Optimization of Quality of Service in Dynamic Systems *

Jeffery P. Hansen
Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, PA 15213
Email: *hansen@cmu.edu*

John Lehoczky
Department of Statistics
Carnegie Mellon University
Pittsburgh, PA 15213
Email: *jpl@stat.cmu.edu*

Ragunathan Rajkumar
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
Email: *raj@ece.cmu.edu*

Abstract

Resource allocation policies which significantly improve Quality of Service (QoS) while minimizing QoS instability are presented. In a dynamic system in which applications are continuously entering and departing the system, QoS instability occurs when the system over-commits resources and can not meet the resource demands of new admission requests. The system is forced to either reject the request or degrade one or more existing tasks. In this paper we introduce three admission control policies and compare their QoS performance/stability tradeoffs. We show that by maintaining a small resource reserve modeled as a competing application in the QoS optimization framework, we can achieve QoS levels that are over 90% of the theoretical maximum while reducing instability by one to two orders of magnitude.

1 Introduction

The need for cost-effective higher performance solutions to system design has necessitated a shift from the traditional stovepipe design to a modular QoS (Quality of Service) managed design. Rather than dedicated computers for each task in the system, applications are dynamically mapped to resources from a heterogeneous pool of computing resources. For example, in the Navy's SC21 (Surface Combatant 21st Century) initiative the goal is to integrate all command and control functionality into a single distributed system. The modular design results in systems that are more reliable and less expensive to build and maintain.

Making effective use of these resources in a dynamic system with imperfect knowledge is a complex and difficult task. We can not always predict when requests for resources will arrive or how long the resources will be required. Resource allocation decisions may have a lasting impact on the performance of resource allocation policies if tasks remain in the system for a long time. For example, while allocating all of the system resources to tasks currently in the system may yield a better instantaneous QoS, the resulting QoS instability, where the QoS level is constantly changing, may make this undesirable.

In this paper we consider a set of tasks which can operate at varying levels of QoS. For example a video conference can operate at multiple frame rates, multiple resolutions, multiple security levels, etc. We also assume a model where tasks randomly join the system requesting resources, and depart the system when they are done. The goal is to choose a resource allocation for these tasks that will deliver the highest total QoS while minimizing variation of the QoS levels.

2 Related Work

Most QoS systems use various mechanisms to reserve resources for tasks. Work on RSVP[11] for network bandwidth reservation is one of the earlier ef-

*This work supported in part by contracts DARPA N66001-97-C-8527, ONR N0014-92-J-1524 and DARPA F33615-00-C-1729

forts. RSVP allows applications to reserve a fraction of the link bandwidth along a multi-cast path to be used exclusively by that application. Another example of a bandwidth reservation approach is the hierarchical scheduler in Darwin[1]. Darwin improves on the RSVP approach by allowing a hierarchy of reservations. This allows for greater control over which tasks can use left-over reserved resources not used by the reserving task.

QoS mechanisms have also been developed for CPU and memory. The resource kernel[7][9] allows the system to make sets of resources organized as a virtual machine to be reserved for specific tasks or sets of tasks. Taken together with the network QoS mechanism, the resource kernel mechanisms can be controlled by a resource manager to control the overall QoS delivered to tasks managed by the system.

Higher-level resource management tools such as DeSiDiRaTa[12] and DQM (Dynamic QoS Manager)[4] have also been proposed. DeSiDiRaTa monitors the QoS delivered to managed tasks and compares it with the contracted QoS. If there is a deviation, diagnosis and action components take corrective action. The action could be a change in the resources allocated to a task, or a change in task QoS. DQM is geared toward self-regulating tasks in a best-effort environment. When task detect that they are not receiving enough resources to maintain the current QoS level (i.e., deadlines are being missed) then the task changes its QoS set-point by negotiating with DQM.

The dynamic QoS optimization algorithms discussed in this paper are designed to be a key component in the Amaranth system [2][3]. Amaranth provides a framework for exploring policy issues associated with QoS management. Policies such as the Q-RAM based QoS optimization algorithm can be plugged into Amaranth and the effects on system performance measured. Amaranth also provides for additional policies such as stochastic enforcement of deadlines[6] and probabilistic bandwidth reservation.

