R-Opus: A Composite Framework for Application Performability and QoS in Shared Resource Pools

Ludmila Cherkasova
Hewlett-Packard Labs
Palo Alto, CA, USA, 94302
Email: Lucy.Cherkasova@hp.com

Jerome Rolia
Hewlett-Packard Labs
Palo Alto, CA, USA, 94302
Email: Jerry.Rolia@hp.com

***Abstract*— We consider shared resource pool management taking into account per-application quality of service (QoS) requirements and server failures. Application QoS requirements are defined by complementary specifications for acceptable and time-limited degraded performance. Furthermore, a requirement specification is provided for both the normal case and for the case where an application server fails in the pool. Independently, the resource pool operator provides a resource access QoS commitment for two classes of service (CoS). These govern statistical multiplexing within the pool. A QoS translation automatically maps application demands onto the resource pool's CoS to best enable sharing. A workload placement service consolidates applications to a small number of servers while satisfying application QoS requirements. The service reports whether a spare server is needed or how applications affected by a single failure can operate according to failure QoS constraints using remaining servers until the failure can be repaired. A case study demonstrates the approach.**

I. Introduction

Many enterprises are beginning to exploit large shared resource pools in data center environments to lower their infrastructure and management costs. These environments may have tens, hundreds, or even thousands of server resources. Capacity management for resource pools decides how many resources are needed to support a given set of application workloads, which applications must be assigned to each resource, and per-application scheduling parameters that ensure appropriate sharing and isolation for the applications. Capacity management is a challenging task for shared environments that currently requires significant manual effort and tends to over-provision resources. This article describes our approach to automate the steps of a capacity self-management system that best matches resource supply with demand.

Resource pools are collections of resources, such as clusters of servers or racks of blade servers, which offer shared access to computing capacity. Virtualization and automation technologies support the lifecycle management (e.g., creation, destruction, migration) of resource containers (e.g., virtual machines, virtual disks [19], [3], [5], [18]). Workload managers for resources [9], [8], [6] provide containers with time-varying access to shares of resource capacity. Application workloads are associated with the containers; the containers are then assigned to resources in the pool. In this paper we assume that each container supports exactly one workload.

Applications can make complex demands on such pools. For example, many enterprise applications operate continuously, have unique time-varying demands, and have performance-oriented Quality of Service (QoS) objectives. Resource pool operators must decide which workloads share specific resources and how workload managers should be configured to support each application. This is a challenge because (i) the capacity of resource pools are generally overbooked (i.e., the sum of per-application peak demands are greater than the capacity of the pool), (ii) because different applications can have different QoS requirements that are affected by the applications' ability to obtain capacity when needed, and (iii) because such pools may incur resource failures – resource pool operators must have a plan to deal with failures and ensure that their service level agreements remain satisfied.

To address these challenges, we propose to replace the standard capacity management process with a framework named R-Opus that supports capacity-as-a-service utilities using resource pools of servers. R-Opus is a composite framework with several features that include:

- independently specified per-application QoS requirements for normal and failure modes;
- resource pool QoS commitments expressed for classes of service (CoS);
- QoS translation that maps application resource demands to resource workload manager allocation priorities that implement resource pool CoS; and
- a workload placement service for normal and failure modes.

Application QoS requirements are defined by complementary specifications for acceptable and time-limited degraded performance and are specified for normal and failure modes. Resource pool QoS commitments quantify the likelihood that a resource container will receive a unit of capacity when required. QoS commitments are expressed for each CoS. Each CoS is associated with a workload manager allocation priority. The workload placement service consolidates applications to a small number of resources while satisfying normal and then failure mode application QoS requirements. The service reports whether a spare server is needed in case of a single node failure.

Section II explains our approach to capacity management for resource pools in more detail. Section III defines application QoS requirements for normal and failure modes. Section IV introduces resource pool QoS commitments. Section V presents methods for automatic QoS translation. The workload placement service is introduced and explained in Section VI. A case study is presented in Section VII. Related work is discussed in Section VIII; summary and concluding remarks are given in Section IX.

II. Capacity Management and Resource Pools

This section provides an introduction to resource pool capacity management. We give an overview of capacity manage-

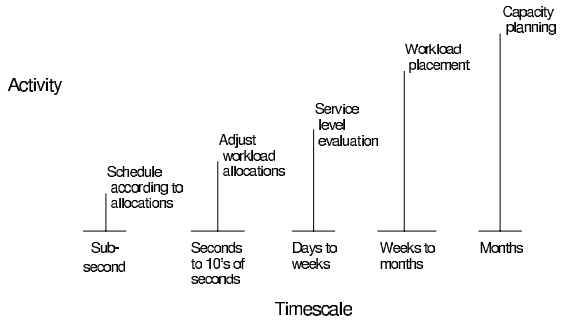

Fig. 1. Capacity Management Activities and Time Scales.

ment activities, their corresponding timescales, and describe resource workload managers and the workload placemen service in more detail. We then describe R-Opus and how it exploits workload managers and the workload placement service to support per-application QoS requirements.

Figure 1 illustrates capacity management activities for resource pools and their different timescales. Long term management corresponds to capacity planning; the goal here is to decide when additional capacity is needed for a pool so that a procurement process can be initiated. Over a medium timescale (e.g., weeks to months), groups of resource containers are chosen that are expected to share resources well. Each group is then assigned to corresponding resources. Assignments may be adjusted periodically as service levels are evaluated or as circumstances change (e.g., new applications must be supported; servers are upgraded, added, or removed). Once resource containers are assigned to a resource, a workload manager for the resource [9], [8] adjusts workload capacity allocations over short timescales based on time-varying workload demand. Finally, resource schedulers operate at the time-slice (sub-second) granularity according to these allocations. Adjustments to allocations in response to changing workloads can greatly increase the efficiency of the resource pool while providing a degree of performance isolation for the containers.

We now describe resource workload managers in more detail. A workload manager monitors its workload demands and dynamically adjusts the allocation of capacity (e.g., CPU) to the workloads, aiming to provide each with access only to the capacity it needs. When a workload demand increases, its allocation increases; similarly, when a workload demand decreases, its allocation decreases. Such managers can control the relationship between demand and allocation using a burst factor; a workload resource allocation is determined periodically by the product of some real value (the burst factor) and its recent demand. For example, if the measured utilization over the previous 5 minutes is 66% of 3 CPUs, then the demand is 2 CPU. A burst factor of 2 would cause an allocation in the next 5 minute interval of 4 CPUs. In this way, a burst factor guides the application towards a utilization of allocation of $\frac{1}{\text{burst factor}}$. In other words, even though the application's allocation may vary over time its utilization of allocation remains somewhat consistent.

