}T}}{h}}$) since we use fewer servers if they are more expensive. Last, observe that $\lambda, M, \alpha, \beta, \mu$ do not appear separately in Equation (13). Rather, they only contribute to m^* through $\hat{r}$.

Now we return to the issue of estimating $\bar{N}$. Our observation is that we are controlling the system in a way so that the number in system should be near $N(m^*) \approx \hat{N}(m^*)$. Thus, we can use fixed point iteration as follows. The j-th value of m^* is obtained using the $j-1$ value of $\bar{N}$. The j-th value of $\bar{N}$ is obtained by computing $\bar{N}$ using the j-th value of m^*. The iteration continues until there is little change in m^*. Figure 7 displays an algorithm that summarizes how we compute m^*.

```
1. $\bar{N} = 0$; // Initial value
2. $m^* = -10$; // An impossible value
3. Do $i = 1$ to Limit
    (a) $m^*_L = \hat{r} - \sqrt{\frac{\hat{r}ae^{-a\bar{N}T}}{h}}$
    (b) $m^*_H = \hat{r} + \sqrt{\frac{\hat{r}ae^{-a\bar{N}T}}{h}}$
    (c) If $m^*_L > 0$ & $c(m^*_L) < c(m^*_H)$, then $m' = m^*_L$. Else $m' = m^*_H$
    (d) If $|m' - m^*|$ is small, then break
    (e) $m^* = m'$
    (f) $\bar{N} = \frac{\hat{r}}{m^* - \hat{r}}$
4. End
5. Return($m^*$)
```

Figure 7: Algorithm for estimating m^*, the optimal number of servers for the QoS metric customer seconds. The algorithm takes as input the cost parameters a and h along with effective traffic intensity $\hat{r}$ and the interval duration T.

Figure 8 plots $c(m)$ for several combinations of parameters. The square marker is the value of m^* estimated by Figure 7. We see that the algorithm in Figure 7 produces good estimates of m^* for a wide range of parameters. We note in passing that Figure 8 provides a justification for our definition $\hat{\lambda} = \lambda M$ in that there is very little difference between Figure 8(b) where $M = 50$ and $\lambda = 0.02$ and Figure 8(c) where $M = 100$ and $\lambda = 0.01$.

Having addressed the issue of how the Application Controller determines the number of servers to use, we now consider how utility owners determine which servers to purchase and how server manufacturers should handle trade-offs between server characteristics. Our approach is to assume that utility owners prefer servers with characteristics such that the Application Providers have a lower cost at their m^*. Thus, we want to compare server characteristics at $c(m^*)$. Substituting Equation (13) into Equation (6) for $I = 1$ and using Equation (11) and Equation (10), we have

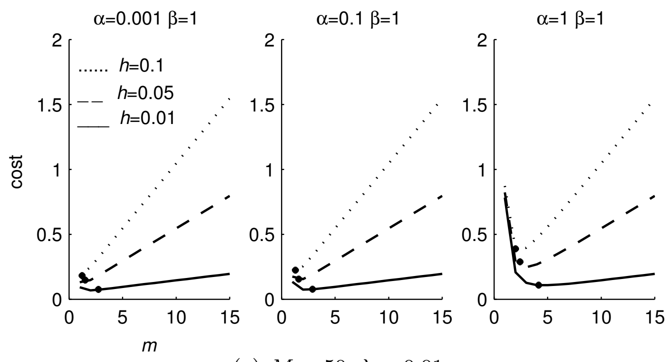
$$c(m^*) = 1 - e^{-T\sqrt{\hat{r}h}\sqrt{ae^{a\bar{N}T}}} + T\hat{r}h + T\sqrt{\hat{r}h}\sqrt{ae^{-a\bar{N}T}}$$

if $m^* = \hat{r} + \sqrt{\frac{\hat{r}ae^{-a\bar{N}T}}{h}}$ (which is the most common case in our experience). Note that $c(m^*)$ is increasing in $\hat{r}h = \hat{\lambda}\hat{v}$, where

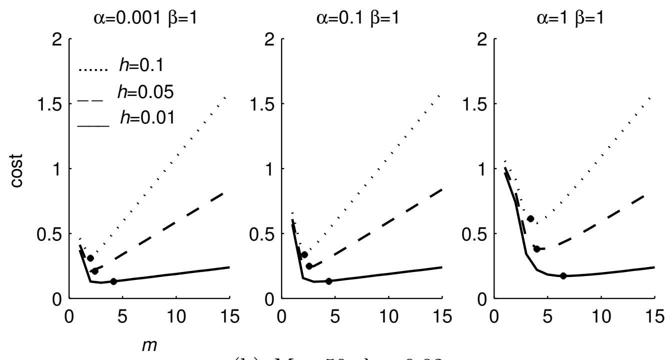
$$\hat{v} = \frac{(\alpha + \beta)\, h}{\mu\beta} \tag{14}$$

We refer to $\hat{v}$ as the **effective price performance** of the server. That is, $\hat{v}$ is the ratio of the server's price to its service rate μ adjusted for failures α and repairs β. Observe that by choosing servers with the smallest $\hat{v}$, the utility owner minimizes $c(m^*)$. Further, manufacturers of server hardware and software should make design trade-offs based on minimizing $\hat{v}$ so that their products are more appealing to utility owners.