3 Static QoS Optimization

The dynamic resource allocation algorithms discussed in this paper are based on extending the Q-RAM (QoS Resource Allocation Model) static resource allocation algorithms[5][8]. In the Q-RAM approach, tasks are modeled by providing a mapping between QoS and resource requirements, and a mapping between QoS "utility", a measure of the desirability a particular QoS level. Q-RAM can quickly find a resource allocation which maximizes the global utility (the sum of the task utilities) while meeting resource constraints. The first step of the Q-RAM algorithm is to factor out the QoS component and combine the two mappings into a single resource to utility mapping. The algorithm then utilizes the fact that the Kuhn-Tucker condition must hold at the optimal point. That is, the slope or marginal utility for all tasks must be equal. The algorithm proceeds by setting all algorithms to their minimum allocation and then adding resources to the task with the highest marginal utility until the resources have been completely allocated. Utility curves are generally assumed to be concave and non-decreasing.

4 Dynamic Traffic Model

In the dynamic QoS optimization problem, the task set is not fixed and tasks are continuously arriving and departing the system. We model the system environment as a finite set of N independent tasks which randomly alternate between an "idle" state and an "active" state. Conceptually this can be thought of as a set of users, who periodically log on to the system, use resources, then exit the system. The traffic model is characterized by the following assumptions:

- Tasks can be in one of two states: idle or active. Holding periods in the idle state have cumulative distribution function (c.d.f.) F_I with mean $1/\lambda$, while holding periods in the active state have c.d.f. F_A with mean $1/\mu$.
- All tasks behave independently of each other.
- The amount of resources a task can use is continuous in the range 0 to 1.
- Tasks have identical utility functions $U(r)$. U is strictly increasing and concave over the region $0 \leq r \leq 1$, and $U(r) = U(1)$ if $r \geq 1$, hence there is no benefit from allocating more than 1 unit of resource to any task.
- There is a total of R units of resources, there are N tasks in total, and we assume $N > R$.

A task stays in the idle state for a random period of time governed by a c.d.f. $F_I(x)$ with mean $1/\lambda$. At the end of the idle period, a task becomes active and requests resources. We assume that the task will always receive at least the minimum amount of resources necessary to execute, although it may receive more and execute at a larger QoS level. There is also a maximum resource allocation beyond which the task receives no additional utility. The task stays in the active state for a random period of time governed by a c.d.f. $F_A(x)$ having mean $1/\mu$. The holding time in the active state is assumed to be independent of the

amount of resources allocated to it. Upon completion of the execution, the task returns to the idle state, and this cycle repeats indefinitely with future cycle holding times in the two states being independent of all past holding times. Two important parameters are the traffic intensity $\rho = \frac{\lambda}{\mu}$ representing the relative rate at which tasks enter the active state and $\theta = \frac{\rho}{1+\rho}$ representing the equilibrium probability that a task is in the active state.

We let $X(t)$ be the number of active tasks (out of N) at time t. If the holding time distributions F_I and F_A are both exponential, then $\{X_t,\ t \geq 0\}$ is a birth death process corresponding to a standard repairman model with an infinite server. The equilibrium distribution for this model is well-known to be given by a Binomial(N, θ) distribution, that is if X_∞ is a random variable with the equilibrium distribution, then

$$P(X_\infty = k) = \binom{N}{k} \theta^k (1-\theta)^{N-k} \qquad (1)$$

where $\binom{N}{k} = \frac{N!}{k!(N-k)!}$. Recall $E(X_\infty) = N\theta$ and $\mathrm{Var}(X_\infty) = N\theta(1-\theta)$.

It is interesting to note that this equilibrium distribution is appropriate even if the holding time distributions are not exponential but are non-lattice. Each task forms an alternating renewal process. Since the equilibrium probability that a particular task is active is θ, and since the tasks are assumed to be independent, the total number that are active will have a Binomial(N, θ) distribution.

5 Dynamic QoS Optimization

Given this framework, we next consider the resource allocation policies that are used when a task becomes active or idle. We consider two performance metrics for resource allocation policies: (1) average global utility U_g, and (2) reconfiguration rate ϕ. The average global utility is defined as the integral of the instantaneous global utility with respect to time divided by the time period through which it is measured. The reconfiguration rate is the total number of times the system changes the QoS level of any active task divided by the time period through which it is measured. Reconfigurations are undesirable because they can be distracting to human users. For example a video conference that maintains a constant 20 fps (frames per second) is more likely to be desirable than one that starts at 20 fps for a few minutes, then switches to 30 fps, then to 15 fps and so on, even if the average frame rate is higher than 20 fps.

In general, we want average global utility to be high and the reconfiguration rate to be low. There are two extreme allocation policies. We could always make minimal resource allocations to tasks, thus never having to cause any reconfigurations. Conversely, we could always make maximal resource allocations, and reduce them when resource demands exceed the residual resource supply.