The burst factor addresses the issue that allocations are adjusted using utilization measurements. Utilization measurements over any interval are mean values that hide the bursts of demand within the interval. The product of mean demand for an interval and this burst factor estimates the true demand of

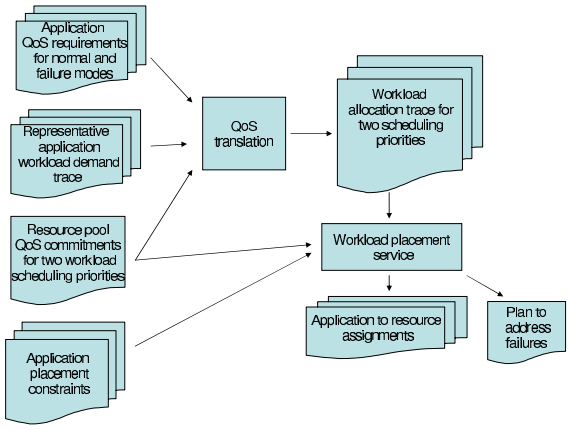

Fig. 2. R-Opus.

the application at short time scales and is used for the purpose of allocation. In general, the greater the workload variation and client population, the greater the potential for bursts in demand, the greater the need for a larger allocation relative to mean demand (i.e., utilization), and hence the greater the need for a larger burst factor.

We assume that the workload manager implements two allocation priorities that correspond to the resource pool CoS. Demands associated with the higher priority are allocated capacity first; they correspond to the higher CoS. Any remaining capacity is then allocated to satisfy lower priority demands; this is the lower CoS. R-Opus uses these CoS along with workload placement to manage application QoS.

The workload placement service employs a trace-based approach to model the sharing of resource capacity for resource pools [12]. Each application workload is characterized using several weeks to several months of demand observations (e.g., with one observation every five minutes) for capacity attributes such as CPU, memory, and disk and network input-output. The general idea behind trace-based methods is that traces capture past demands and that future demands will be roughly similar. We simulate the assignment of multiple application workloads to a resource and estimate the resulting resource access QoS that is provided. Placements are found that satisfy resource access QoS commitments for the historical data. We assume the resource access QoS will be similar in the near future. Though we expect demands to change, for most applications they are likely to change slowly (e.g., over several months). By working with recent data, we can adapt to such a slow change. Significant changes in demand, due for instance to changes in business processes or application functionality, are best forecast by business units; they need to be communicated to the operators of the resource pool so that their impact can be reflected in the corresponding traces.

Figure 2 illustrates the R-Opus approach to capacity management for resource pools as supported by such workload managers and a workload placement service:

- A resource pool operator decides on resource access QoS commitments for two classes of services for resources in the resource pool [12]. These specifications express the likelihood that a unit of capacity will be available when needed.
- For each application workload, the application owner specifies its application's workload QoS requirement as

an acceptable range for the burst factor. The range corresponds to *low* and *high* utilization of allocation targets for the application. Furthermore, the application is permitted time-limited and bounded performance degradation beyond the range, e.g., a service level *degradation*, to account for spikes in demand that the application owner does not want to affect capacity sizing.

- A QoS translation takes the independently specified application QoS requirements and the resource access QoS commitments as input and maps the application's workload demands to allocations for the workload manager's two allocation priorities in a manner that assures the application QoS requirement will be satisfied as long as the resource pool offers the per-CoS QoS it commits to.
- Finally, over the medium term, the workload placement service [12], [15] assigns application workload resource containers to resources in the pool in a manner expected to satisfy the resource access QoS commitments for the two resource CoS in the pool.

The overall approach assumes that the analysis of application behaviour as described in the traces is representative of future behaviour. We rely on historical data to forecast whether certain applications can co-exist on a resource while satisfying QoS requirements [12].

III. Application QoS Requirements

The relationship between acceptable application QoS and system resource usage is complex. We employ an empirical approach that aims to find an acceptable range for the burst factor that relates workload demand to a scheduled allocation for CPU capacity. A stress testing exercise is used to submit a representative workload to the application in a controlled environment [10]. Within the controlled environment, we vary the burst factor that governs the relationship between application demand and allocation. First, we search for the value of the burst factor that gives the responsiveness required by application users (i.e., good but not better than necessary). Next, we search for the value of the burst factor that offers adequate responsiveness (i.e., a worse responsiveness would not be acceptable to the application users). These define an acceptable range of operation for the application on the resource and give a preferred range for the utilization of the allocation for a given application. Similarly, these values can be chosen or refined in operational environments.

More formally, each QoS requirement has a goal and constraints with respect to a range of utilization of allocation:

- U^{low} - defines a utilization of allocation of that supports ideal application performance [1];
- U^{high} - represents a threshold on utilization of allocation beyond which the application performance would be undesirable to users;
- U^{degr} - defines another threshold on utilization of allocation that can be used for coping with infrequent high bursts in demand. Typically, these occasional bursts of demand should not be used for determining the overall application's capacity requirements, since they might lead to significant over provisioning and increased configuration cost.

An application owner can specify application QoS by stating his/her requirement of *acceptable* and *degraded* application performance. The owner specifies application QoS for two modes of operation: *i) normal mode* that means that all planned resources are available; *ii) failure mode* that corresponds to the case of 1-node failure (note that this scenario can be extended to multiple node failures).

- *acceptable* performance: for at least $M\%$ of measurements, utilization of allocation U^{alloc} should be within the desirable range, i.e., $U^{low} \leq U^{alloc} \leq U^{high}$;
- *degraded* application performance: for the remaining measurements $M^{degr} = 100\% - M\%$ the utilization of allocation should not exceed U^{degr}, i.e., $U^{high} < U^{alloc} \leq U^{degr}$. Moreover, T^{degr} specifies the maximum contiguous time [2] when measured utilization of allocation U^{alloc} may exceed U^{high}.

A time-limited degradation T^{degr} value relates the specification to user experience. While application users may tolerate intermittent poor performance, e.g., for 5-10 minutes, but sustained poor performance typically leads to user complaints. We further bound the utilization of allocation for times of non-compliance to $U^{degr} < 1$ to ensure that, in our model, demands are satisfied within their measurement interval.

Consider the following application QoS requirement. Let $U^{low} = 0.5$, $U^{high} = 0.66$, $M^{degr} = 3\%$, $U^{degr} = 0.9$, and $T^{degr} = 30$ minutes. This states that based on the past history of application demands, we need to tailor the resource allocation schema for this application to permit no more than $M^{degr} = 3\%$ of measurements in the trace to have utilization of allocation above $U^{high} = 66\%$. Additionally, these observations must not have value greater than $U^{degr} = 90\%$ and must not exceed $U^{high} = 0.66$ for more than $T^{degr} = 30$ minutes at a time.

Historical demand values are transformed to time-varying allocation requirements for the pool using these utilization of allocation values. The M^{degr} and T^{degr} terms affect the maximum allocation value for the application.

IV. Resource Pool QoS Commitments

The resource access QoS commitments specified by the resource pool operator govern the degree of overbooking in the resource pool. We assume that the first class of service offers guaranteed service. For each resource, the workload placement service ensures that the sum of the per application peak allocations associated with this higher class of service does not exceed the capacity of the resource. The second class of service offers a lower QoS. It is associated with a resource access probability, θ, that expresses the probability that a unit of resource capacity will be available for allocation when needed. The workload placement service finds workload placements such that both constraints are satisfied. Thus it manages overbooking for each resource (i.e., statistical multiplexing).