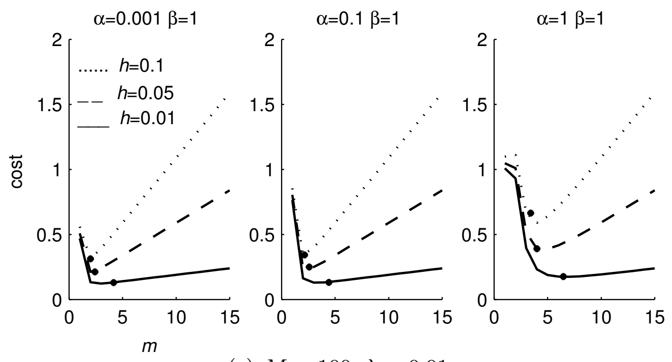
We present an example to illustrate the application of Equation (14) in making design trade-offs. Suppose that a middleware product has $1/\mu = 0.03$ seconds as in HotRod, $\alpha = 1/720$ per hour and $\beta = 1/30$ per minute as in software rejuvenation [26], and $h = 0.1$ as obtained in Section 3. Two designs are proposed to improve the product's position in the market. The first reduces the failure rate and is able to reduce the product's cost as well, although the service rate is decreased: $h = 0.09, 1/\mu = 0.04, \alpha = 1/1000, \beta = 1/30$. The second design actually increases the failure rate and decreases the repair rate; however, the service rate is increased, and price is reduced further: $h = 0.08, 1/\mu = 0.025, \alpha = 1/600, \beta = 1/50$. Applying Equation (14), we see that the original $\hat{v}$ is 0.003. The first design results in $\hat{v} = 0.0036$. That is, the effective price performance is worse than the original product. The second design has $\hat{v} = 0.002$, which is an



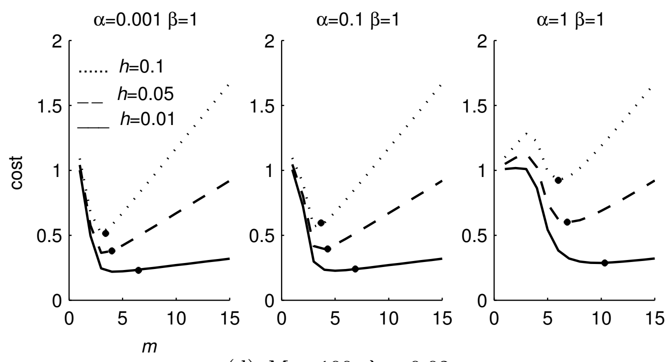
(a) $M = 50$, $\lambda = 0.01$



(b) $M = 50$, $\lambda = 0.02$



(c) $M = 100$, $\lambda = 0.01$



(d) $M = 100$, $\lambda = 0.02$

Figure 8: Accuracy of m^* approximation for cost curves with $a = 0.1$ and $T = 1$. The square marker is m^* estimated by Figure 7.

improvement over the original product.

5 Conclusions

This paper develops a framework for capacity management in computing utilities for application clusters where Application Providers may want to add or remove servers fairly frequently in order to respond to changing workloads. In particular, we study how to minimize total cost, the sum of server holding costs and quality of service (QoS) costs due to SLA penalties as a result of poor performance; the minimization is done with considerations for server failures and repairs (where server capabilities refer to both hardware and software). Based on studies that show a good fit between response times obtained from a product-level testbed and estimates from an $M/M/m/M/M$ queueing system, we develop a performance and availability model of clustered systems for both homogeneous and heterogeneous server characteristics (i.e., service rate, failure rate, and repair rate) that augments $M/M/m/M/M$ by including transitions for failures and repairs. We then construct a cost model that expresses the trade-off between QoS costs and server holding costs. QoS costs are expressed using the QoS cost rate parameter a that characterizes the steepness of the staircase function used to represent SLA penalties. Server holding costs employ a single parameter h that represents the relationship between server costs and the maximum SLA penality. We develop a closed-form approximate solution for the optimal number of servers that works well for a wide range of system parameters. This result indicates that the optimal number of servers m^* is the effective traffic intensity (the number of servers consumed by the workload with considerations for failures and repairs) with an adjustment for QoS costs and server holding costs. Last, we show that the Application Provider cost at m^* is an increasing function of $\hat{v}$, the effective server price performance (which considers failures and repairs). Thus, we propose that utility owners should consider $\hat{v}$ in their purchasing decisions, and manufacturers of servers should assess design trade-offs in terms of their impact on $\hat{v}$.

Much work remains. We hope to generalize our optimization result to multiple server types using the performance and availability model presented in this paper. Also, our work to date has not considered the impact of forecast variability on the optimal number of servers for Application Providers. Further, thus far, we have only considered one Application Provider in isolation, not a set of Application Providers competing for servers.