5.1 Homogeneous Allocation Policies

We next consider resource allocation policies. We begin by considering a special class of policies for which we can analytically calculate the performance measures. This class is characterized by the assumption that all active tasks will always have identical resource allocations. We refer to these policies as "homogeneous allocation policies." Given that the number of active tasks in the system forms a Markov chain, without any loss in generality, one can base the resource allocation policy on the number of active tasks in the system. Consequently, one can represent all memoryless homogeneous allocation policies by vectors of dimension N, $(x_1, x_2, \ldots, x_N)$ where x_k refers to the resources allocated to each of the k tasks when there are k active tasks in the system. We define a fesible policy to be one where $0 \leq x_k \leq \min(1, \frac{R}{k})$ hold for $1 \leq k \leq N$. The long run total system utility generated by any homogeneous allocation policy is given by

$$\sum_{k=1}^{N} kU(x_k) \binom{N}{k} \theta^k (1-\theta)^{N-k} \qquad (2)$$

A homogeneous allocation policy causes a reconfiguration when the number of tasks in the system transitions from k to $k+1$ and $x_k \neq x_{k+1}$. The arrival of a new task calls for different resource allocations for the k tasks that are running, thus k reconfigurations occur. Similarly, if the system transitions from $k+1$ to k and $x_k \neq x_{k+1}$, then the k tasks remaining in the system will be reconfigured by receiving new resource allocations. Note that in our model in which only one step transitions can occur, the number of transitions from k to $k+1$ must always be within 1 of the number of transitions from $k+1$ to k. Thus for homogeneous allocation policies, the rate of reconfigurations caused by arrivals of tasks to the system must equal the rate of reconfigurations caused by the departure of tasks, so long as F_I and F_A are not concentrated on a lattice. For exponential holding times, the long run rate of reconfigurations for a policy is given by

$$2 \sum_{k=1}^{N-1} \lambda(N-k) k I_{\{x_k \neq x_{k+1}\}} \binom{N}{k} \theta^k (1-\theta)^{N-k} \qquad (3)$$

where $I_{\{C\}} = 0$ or 1 depending whether condition C holds or not.

Note, we have combined the two reconfiguration terms together and given them equal weight. One could easily keep them separate or combine them using some other weighting.

Within the class of homogeneous allocation policies, we can, without any loss in generality, further restrict these policies. Let M be the integer satisfying $M \leq R < M+1$. If $M \geq N$, then all N tasks can be allocated 1 unit, and the allocation $x_k = 1$ for all k is optimal. Furthermore, no reconfigurations will occur. More realistically, suppose $M < N$ and the policy $x_k = 1$ for all k is not feasible. We restrict attention to this case.

There are two results available for homogeneous allocation policies:

Theorem 1
If a feasible homogeneous allocation policy $(x_1, \ldots, x_N)$ satisfies $x_k < x_{k+1}$ for at least one k, $1 \leq k \leq N$, then there is a different homogeneous allocation policy satisfying $x_k \geq x_{k+1}$ for all $1 \leq k \leq N$ which has greater long run total system utility and no greater reconfiguration rate. □

Proof
Consider any feasible homogeneous allocation policy that does not satisfy the non-increasing allocation property. Let i be the smallest integer for which $x_i > x_{i+1}$ and there exists $j > i+1$ with $x_j > x_{i+1}$. Chose j to be as small as possible. Such an i and j must exist because the allocation policy is assumed to not satisfy the non-increasing property. Note that since j was chosen to be minimal, it follows that $x_{i+1} \geq x_k$ for $i+1 \leq k < j$. Consider the modified policy with $x_k = x_j$ for all $i+1 \leq k \leq j$ and the other allocations unchanged. The new allocations are larger than the old for $i+1 \leq k < j$ and otherwise unchanged, so the long run system utility is increased. Since the new policy has identical allocations for $i+1 \leq k \leq j$, none of these are reconfiguration points. Thus the number of new reconfiguration points is less with the new policy than with the old. Therefore the new policy increases total system utility and reduces reconfiguration rate. By applying this argument to every non-decreasing resource allocation, we will improve performance and end up with a non-increasing homogeneous allocation policy. □

Theorem 1 indicates that we can restrict attention to homogeneous allocation policies in which the allocations are non-increasing as the number of tasks in the system increases.

Further restrictions are possible. Define the set $\mathcal{S} = \{1, \frac{R}{M+1}, \frac{R}{M+2}, \ldots, \frac{R}{N}\}$. All resource allocations must be drawn from $\mathcal{S}$, or the policy can be improved.