A formal definition for a resource access probability θ is as follows. Let C be the number of workload traces under

[1] Clearly, the utilization of allocation lower than U^{low} also supports the ideal application performance, however at a price of underutilized (over-allocated) resources. We use $\frac{1}{U^{low}}$ as a burst factor for determining the relationship between the demand and the required ideal allocation.

[2] An additional constraint on the number of degraded epochs per time period, e.g., per day or week, is a useful enhancement. To simplify presentation we do not consider it in this paper.

consideration. Each trace has W weeks of observations with T observations per day as measured every m minutes. Without loss of generality, we use the notion of a *week* as a timescale for service level agreements. Time of day captures the diurnal nature of interactive enterprise workloads (e.g., those used directly by end users). Other time scales and patterns can also be used. Each of the T times of day, e.g., 8:00am to 8:05am, is referred to as a slot. For 5 minute measurement intervals we have $T = 288$ slots per day. We denote each slot using an index $1 \leq t \leq T$. Each day x of the seven days of the week has an observation for each slot t. Each observation has an allocation value for each of the capacity attributes considered in the analysis. Without loss of generality, consider one class of service and one attribute that has a capacity limit of L units of demand. Let $A_{w,x,t}$ be the sum of the allocations upon the attribute by the C workloads for week w, day x and slot t. We define the measured value for θ as follows.

$$\theta = \min_{w=1}^{W} \min_{t=1}^{T} \frac{\sum_{x=1}^{7} \min(A_{w,x,t} \ , \ L)}{\sum_{x=1}^{7} A_{w,x,t}}$$

Thus, θ is reported as the minimum resource access probability received any week for any of the T slots per day. Furthermore, we define a CoS constraint as the combination of a required value for θ and a deadline s such that those demands that are not satisfied are satisfied within the deadline. Let L' be the required capacity for an attribute to support a CoS constraint. A required capacity L' is the smallest capacity value, $L' \leq L$, to offer a probability θ' such that $\theta' \geq \theta$ and those demands that are not satisfied upon request, $A_{w,x,t} - L' > 0$, are satisfied within the deadline. We express the deadline as an integer number of slots s.

V. Partitioning Application Demands Across Two Classes of Service

We now describe our technique for mapping an application's workload demands across two CoS to realize its application QoS objectives. Our method takes as input a characterization of an application's workload demands on the resource, the resource access QoS commitments for resources in the resource pool, and the application-level QoS requirements (expressed using a range for the burst factor that corresponds to U^{low} and U^{high}). As output, it describes how the application's workload demands should be partitioned across the pool's two classes of service (i.e., workload manager allocation priorities).

The proposed method is motivated by portfolio theory [7] which aims to construct a portfolio of investments, each having its own level of risk, to offer maximum expected returns for a given level of risk tolerance for the portfolio as a whole. The analogy is as follows. The resource access QoS commitments quantify expected risks of resource sharing for the two CoS. These CoS correspond to potential investments with the lower CoS having a greater return because the resource pool operator can provide a lower cost service when permitted to increase overbooking. The application demands represent investment amounts. They are partitioned across the CoS so that application QoS remains in the tolerated range, which corresponds to the risk tolerance for the portfolio as a whole. By making greatest use of the lower CoS we offer the resource pool operator the greatest opportunity to share resources and hence lower the cost to the application owner.

We present the approach in three steps:

1) first, we describe how to partition an application's workload demands across two classes of service to support acceptable performance [13];
2) second, we extend the partitioning method for the case where $M^{degr}\%$ of measurements can experience degraded performance;
3) finally, we describe how to partition an application's workload demands across two classes of service to support time-limited performance degradation.

1). First, we describe how to partition an application's workload demands across two classes of service, CoS_1 and CoS_2, to ensure that an application's utilization of allocation U^{alloc} remains within the acceptable performance range:

$$U^{low} \leq U^{alloc} \leq U^{high}$$

CoS_1 offers guaranteed access to capacity. By associating part of the demands with CoS_1, we limit the resource access risk to the demands associated with CoS_2. The resource access probability θ of CoS_2 is chosen by the resource pool operator. Consider three operating scenarios for a resource: (i) it has sufficient capacity to meet its current demands; (ii) demand exceeds supply but the resource is satisfying its resource access constraint; and (iii) demand exceeds supply and the resource is not satisfying its resource access constraint. We consider the first two scenarios here and rely on workload placement techniques to avoid and react to the third scenario [12].

When the system has sufficient capacity, each application workload gets access to all the capacity it needs. In this case, the application's resource needs will all be satisfied and the application's utilization of allocation will be ideal., i.e. less than or equal to U^{low}.

In the case where demands exceed supply, the allocations associated with CoS_1 are all guaranteed to be satisfied. However, the allocations associated with CoS_2 are not guaranteed and will be offered with at worst the operator-specified resource access probability θ. We aim to divide workload demands across these two classes of services while ensuring that the utilization of allocation remains in the acceptable range (U_{low}, U_{high}) defined above to satisfy the application's QoS requirements.

Let p be a fraction of peak demand D_{max} for the CPU attribute for the application workload that is associated with CoS_1. The value $p \times D_{max}$ gives a breakpoint for the application workload such that all demand less than or equal to this value is placed in CoS_1 and the remaining demand is placed in CoS_2.

The range of acceptable allocations must be between $A_{ideal} = D_{max} \times \frac{1}{U^{low}}$ and $A_{ok} = D_{max} \times \frac{1}{U^{high}}$.

So the allocation for the lower but acceptable QoS offered to the application is:

$$A_{ok} = A_{ideal} \times p + A_{ideal} \times (1 - p) \times \theta.$$

Solving this equation for p, we get:

$$p = \frac{\frac{U^{low}}{U^{high}} - \theta}{1 - \theta} \tag{1}$$

where $1 \geq \theta > 0$.

If $\frac{U^{low}}{U^{high}} \leq \theta$ then $p = 0$, i.e., all the demand can be associated with class CoS_2. This provides desirable performance for utilization of allocation in the acceptable range (U_{low}, U_{high}).

Thus, breakpoint p is computed using the three basic parameters: bounds for acceptable utilization of allocation U_{low}, U_{high}, and resource access probability θ for the second class of service CoS_2.

Then, applying breakpoint p to the workload peak demand D_{max}, we compute the maximum portion of demand that should be assigned to CoS_1:

$$D_{CoS_1}^{max} = p \times D_{max}$$

Consider demand D_{cur} from the workload trace. We partition D_{cur} across two classes of services: CoS_1 and CoS_2 as follows:

- if $D_{cur} \leq D_{CoS_1}^{max}$ then it is assigned entirely to CoS_1;
- if $D_{cur} > D_{CoS_1}^{max}$ then demand D_{cur} is split across two classes:
  - a fraction of demand equal to $D_{CoS_1}^{max}$ is satisfied using CoS_1,
  - the remaining part $(D_{cur} - D_{CoS_1}^{max})$ is satisfied using the second class of service CoS_2.