References

[1] Farshid Maghami Asl and A. Galip Ulsoy. Capacity management in reconfigurable manufacturing systems with stochastic market demand. In *ASME International Mechanical Engineering Contress and Exposition*, 2002.

[2] Benita Beamon and Jose Bermudo. A hybrid push/pull control algorithm for multistage, multiline production systems. *Production Planning and Control*, 11(4), 2000.

[3] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *ACM Sigmetrics*, 2003.

[4] Chang Ho Choi and Sung Jo Kim. A call admission control mechanism using mobility graph in mobile multimedia networks. In *Wireless Personal Communications*, volume 25, 2003.

[5] Standard Performance Evaluation Corporation. SpecJAPPServer2002 (JAVA Server Application). 2002.

[6] Olivia Das and C. Murray Woodside. Evaluating layered distributed software systems with fault-tolerant features. *Performance Evaulation*, 45, 2001.

[7] V.D. Dinopoulou and C. Melolidakis. Asymptotically optimal component assembly plans in repairable systems and server allocation in parallel multiserver queues. *Naval Research Logistics*, 48(8), 2001.

[8] Ed Frauenheim. Desktop service debuts at eds. In *News.Com*, August 2003.

[9] E. Gelenbe and I. Mitrani. *Analysis and Synthesis of Computer Systems*. Academic Press, 1980.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SOSP*, 2003.

[11] Joseph L. Hellerstein. Rules of thumb for selecting metrics for detecting performance problems. In *Proceedings of the Computer Measurement Group*, 1996.

[12] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, April 2001.

[13] Dimitri Kececioglu. *Reliability Engineering Handbook*. Prentice Hall, 1st edition, 1991.

[14] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1), March 2003.

[15] K. H. (Kane) Kim. Toward globally optimal resource management in large-scale real-time distributed computer systems. In *IEEE CS 6th Workshop on Future Trends of Distributed Computing Systems*, October 1997.

[16] Leonard Kleinrock. *Queueing Systems*. Wiley-Interscience, 2nd edition, 1975.

[17] Ed Lassettre, David W. Coleman, Yixin Diao, Steven Froehlich, Joseph L. Hellerstein, Lawrence S. Hsiung, Todd W. Mummert, Mukund Raghavachari, Geoffrey Parker, Lance Russell, Maheswaran Surendra, Veronica Tseng, Noshir Wadia, and Peng Ye. Dynamic surge protection: An approach to handling unexpected workload surges with resource actions that have lead times. In *Distributed Systems Operations and Management*, 2003.

[18] G. Levitin and L. Meizin. Structure optimization for continuous production systems with buffers under reliability constraints. *International Journal of Production Economics*, 70(1), 2001.

[19] Christopher R. Lumb, Arif Merchant, and Guillermo A. Alvarez. Facade: Virtual storage devices with performance guarantees. In *Proceeding of the 2nd USENIX Conference on File and Storage Technologies*, March 2003.

[20] John Markov. Ibm says it benefits 2 ways from on-demand computing. In *New York Times*, November 2003.

[21] Daniel A. Menasce, Daniel Barbara, and Ronald Dodge. Preserving qos of e-commerce sites through self-tuning: A performance model approach. In *Proceedings of the 3rd Conference on eCommerce*, 2001.

[22] John F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, 29(8), August 1980.

[23] I. Mitrani. Queues with breakdowns. In *Performability Modelling: Tools and Techniques*. Wiley, 2001.

[24] G.A. Paleologo. Price-at-risk: A methodology for pricing utility computing services. *IBM Systems Journal*, 43, 2004.

[25] K.S. Trivedi, S. Dharmaraja, and Xiaomin Ma. Analytic modeling of handoffs in wireless cellular networks. *Information Sciences*, 148, 2002.

[26] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi. Analysis and implementation of software rejuvenation in cluster systems. In *ACM Sigmetrics*, 2001.

[27] Christopher Ward, Melissa J. Buco, Rong N. Chang, and Laura Z. Luan. A generic sla semantic model for the execution management of e-business. In *Third International Conference on E-Commerce and Web Technologies*, September 2002.

[28] Gehan Weerasinghe, Imad Antonios, and Lester Lipsky. A generalized analytic performance model of distributed systems that perform n tasks using p fault-prone processors. In *15th International Conference on Parallel and Distributed Computing*, 2002.

[29] Katinka Wolter and Andrea Zisowsky. On markov reward modelling with fspns. In *IEEE International Computer Performance and Dependability Symposium*, 2000.

[30] Sang-Jo Yoo and Myungchul Kyung-Sup Kim. Predictive and measurement-based dynamic resource management and qos control for videos. *Computer Communications*, 26, 2003.

[31] A.I. Zreikat, B. Bolch, and J. Sztrik. Performance modelling of nonhomogeneous unreliable multiserver systems using mosel. *Computers and Mathematics with Applications*, 46, 2003.