Theorem 2
Suppose a feasible homogeneous allocation calls for an allocation $x_k \notin \mathcal{S}$ for some $1 \leq k \leq N$. Then there is another feasible homogeneous policy with allocations in $\mathcal{S}$ which has larger long run total system utility and no greater reconfiguration rate.□

Proof
Consider any homogeneous allocation policy with feasible assignments $0 \leq x_k \leq \min(1, \frac{R}{k})$, $1 \leq k \leq N$. Suppose $x_k \notin \mathcal{S}$ for some $1 \leq k \leq N$. Increase x_k to the nearest element of $\mathcal{S}$. That is, if $\frac{R}{M+i+1} < x_k < \frac{R}{M+i}$, then increase x_k to $\frac{R}{M+i}$. If $\frac{R}{M+1} < x_k < 1$, then increase x_k to 1. Clearly the new policy increases resources and thus will generate greater long run system utility. No greater (and possibly much less) reconfigurations will occur since if $x_i = x_{i+1}$ under the old allocation, this will also be true under the new allocation. Thus, any homogeneous allocation policy with allocation value not in $\mathcal{S}$ can be improved on both performance dimensions. □

5.1.1 Basic Policy

We now consider a restricted special case of the general homogeneous allocation policy which we will call the "basic" policy and denote it by $P_B(A_1, A_2)$. The policy is characterized by two coefficients A_1 and A_2 where $A_1 \leq R \leq A_2$. When the number of active tasks is less than or equal to A_1, all tasks will be given their maximum resource allocation (i.e., 1 unit), when the number of active tasks is greater than A_1 and less than or equal to A_2, then all tasks will be given R/A_2 units of resource, and when the number of active tasks is greater than A_2 the tasks will consume all of the remaining resources equally. Figure 1 shows a graph of the resource allocation for each task as a function of the number of active tasks for the case $N = 50, R = 20$. Two important special cases of the basic policy are the always optimal policy $P_B(0, R)$ and the minimum-reconfiguration policy $P_B(0, N)$.

Always Optimal The Always Optimal policy is a homogeneous allocation policy for which maximum feasible allocations are made, i.e. $x_k = \min(1, \frac{R}{k})$. It is optimal with respect to long run system utility, because the tasks have a common utility function which is concave, hence the always optimal allocation satisfies the Kuhn-Tucker conditions and is globally optimal. It can, however, have a large reconfiguration cost.

Minimum-Reconfiguration Policy This policy allocates $\frac{R}{N}$ to every task. Consequently, no task is

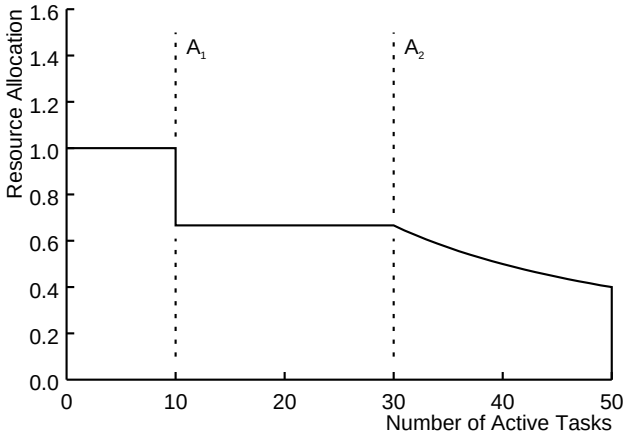

Figure 1. Basic Policy

ever reconfigured. The long run total system utility generated is given by the average number of tasks in the system times the utility generated by those tasks, namely $N\theta U(\frac{R}{N})$.

5.2 Non-Homogeneous Allocation Policies

Suppose at any time t there are $X(t)$ active tasks, and these tasks have been allocated a total of A units of the resource. Because tasks have identical concave utility functions, the equal allocation policy, $A/X(t)$, will maximize the total system utility. This is the principle underlying the use of these policies. The main problem is that forcing all tasks to always have the same resource allocation can result in high reconfiguration rates or conversely low average global utilities. In this section we consider non-homogeneous allocation policies in which some tasks may have different allocations than others. While non-homogeneous algorithms are generally not optimal when considering global utility alone, they generally provide better utility/stability tradeoffs.

5.2.1 Reserve Resource Policy

One condition which can result in frequent reconfigurations is when all or most of the resources have been allocated. When this occurs, there are insufficient resources to meet even the minimum requirements of an incoming task triggering reconfigurations in the other tasks. One way to avoid this is to keep some of the resources unallocated as a reserve. In our reserve-resource policy $P_R(W, \alpha, N_r, T_l)$ we do this by modeling the reserve resources as an additional pseudo-task which receives utility for keeping unused resources. We define the utility function for the reserve resources pseudo-task to be:

$$U_{rr}(r) = \begin{cases} \frac{1}{1-e^{-\alpha}}(1 - e^{-\alpha r/R}) & 0 \le r \le R \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

This results in the reserve pseudo-task receiving a high marginal utility when there are few reserve resources remaining, and a low marginal utility when there is a large amount of resources remaining.