2). Now, let us consider more complex application QoS requirements that have a description of acceptable and degraded performance:

- *acceptable* performance: for at least $M\%$ of measurements, utilization of allocation U^{alloc} should be within the desirable range, i.e., $U^{low} \leq U^{alloc} \leq U^{high}$;
- *degraded* application performance: for the remaining measurements $M^{degr} = 100\% - M\%$ the utilization of allocation should not exceed U^{degr}.

Let $D_{M\%}$ be a demand that corresponds to M-th percentile of the workload demands. For many workloads, $D_{M\%}$ is much smaller than $D_{100\%}$ for $M\% < 100\%$.

The condition for acceptable application performance requires that the maximum allocation for a workload should be at least:

$$A_{ok} = \frac{D_{M\%}}{U^{high}}$$

At the same time, the condition for degraded performance requires that the maximum allocation for a workload should be at least:

$$A_{degr} = \frac{D_{max}}{U^{degr}}$$

If $A_{ok} \geq A_{degr}$ then the allocation provided by acceptable performance requirement (based on M-th percentile of workload demands) is also sufficient for assuring the degraded performance for the remaining $M^{degr}\%$ of the measurements. In this case, demand $D_{M\%}$ is used as a new maximum demand D_{new_max} that controls maximum allocation for given workload:

$$D_{new\_max} = D_{M\%} \tag{2}$$

Therefore, all demands less than or equal to $p \times D_{new\_max}$ are placed in CoS_1 and the remaining demands are placed in CoS_2, where breakpoint p is computed by formula 1.

If $A_{ok} < A_{degr}$ then the allocation A_{ok} provided by acceptable performance requirement (based on M-th percentile of workload demands) is not sufficient for providing degraded performance for the remaining $M^{degr}\%$ of the measurements. Hence, we need to use allocation A_{degr} as the maximum allocation, and compute a new maximum demand D_{new_max} that supports such an allocation in the following way:

$$D_{new\_max} = \frac{A_{degr}}{\frac{1}{U^{high}}} = \frac{D_{max} \times U^{high}}{U^{degr}} \tag{3}$$

Using this formula we can evaluate an upper bound on potential capacity savings one can realize by weakening application QoS requirements and allowing some percentage of points be supported at degraded performance. The potential reduction in capacity, called $MaxCapReduction$, can be computed in the following way:

$$MaxCapReduction = \frac{D_{max} - D_{new\_max}}{D_{max}} \tag{4}$$

Using formula 3 we can replace D_{new_max} in formula 4 above and express the upper bound on $MaxCapReduction$ in the following way:

$$MaxCapReduction \leq \frac{D_{max} - \frac{D_{max} \times U^{high}}{U^{degr}}}{D_{max}} = 1 - \frac{U^{high}}{U^{degr}} \tag{5}$$

Since formula 5 depends only on U^{high} and U^{degr}, we can see that the upper bound for $MaxCapReduction$ is the same for different values of U^{low}, θ, and values used for M-th percentile.

For example, if $U^{high} = 0.66$ and $U^{degr} = 0.9$ then potential $MaxCapReduction = 26.7\%$. This is an upper bound. Whether these maximum capacity reduction can be realized or not depends on the application workload as well as whether $A_{ok} < A_{degr}$.

3). Finally, when a degraded performance has an additional time-limiting constraint that $U^{high} < U^{alloc} \leq U^{degr}$ for no more than T^{degr} contiguous minutes at a time, we perform a special trace analysis to verify this condition within the trace.

Let there be R observations in T^{degr} minutes. Suppose we discover during the trace analysis that there are $R+1$ contiguous observations with utilization of allocation higher than U^{high}, i.e., they have degraded performance. In order to support time-limiting constraint on degraded performance we need to "break" this continuous "degraded performance" sequence by supporting at least one of these demands at acceptable performance range, i.e., for one of those demands we have to increase its allocation so that its utilization is less or equal to U^{high}.

Let D_{min_degr} be the smallest demand among considered $R+1$ contiguous measurements. The current maximum allocation for the overall workload is based on demand D_{new_max} that is computed using formula 2 or 3. Since we need to increase the current maximum allocation for D_{min_degr} we have to recompute a value for D_{new_max} in such a way that a new allocation for D_{min_degr} based on a recomputed D_{new_max} has its utilization of allocation not greater than U^{high}.

First of all, let us compute the allocation that currently is assigned for demand D_{min_degr}. According to our portfolio approach demand D_{min_degr} is split across two classes of services CoS_1 and CoS_2, where

- the fraction of demand assigned to CoS_1 is

$$D_{CoS_1}^{min\_degr} = p \times D_{new\_max} \tag{6}$$

- the fraction of demand assigned to CoS_2 is

$$D_{CoS_2}^{min\_degr} = D_{new\_max} - p \times D_{new\_max} \tag{7}$$

Note, that if $D_{min\_degr} \leq D_{new\_max}$ then $D_{CoS_2}^{min\_degr} = D_{min\_degr} - p \times D_{new\_max}$. However, when $D_{min\_degr} > D_{new\_max}$, demand D_{new_max} is enforcing a limiting cap on the maximum allocation, and $D_{CoS_2}^{min\_degr} = D_{new\_max} - p \times D_{new\_max}$.

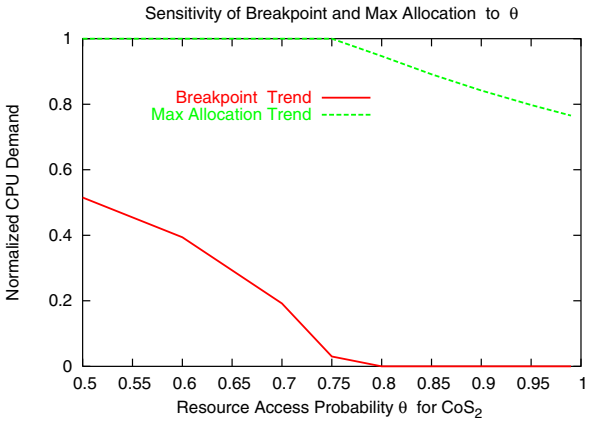

Fig. 3. Sensitivity of breakpoint p and maximum allocation to θ for CoS_2, where $(U_{low}, U_{high}) = (0.5, 0.66)$.

This way, the overall allocation A_{min_degr} provided for demand D_{min_degr} is computed in the following way:

$$A_{min\_degr} = (D_{CoS_1}^{min\_degr} + D_{CoS_2}^{min\_degr} \times \theta) \times \frac{1}{U_{low}} \quad (8)$$

We use A_{min_degr} to recompute a value for D_{new_max}, such that the utilization of a new allocation for D_{min_degr}, based on this recomputed D_{new_max}, is less or equal to U^{high}:

$$U_{high} = \frac{D_{min\_degr}}{A_{min\_degr}} \quad (9)$$

By replacing A_{min_degr} in this equation with formula 8, and by replacing $D_{CoS_1}^{min\_degr}$ and $D_{CoS_2}^{min\_degr}$ using formulas 6 and 7, we have:

$$U_{high} = \frac{D_{min\_degr} \times U_{low}}{p \times D_{new\_max} + (D_{new\_max} - p \times D_{new\_max}) \times \theta}$$

By solving this equation relative to D_{new_max}, we find:

$$D_{new\_max} = \frac{D_{min\_degr} \times U_{low}}{U_{high} \times (p \times (1 - \theta) + \theta)} \quad (10)$$

This trace analysis continues iteratively until we find a new D_{new_max} that satisfies the additional time-limiting constraint on degraded performance for the entire workload.

Note that if $p > 0$ then equation 10 has a very simple outcome (once p us replaced using Formula 1):

$$D_{new\_max} = D_{min\_degr}.$$

If $p = 0$, i.e., all the demand is associated with class of service CoS_2, then equation 10 can be transformed as follows:

$$D_{new\_max} = \frac{D_{min\_degr} \times U_{low}}{U_{high} \times \theta} \quad (11)$$

Suppose the values U_{low} and U_{high} are fixed. The outcome of formula 10 strongly depends on resource access probability θ for CoS_2. Higher values of θ lead to a smaller D_{new_max} because the higher values of θ imply a lower risk for class of service CoS_2. Since D_{new_max} limits and controls the maximum allocation per application, smaller values of D_{new_max} lead to smaller capacity requirements on the resource pool. Thus, we have shown that if there are time-limiting constraints on degraded performance then higher values of θ may result in the smaller maximum allocation per application.

Figure 3 shows impact of θ on breakpoint p (where p defines a fraction of demand that is assigned to CoS_1) and

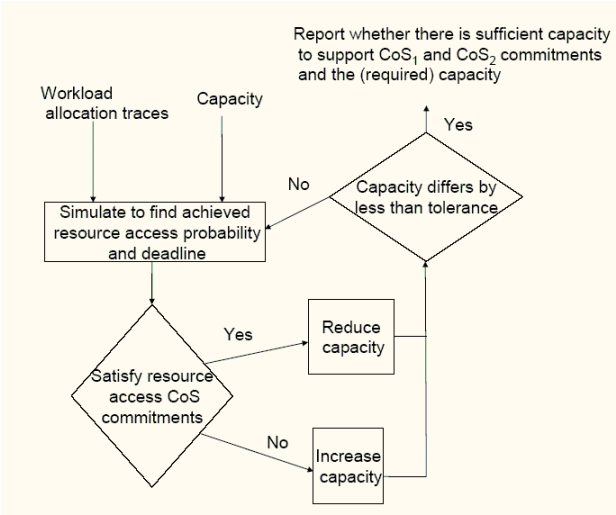

Fig. 4. Simulation Algorithm

maximum allocation per application. We used $(U_{low}, U_{high}) = (0.5, 0.66)$ as an acceptable performance range.

The plot shows the trend for D_{new_max} in a normalized way: the ratio of any two points on the line approximates the ratio in D_{new_max} per application for different values of θ. For example, it shows that for $\theta = 0.95$ the maximum demand D_{new_max} is 20% lower than for $\theta = 0.6$, i.e., the maximum allocation for $\theta = 0.95$ is 20% less than for $\theta = 0.6$.

VI. Workload Placement Service

The workload placement service has two components. A simulator component emulates the assignment of several applications to a single resource. It traverses the traces of allocation requirements to estimate a required capacity that satisfies the resource access QoS commitments. The required capacity can be compared with resource capacity limits. An optimizing search algorithm examines many alternative assignments and reports the best solution found for the consolidation exercise. These components are described in the following sections.

A. Simulator and resource access CoS commitments

Figure 4 illustrates the simulator algorithm. The simulator considers the assignment of a set of workloads to a single resource. It replays the workload allocation traces, compares the aggregate allocations of the observations in the traces with the capacity of the resource, and computes resource access CoS statistics. If the computed values satisfy the CoS commitments then the workloads are said to fit on the resource. A search method, e.g., a binary search, is used to find the required capacity, i.e., smallest value, for each capacity attribute such that the CoS resource pool commitments are satisfied.

When two CoS are involved, the simulation component schedules access to capacity in the following way. Capacity is assigned to CoS_1 first. The remaining capacity is then assigned to CoS_2. This corresponds to the behaviour of the workload managers as described earlier in the paper.

The required capacity of each attribute is found as follows. First a check is made to ensure the sum of the peak application demands associated with CoS_1 do not exceed the capacity of the resource. If they do then the workloads do not fit, otherwise they may fit. If the workloads may fit, then the following process is initiated. If the current capacity satisfies the CoS

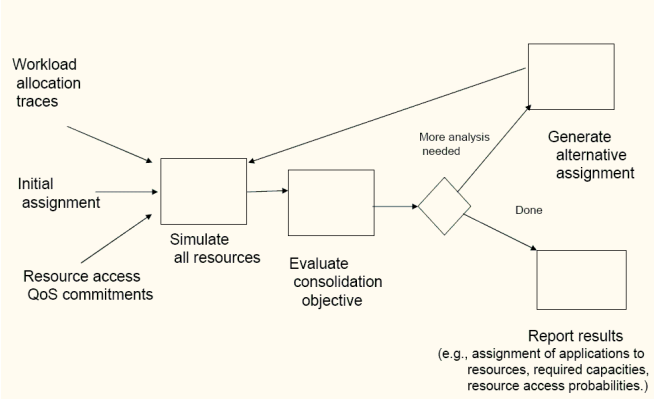

Fig. 5. Optimizing Search

commitments then the algorithm reduces the capacity value for the attribute. If the current capacity does not satisfy the commitments, the algorithm increases the value for capacity up to the limit L of the attribute. The algorithm completes when it finds that the commitments cannot be satisfied or when the value for capacity changes by less than some tolerance. Currently we use a binary search, but other search mechanisms could also be used. Upon termination, the algorithm reports whether the commitments are satisfied (for each attribute). If so, the resulting value for capacity is reported as the required capacity (for each capacity attribute).

B. Optimizing search

Figure 5 illustrates the optimizing search algorithm for the consolidation exercise. The resource access QoS commitments, workload allocation traces, and an initial assignment of workloads to resources are the inputs of the algorithm. The behavior of each resource is simulated using the method described in Section VI-A. The results of the simulations are then used to compute a score for the consolidation objective function. If there is little improvement in the score then the algorithm reports a configuration that achieved the best score while satisfying resource access QoS commitments and terminates. Otherwise a new configuration is enumerated and the simulation process is repeated. A genetic algorithm is used to guide the search.

The consolidation exercise begins with the initial configuration of the system and causes a search for a workload assignment that satisfies commitments and uses a small number of servers. A score is computed for each assignment of the workloads to resources. The score is a sum of values computed for each resource. To simplify the presentation, we assume that each CPU in the pool has the same processing capacity but that resources may have different numbers of CPUs. The values that contribute to the score are:

- 1: for a resource in the pool that isn't used;
- $f(U)$: a function of utilization for a resource with required capacity R less than or equal to the capacity of the resource L, where $U = \frac{R}{L}$ and $0 < U \leq 1$; and,
- $-N$: for resources that are over-booked, i.e., $R > L$, where N is the number of application workloads assigned to the resource.