When there are "adequate" reserves and a new task becomes active, the reserve-resources policy proceeds by using Q-RAM to optimize the joint utility of the active task and the reserve pseudo-task. When the reserves are not adequate, N_r of the existing tasks are reconfigured and their new resource allocations determined by using Q-RAM to optimize the joint utility of the N_r tasks, the incoming task and the reserve pseudo-task. The reserve is deemed to be adequate if there are at least $T_l R$ units of unallocated resources where T_l represents the low-water level.

If all tasks have a utility function of the form:

$$U_t(r) = \begin{cases} \frac{1}{1-e^{-1}}(1 - e^{-r}) & 0 \le r \le 1 \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

then it is possible to analytically determine points r_0 and r_1 such that when the unallocated resources are r_0 or lower, zero (or minimal) resources will be allocated to the incoming task, and when the unallocated resources are r_1 or higher, then the maximum resources will be allocated to the incoming task. These points can be determined by exploiting the Kuhn-Tucker condition which holds at the optimal point. r_0 and r_1 are the points where $\frac{d\,U_{rr}}{dr}\big|_{r=r_0} = \frac{d\,U_t}{dr}\big|_{r=0}$ and $\frac{d\,U_{rr}}{dr}\big|_{r=r_1} = \frac{d\,U_t}{dr}\big|_{r=1}$. solving for r_0 and r_1 yields $r_0 = \frac{-R}{\alpha} \ln\left(\frac{1}{W}\frac{R}{\alpha}\frac{1-e^{-\alpha}}{1-e^{-1}}\right)$ and $r_1 = r_0 + \frac{R}{\alpha}$.

Note that these equations imply that unless $\alpha \ge \frac{R}{R-r_0}$, no tasks will ever be given their maximum allocation. Note also that it is possible to specify the reserve pseudo-task utility function by specifying r_0 and r_1 and computing $\alpha = \frac{R}{r_1 - r_0}$ and $W = \frac{1-e^{-\alpha}}{1-e-1}\frac{R}{\alpha}e^{-\alpha r_0/R}$.

5.3 Bound Utility Policy

While the reserve resource policy can significantly reduce the reconfiguration rate for a given average global utility value compared to homogeneous allocation policies, there is no constraint on how far the global utility can deviate from optimal. For this reason we consider a modification to the reserve resource policy which we call the bound-utility policy $P_U(W, \alpha, B_m)$. The bound-utility policy guarantees that the instantaneous global utility is within at least B_m of the optimal utility.

In the bound utility policy, we first use Q-RAM to compute the utility U_{opt} which would be possible if it could select set-points for all of the existing tasks and the incoming task. Tasks are admitted in the same manner as in the reserve resource policy where the incoming task competes with a reserve resources pseudo-task. We use the same utility function as shown in Equation 4 for the bound utility policy as with the reserve resources policy. If the utility which would result is less than $U_{opt}B_m$, then we reconfigure the smallest number of tasks necessary to achieve a utility greater than $U_{opt}B_m$. The reconfigured tasks compete with the incoming task and the reserve pseudo-task for all of the resources not used by the other fixed tasks. When selecting tasks to reconfigure, we always choose tasks with the lowest marginal utility. This will result in the greatest release of resources for the smallest decrease in utility.

We also incorporate an exit policy to ensure that utility remains within the bound. If after a task departs (becomes inactive) the global utility becomes less than $U_{opt}B_m$, then we reconfigure the smallest number of tasks necessary to result in a global utility greater than $U_{opt}B_m$. Just as in an arrival triggered reconfiguration, the tasks compete against the reserve pseudo-task, but unlike in the arrival triggered reconfiguration we select tasks having the highest marginal utility. This gives us the largest gain in utility for the smallest amount of resources.

6 Experimental Results

In this section we compare the performance of the various admission control policies using simulation results. The simulator generates streams of admission and departure requests according to the model described in Section 4. In the simulation experiments we take the maximum amount of resources R to be 20, and we assume exponentially distributed arrivals and departures with $\lambda = \mu = 1$. We control the load by changing the total number of tasks N (idle and active). We simulate a low load case with $N = 35$, a medium load case with $N = 50$ and a high load case with $N = 65$. Note that the arrival λ and departure μ rates are on a per-task bases (e.g., if there are 12 idle tasks, the system arrival rate will be 12λ).

The majority of the results are presented using an Average Global Utility versus Reconfiguration Rate tradeoff curve. Each point on these curves represents an operating point for some policy with some settings of the policy parameters. Ideally we would like to be as high as possible on the Y-axis (high utility, zero reconfigurations). Many of the techniques used to analyze ROC curves[10] can be used to analyze these policy tradeoff curves. For example, if we associate a profit p_u and a cost c_r with utility and reconfigurations respectively, then a line of constant profit on the tradeoff plot would be a line of slope $\frac{c_r}{p_u}$. As the constant-profit line is translated down and to the right, the lines represent smaller profits, and as the lines shift up and to the left the lines represent larger profits.

The constant-profit line model gives us a tool for graphically comparing policies. We can imaging plotting points representing the measured performance points for a set of policies. We then mentally picture sliding a line of slope $\frac{c_r}{p_u}$ down and to the right until we touch the first point. This will be the point corresponding to the highest profit policy. Many policies have parameters which can be adjusted to yield a family of policies. This can be represented as a curve or set of curves on the tradeoff graph.

The tradeoff curve for the homogeneous policy $P_B(A_1, A_2)$ under a medium load (i.e., N=50) is shown in Figure 2. Each of the lines on this curve is for a particular value of A_1, and the points on each of the curves correspond to different values of A_2 with the largest values of A_2 corresponding to the lowest reconfiguration rates. The maximum utility value occurs when $A_1 = 0$ and $A_2 = 20$ yielding a utility of 21.57 and a reconfiguration rate of 1152. We can see that the curve for $A_1 = 0$ is above all of the other curves for reconfiguration rates between 0 and 10. This implies that for the medium load case, it is never advantageous to set A_1 to any value other than zero.

The medium-load tradeoff-curve for the reserve resource policy $P_R(W, \alpha, N_r, T_l)$ is shown in Figure 3. The low-water parameter was set at $T_l = 0.05$ for this experiment. Each curve is for different combinations of the number of tasks N_r to reconfigure on a resource shortfall and the exponential shape parameter α. Points on the curve correspond to different weight values for the reserve resource pseudo-task. Here we see that in general setting the number of tasks to reconfigure N_v to 1 generally results in better performance, and setting the shape parameter α to 1 also generally produces better performance, although there is a small region where $\alpha = 5$ yields better performance.

Figure 4 shows the medium-load tradeoff-curve for the bound utility policy $P_U(W, \alpha, B_m)$. In this plot, curves for different weight and α parameter values are shown, with the points on the curves representing values of the bound B_m ranging from 0.5 to 0.9. Through experimentation we have found that weight factors near 0.05 tend to result in the best performance. We also see that the for reconfiguration rates below $\phi = 8$, lower α values are preferred, but at higher reconfiguration