The function $f(U)$ provides a greater value for higher utilizations than lower utilizations. However, the function scales utilization with respect to the number of CPU resources to reflect that resources with more CPUs can operate at higher utilization levels. Let Z be the number of CPUs per server, we define $f(U)$ as: $f(U) = (U^Z)^2 = U^{2\times Z}$. The square term in the power exaggerates the advantages of higher utilizations (in a least squares sense), the Z term demands that servers with greater numbers of CPUs be higher utilized. The Z term is motivated by the function $\frac{1}{1-U^Z}$ that estimates the mean response time of clients with unit demand in an open queueing network having a single resource with Z CPUs.

The genetic algorithm has mutation and cross-over functions. The mutation function associates a mutation probability with each server that is used according to its value for $f(U)$. The lower the value of $f(U)$ for a resource the greater the likelihood that the resource's application workloads will be migrated to other resources. With each mutation step, the algorithm tends to reduce the number of resources being used by one. The cross-over function mates earlier assignments in a straightforward manner. It simply takes some random number of application assignments from one assignment and the rest from the other to create a new assignment.

C. Planning for Failures

The workload placement service can also be used to report on the impact of single and/or multiple failures. Basically, the configuration of the consolidated system is taken as the initial configuration. This configuration is for a small number of servers as needed to support the applications with their normal mode QoS requirements. For failure modes (e.g., one server at a time), the workload placement service systematically removes one server at a time from the pool, associate its affected applications with their failure mode application QoS requirements, and repeats the consolidation algorithm. The consolidation algorithm reports whether it is possible to place all the affected applications on the remaining servers in the pool with their failure QoS requirements. If this is possible for all failures under study then the service reports that failure modes can be supported without an additional spare server. More detailed information about which applications can be supported in this way and for which failures can be combined with expectations regarding time to repair for servers, the frequency of failures, and penalties to decide on whether it is cost effective to have a spare server or not. However, in our case study in this paper we simply show that the use of an alternative set of application QoS constraints can result in the requirement for one less server.

VII. Case Study

In this section, we present a case study to demonstrate the features of R-Opus for a large enterprise order entry system with 26 applications. The study presents a characterization of the application workloads, results regarding the portfolio approach, and workload placement results. The case study relies on four weeks of workload CPU demand traces with measurement values recorded every 5 minutes.

Figure 6 shows the percentiles of CPU demand for the 26 applications. The demands are normalized as a percentage with respect to their peak values. The 100-percentile of demand corresponds to 100% normalized CPU demand. Several curves are shown that illustrate the 99.9 through 97 percentile of demand. The figure shows that 2 applications, i.e., the leftmost in the figure, have a small percentage of points that are very large with respect to their remaining demands. The leftmost 10 applications have their top 3% of demand values

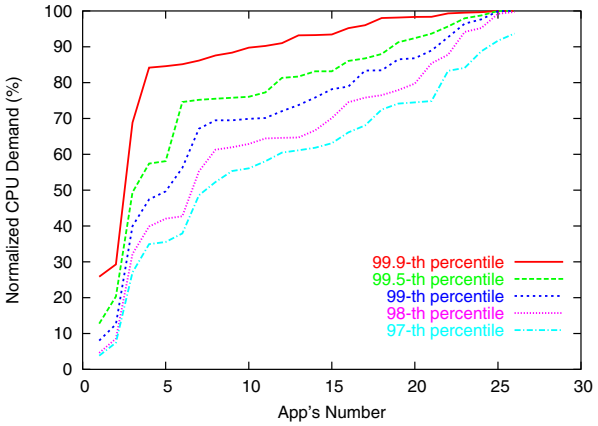

Fig. 6. Top Percentile of CPU Demand for Applications under Study.

between 10 and 2 times higher than the remaining demands in the trace. It shows the bursty nature of demands for most of the applications under study.

In our case study, we consider the following application QoS requirements:

- acceptable application performance: $U^{low} = 0.5$, $U^{high} = 0.66$, with utilization of allocation in the range $(0.5, 0.66)$ for 97% of measurements;
- degraded application performance: for the remaining measurements $M^{degr} = 3\%$ the utilization of allocation should not exceed $U^{degr} = 0.9$. We consider four values for T^{degr}: none, 2 hours, 1 hour, and 30 min, i.e., from no additional time-limiting constraints on degraded performance to the case when degraded performance should not persist longer than 30 min.

The workloads are partitioned across the two CoS to satisfy these application QoS requirements.

Figures 7 a) and b) show the impact of M^{degr} and T^{degr} on maximum allocations for 26 applications under study and two different values for resource access probability: $\theta = 0.95$ and $\theta = 0.6$. The y-axis shows the percent reduction for the maximum allocation with M^{degr}=3% as compared to M^{degr} = 0%. For both values of θ, many of the 26 applications have a 26.7% in reduction for maximum allocation that corresponds to an expected upper bound on $MaxCapReduction$ (as described by formula 5 in Section V). Overall $MaxCapReduction$ is affected more by T^{degr} for $\theta = 0.6$ than for the higher value of $\theta = 0.95$. Again, this is consistent with our general derivations in Section V, where we observe that under time-limiting constraints, higher values of θ lead to a smaller maximum allocation requirements.

Figures 8 a) and b) show the percentage of measurements that have degraded performance, i.e., with utilization of allocation in the range (U^{high}, U^{degr}). While up to 3% of measurements were allowed to be in the degraded range, the additional time-limiting constraint T^{degr}=30 min significantly reduces the number of measurements with degraded performance: it is less than 0.5% for $\theta = 0.95$ and less than 1.5% for $\theta = 0.6$ as shown in Figures 8 a) and b), respectively.

To summarize, for these workloads, a small but controlled relaxation for application QoS requirements can lead to up to an approximately 25% reduction in maximum allocation requirements.

We now consider the use of the workload placement service.

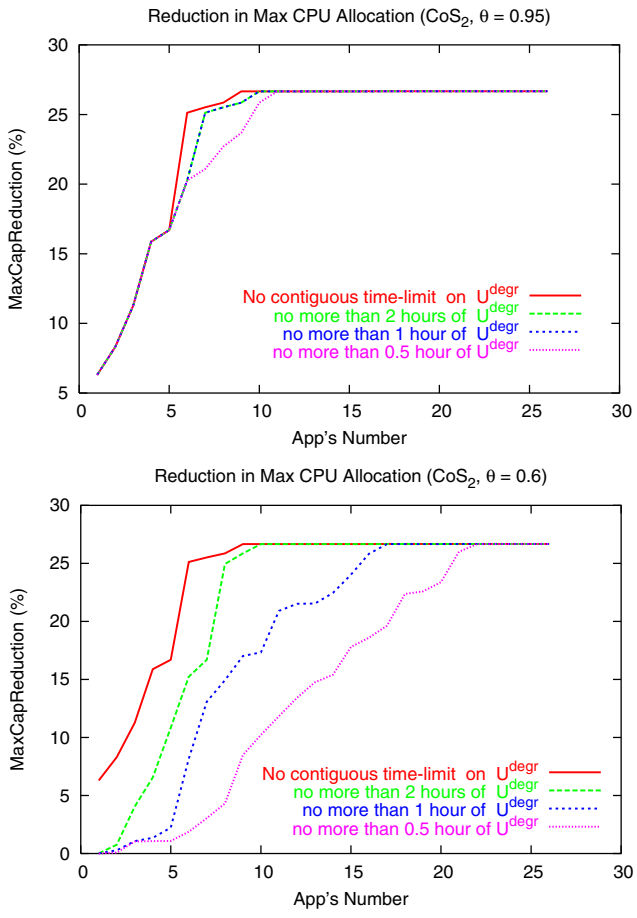

Fig. 7. MaxCapReduction per Application under Different Time-Contiguous Requirement on Degraded Performance.