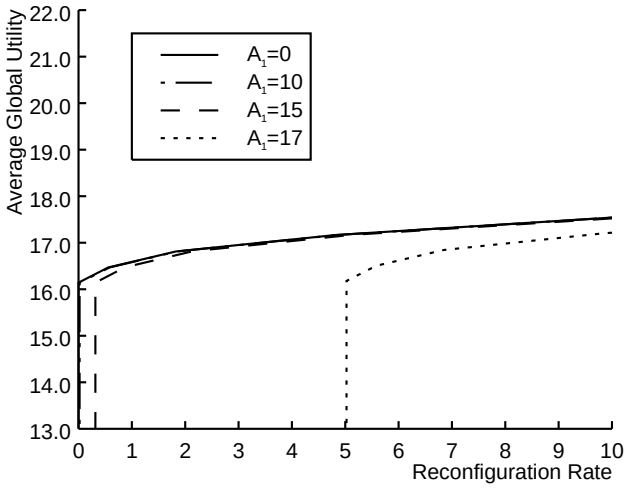

Figure 2. Homogeneous Policy Tradeoffs

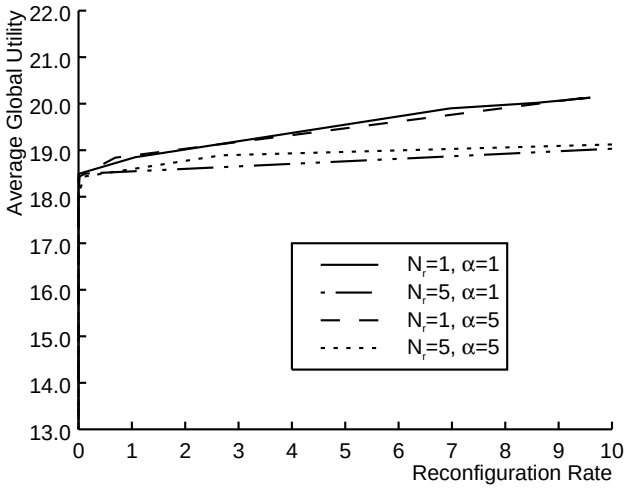

Figure 3. Reserve Resource Policy Tradeoffs

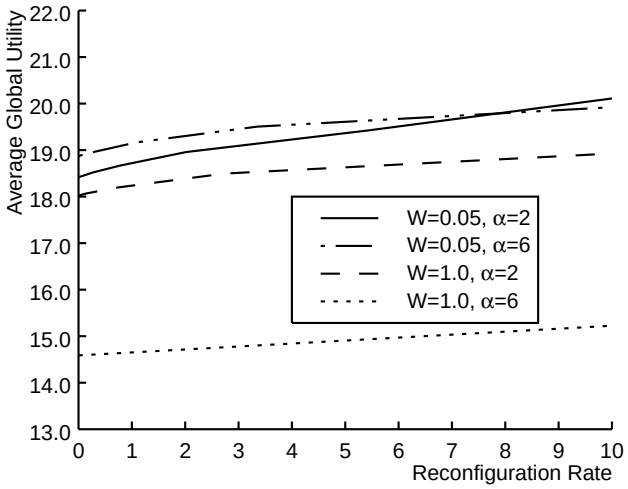

Figure 4. Bound Utility Policy Tradeoffs

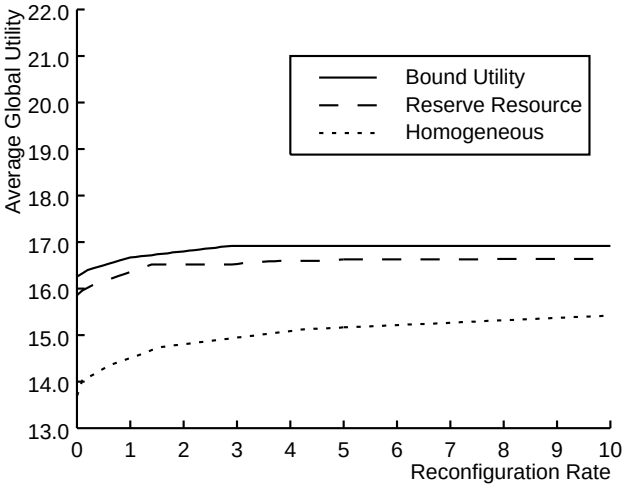

Figure 5. Low Load (N=35)

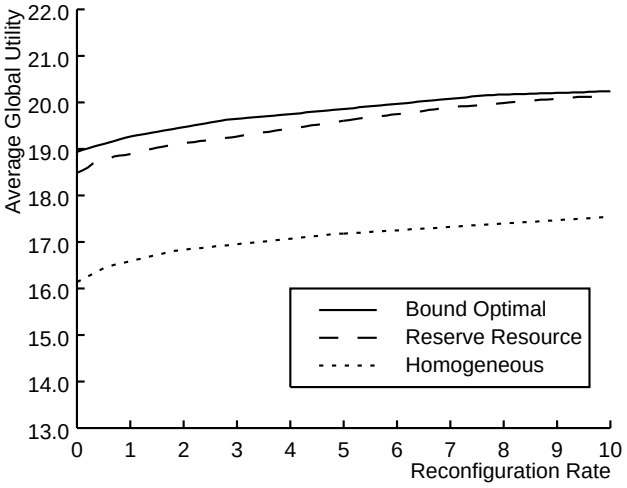

Figure 6. Medium Load (N=50)

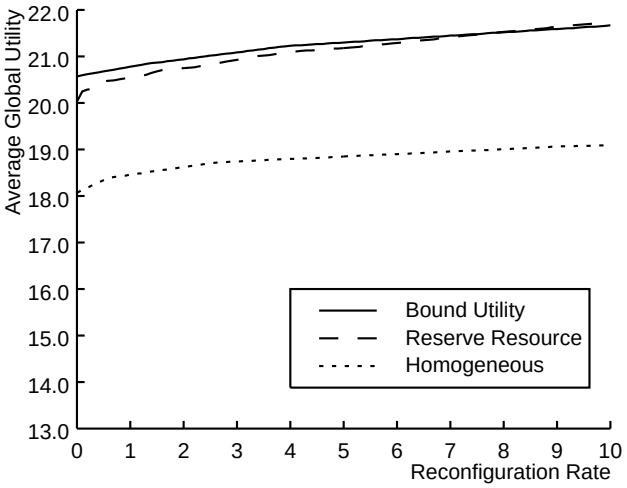

Figure 7. High Load (N=65)

	Always Optimal		Bound Utility	
Load	U	ϕ	U	ϕ
low	17.17	141	16.26	0
medium	21.57	1152	19.43	1.8
high	23.51	2079	21.16	3.5

Table 1. Always Optimal vs. Bound Utility

rates, higher α values are preferred.