Table I shows the impact of M^{degr}, T^{degr} and θ [3] on the CPU capacity needed to satisfy the 26 application workloads. The table shows the number of 16-way servers reported as being needed by the workload placement service, the sum of per server required capacity C^{requ}, and the sum of per-application peak CPU allocations C^{peak}. All cases had the same workload placement algorithm termination criteria and used approximately 10 minutes of CPU time on a 3.4 Ghz Pentium CPU. The required capacity values are between 37% to 45% lower than the sum of per-application peak allocations. This shows that resource sharing presents significant opportunity for reducing capacity requirements for these workloads. Furthermore, for cases 1-3 some demands are in both CoS_1 and CoS_2; for cases 4-6 all demands are in CoS_2. If all demands were associated with CoS_1 then, because we would have to limit the sum of per-application peak allocations to the capacity of the resource, we would require at least 15 servers for case 1 and 11 servers for case 3. Thus having multiple classes of service is advantageous.

We now consider the impact of M^{degr} on C^{peak} and then on C^{requ}. With $M^{degr} = 3\%$, we allow 3% of the measurement points to have utilization of allocation between U^{high} and U^{degr}.

For the cases with $T^{degr} = none$, the impact of $M^{degr} = 3\%$ on C^{peak} is identical for both values of θ. There is a

[3]For all the experiments the resource access QoS commitment has a deadline value s that corresponds to 60 min (see Section IV).

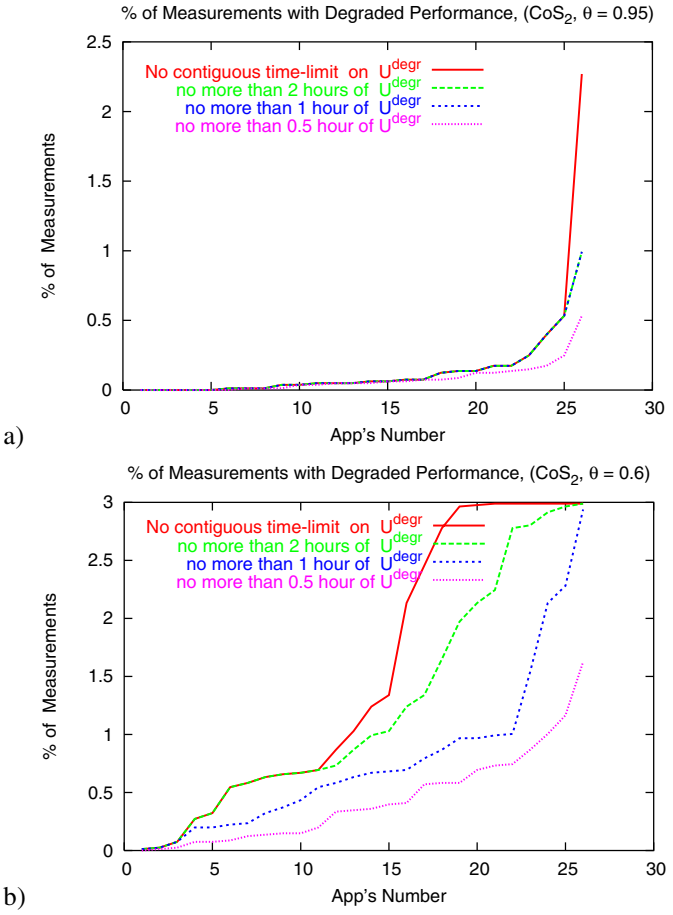

Fig. 8. Percentage of Allocations with Degraded Performance for Different Time-Contiguous Requirements.

Case	M^{degr}	θ	T^{degr}	Num. of 16-way servers	C^{requ} CPU	C^{peak} CPU
1	0	0.60	none	8	123	218
2	3	0.60	30 min	7	106	188
3	3	0.60	none	7	104	166
4	0	0.95	none	8	118	218
5	3	0.95	30 min	7	103	167
6	3	0.95	none	7	104	166

TABLE I
IMPACT OF M^{degr}, T^{degr} AND θ ON RESOURCE SHARING.

reduction in C^{peak} of 24%. For the cases with $T^{degr} = 30$ minutes, for $\theta = 0.6$ there is a reduction in C^{peak} of 14%, whereas for $\theta = 0.95$ there is a reduction of 23%. This is due to the interaction between T^{degr} and θ as discussed above. Having a higher θ value appears advantageous for a resource pool operator.

We now compare the impact of $M^{degr} = 3\%$ and $T^{degr} = 30$ minutes on the values for C^{requ} and C^{peak}. For $\theta = 0.6$ the impact is about the same, and is a reduction of approximately 14% with respect to the $M^{degr} = 0\%$ case. For $\theta = 0.95$, the C^{requ} is reduced by 14% and C^{peak} is reduced by 23% with respect to the $M^{degr} = 0$ case. The workload placement service was not able to realize the same reduction in C^{requ}, most likely because *lowered* per-application peak demands do not necessarily coincide with peaks in aggregate demand.

Finally, we note that cases 1 and 4 from Table I require 8 servers, one more server than the remaining cases. Thus, from the perspective of application QoS constraints for normal and failure modes, we can offer case 1 and 4 constraints as normal mode constraints and the remaining cases as possible constraints for failure mode. In normal mode, the system would use 8 servers. In the case of a single server failure the table shows that the remaining 7 servers could support the system with the other application QoS constraints, i.e, cases 2, 3, 5[4], or 6. However, an appropriate workload migration technology is needed to realize the new configuration without disrupting the application processing.

To summarize, higher values of θ permit more demand to be associated with CoS_2. This gives greater freedom to the workload placement service to overbook capacity. M^{degr} has a bigger impact on C^{peak} than C^{requ} because not all reductions in peak application demands occur at the same times as peak aggregate allocation requirements. Greater values for θ can decrease the maximum required allocations of applications as compared to lower values for θ. Finally, even minor reductions in application QoS requirements can have a big impact on system resource requirements. The appropriate use of QoS requirements can help to support workload placement exercises that deal with resource failures.

VIII. Related Work

Historically, enterprise capacity management groups have relied upon curve fitting and queueing models to anticipate capacity requirements for shared resources such as mainframes or large servers. Curve fitting and business level demand forecasting methods are used to extrapolate measurements of application demands on each resource. Queueing models may be used to relate desired mean response times for model specific workload classes (e.g., batch or interactive, payroll, accounts receivable) to targets for maximum resource utilizations. Unfortunately, such planning exercises are a people intensive and hence expensive process. Most organizations only conduct these exercises when the costs can be justified. Even so, capacity management remains a challenge as today's enterprise data centers can have hundreds of large shared servers and thousands of lightly utilized smaller server resources.

We employ a trace-based approach to model the sharing of resource capacity for resource pools. Many groups have applied trace-based methods for detailed performance evaluation of processor architectures [11]. They can also be used to support capacity management on more coarse data, e.g., resource usage as recorded every five minutes. Our early work on data center efficiency relied on traces of workload demands to predict opportunities for resource sharing in enterprise data centers [2]. We conducted a consolidation analysis that packed existing server workloads onto a smaller number of servers using an Integer Linear Programming based bin-packing method. Unfortunately the bin-packing method is NP-complete for this problem, and as a result is a computationally intensive task. This makes the method impractical as a method for larger consolidation exercises and on-going capacity management.

[4]We note that case 5 requires one fewer CPU than case 6 where we would have expected it to require the same or a larger number of CPUs. We believe this is an anomaly of the genetic algorithm based workload placement algorithm.

As a result heuristic search approaches seem most appropriate for this problem.

Traces have been used to support what-if analysis that consider the assignment of workloads to consolidated servers. AOG [1] and TeamQuest [16] offer products that employ trace-based methods to support consolidation exercises. AutoGlobe [4] proposes a static analysis that also relies on traces for workload placement. To the best of our knowledge these rely on greedy algorithms (or provide a manual interface for a planner) for workload placement. We believe our genetic algorithm approach provides for better solutions, as it has compared favorably to the greedy algorithms we implemented ourselves. Other heuristic search approaches that also take into account correlations in resource demands among workloads may also be worth exploring. Furthermore, our workload placement methods go further than these other methods by addressing issues including resource access Quality of Service (QoS) [14], and, as described in this paper, per-application QoS requirements. This permits a layering of application and resource access QoS objectives to be realized as we have shown in this paper.

Finally, the application QoS objective we considered improves on other QoS objectives we have seen for resource pools. Some researchers propose to limit the capacity requirement of an application workload to a percentile of its demand [17]. This does not take into account the impact of sustained performance degradation on user experience as our M^{degr} and T^{degr} terms do. Others look only at QoS objectives for resources as a whole [4] rather than permitting each application workload to have an independently specified QoS objective as is the case with R-Opus.

IX. Summary and Conclusions

We have introduced R-Opus, a composite framework for realizing application QoS requirements in shared resource pools. The framework brings together several features. It includes a method for dividing application workload demands across two workload manager allocation priorities. We have shown how this can be done to satisfy per-application QoS objectives in shared resource environments. Application owners specify application QoS requirements using a range for acceptable performance along with terms the limit acceptable degradations to this performance. These, along with resource pool resource access QoS, determine how much of each application's demands must be associated with a guaranteed allocation class of service and how much with a second class of service that offers resources with a given probability defined by a resource pool operator. A workload placement service assigns workloads to resources in a manner expected to satisfy the resource access CoS objectives. The more workload that is associated with the second class of service, the greater the opportunity for the resource pool to overbook resources.

Case study results validate our technique. The results show that relatively small diminishment in application QoS requirements can lead to a significant reduction in per-application maximum allocation, e.g., 25% in our case study. Higher θ values from resource pool operators can lead to greater reductions; in particular when the time-limited degradation is employed. Having a non-guaranteed CoS greatly reduces aggregate capacity requirements when consolidating workloads to servers. The workload placement service was able to realize significant benefits from consolidation, e.g., up to 45% with respect to the sum of peak aggregate application allocation requirements, for these workloads.

Finally, the approach we present aims to ensure that applications have utilization of allocation values that they need. This is necessary to provide application quality of service. However, other system features may also affect responsiveness but are not modeled by our approach. These include the impact of caching, database locks, garbage collection, and software bugs. Furthermore, performance tuning and capacity management exercises often aim ensure that sufficient memory and input-output capacity are available to make CPU resources the bottleneck to manage. Future work will look at extending our techniques to consider the impact of greater sharing of other capacity attributes such as memory and input-output resources. We believe R-Opus should be part of a larger management system that takes these aspects of system behaviour into account as well.

References

[1] www.aogtech.com

[2] A. Andrzejak, J. Rolia and M. Arlitt, Bounding Resource Savings of Utility Computing Models, HP Labs Technical Report HPL-2002-339.

[3] G. Banga, P. Druschel, J. Mogul. Resource containers: a new facility for resource management in server systems, in Proc. of the 3rd Symposium on Operating System Design and Implementation (OSDI '99), New Orleans, LA, 1999.

[4] Daniel Gmach, Stefan Seltzsam, Martin Wimmer, and Alfons Kemper AutoGlobe: Automatische Administration von dienstbasierten Datenbankanwendungen GI Conference on Database Systems for Business, Technology, and Web (BTW), Karlsruhe, Germany, February 2005.

[5] B. Dragovic, K. Fraser, S. Hand, et al. Xen and the Art of Virtualization, in Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP 2003), Bolton Landing, NY, October 2003.

[6] K. Duda and D. Cheriton. Borrowed-virtual-time (BVT) scheduling: Supporting latency-sensitive threads in a general purpose scheduler. In Proc. of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999), Kiawah Island Resort, SC, December 1999.

[7] E. J. Elton and M. J. Gruber, Modern Portfolio Theory and Investment Analysis, Wiley, 1995.

[8] IBM Enterprise Workload Manager. http://www.ibm.com/developerworks/autonomic/ewlm/

[9] HP-UX Workload Manager. http://www.hp.com/products1/unix/operating/wlm/

[10] D. Krishnamurthy, Synthetic Workload Generation for Stress Testing Session-Based Systems. Ph.D. Thesis, Carleton University, Jan. 2004.

[11] J. J. Pieper, A. Mellan, J. M. Paul, D. E. Thomas and F. Karim, High level cache simulation for heterogeneous multiprocessors, Proceedings of the 41st annual conference on Design automation, San Diego, USA, ACM Press, pages 287-292, 2004.

[12] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak. A Capacity Management Service for Resource Pools, in Proc. of the 5th International Workshop on Software and Performance (WOSP 2005), Palma, Spain, July 2005, pp. 229-237.

[13] J. Rolia, L. Cherkasova, M. Arlitt, and V. Machiraju. An Automated Approach for Supporting Application QoS in Shared Resource Pools. In Proc. of the 1st International Workshop on Self-Managed Systems & Services (SelfMan 2005), Nice, France, May 2005.

[14] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, environments. Performance Evaluation Journal 58, pages 319-339, 2004.

[15] S. Singhal, S. Graupner, A. Sahai et al. Resource Utility System. In Proc. of the 9th International Symposium on Integrated Network Management (IM 2005), Nice, France, May 2005.

[16] www.teamQuest.com

[17] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In USENIX OSDI, December 2002

[18] A. Whitaker, M. Shaw, and S.Gribble. Scale and Performance in the Denali Isolation Kernel. In Proc. of the 5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, MA, December 2002.

[19] VMware VirtualCenter 1.2. http://www.vmware.com/products/vmanage/vc_features.html