Now let $\phi(P(x))$ be the reconfiguration rate resulting from using admission policy P with parameter setting x, and let $U_g(P(x))$ be the average global utility resulting from the same policy and setting. We define the policy performance curve of an admission policy to be the set of points $(\phi(P(x)), U_g(P(x))$ such that for all x', $\phi(P(x)) = \phi(P(x'))$ implies $U_g(P(x)) \geq U_g(P(x'))$. In other words, it is the curve that would be formed by plotting the performance points for all possible settings and drawing a line along the top edge of these points. The performance curve of a policy represents the best that a policy can do when its parameters are optimally tuned for a particular utility/reconfiguration rate tradeoff.

Policy performance curves for low, medium and high load conditions are shown in Figures 5, 6 and 7. The curves show that as expected, the non-homogeneous policies perform significantly better than the homogeneous policy, and that the bound utility policy generally performs slightly better than the reserve resource policy. Table 1 compares utility and reconfiguration rate for the always optimal policy with the bound utility policy at the point where it achieves 90% of the optimal utility value. Even under high load, there is a very significant reduction in the reconfiguration rate for a modest decrease in the global utility.

7. Conclusion

In this paper we have explored the performance characteristics of several admission control policies for dynamic optimization of QoS. We have shown that the selection of an admission policy can have significant impact on the performance. In general, we have shown that while homogeneous algorithms can achieve the highest global utility, when stability is considered, non-homogeneous provide better a utility/stability tradeoff.

References

[1] P. Chandra, A. Fisher, C. Kosak, T. S. Eugene Ng, P. Steenkiste, E. Takahashi, H. Zhang, "Darwin: Resource Management for Value-Added Customizable Network Service", From *6th IEEE International Conference on Network Protocols (ICNP'98)*, October 1998

[2] C.L. Hoover, J.P. Hansen, P. Koopman, S. Tamboli, "The Amaranth Framework: Policy-Based Quality of Service Management for High-Assurance Computing", Accepted for publication in *International Journal of Reliability, Quality, and Safety Engineering*

[3] C.L. Hoover, J.P. Hansen, P. Koopman, S. Tamboli, "The Amaranth Framework: Probabilistic, Utility-Based Quality of Service Management for High Assurance Computing", From *Proceedings of the 4th IEEE International High-Assurance Systems Engineering Symposium (HASE '99)*, November 1999

[4] G. Nutt, S. Brandt, A. Griff, S. Siewert, T. Berk, and M. Humphrey, "Dynamically Negotiated Resource Management for Data Intensive Application Suites", IEEE Transactions on Knowledge and Data Engineering, 12(1):78-95, January/February 2000.

[5] C. Lee, J.P. Lehoczky, R. Rajkumar, and J.P. Hansen, "A Scalable Solution to the Multi-Resource QoS Problem", From *Proceedings of the 20th IEEE Real-Time Systems Symposium*, December 1999

[6] J.P. Lehoczky, "Scheduling communication networks carrying real-time traffic", In *Proceedings Real-Time Systems Symposium*, pg. 470-479, December 1998

[7] S. Oikawa and R. Rajkumar, "Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior", From *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 1999.

[8] R. Rajkumar, C. Lee, J. Lehoczky and D. Siewiorek, "Practical Solutions for QoS-based Resource Allocation Problems", From *Proceedings of the IEEE Real-Time Systems Symposium* , December 1998

[9] , R. Rajkumar, K. Juvva, A. Molano and S. Oikawa, "Resource Kernels: A Resource-Centric Approach to Real-Time Systems", From *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998

[10] J.A. Swets and R.M. Picket, "Evaluation of Diagnostic Systems", Academic Press, Inc., 1982

[11] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, "RSVP: A New Resource ReSerVation Protocol", From *IEEE Network*, vol. 7, no. 5, p. 8-18, September 1993

[12] L.R. Welch, B. Shirazi, B. Ravindran, C. Bruggeman, "DeSiDeRaTa: QoS Management Technology for Dynamic, Scalable, Dependable, Real-time Systems", 15th IFAC Workshop - Distributed Computer Control Systems (DCCS '98), Sept. 1998, pp. 7-12.