An Automated Approach for Supporting Application QoS in Shared Resource Pools

Jerry Rolia, Ludmila Cherkasova, Martin Arlitt, Vijay Machiraju
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94303, USA
{jerry.rolia,lucy.cherkasova,martin.arlitt,vijay.machiraju}@hp.com

Abstract

Many enterprises are beginning to exploit shared resource pool environments. In these environments, the application workloads exploit a common set of hardware resources. These are complex environments where selecting per-application scheduler parameter settings is a challenging task. It 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, and ii) because different applications can have different quality of service (QoS) requirements that are affected by the applications' ability to obtain capacity when needed. This paper presents a new method that automates the computation of appropriate values for scheduler parameters. The method takes as input a characterization of an application's workload demands, an application QoS requirement, and a measure of resource access QoS for resources that governs overbooking within the pool. As output, the method automatically specifies resource scheduler parameter settings that are expected to realize the application's requirements. We assume that the pool's resource schedulers can support at least two service priorities and can also manage the relationship between resource demand and allocation. A case study demonstrates the method and illustrates the per-application risks associated with resource sharing.

1 Introduction

Resource pools are collections of resources, such as clusters of servers or racks of blades, that offer shared access to computing capacity. Virtualization services offer interfaces that support the lifecycle management (e.g., create, destroy, move, size capacity) of resource containers (e.g., virtual machines, virtual disks [1, 2, 3, 4]) that provide access to shares of capacity. Application workloads are assigned to the containers which are then associated with resources. Resources in the pool have capacity attributes, e.g., CPU, memory, I/O operation rates, and bandwidths, each with limited capacity.

We assume that each resource in the pool has a scheduler that monitors its workloads' demands and dynamically varies the allocation of capacity, e.g., CPU, to the workloads aiming to provide each with access only to the capacity it needs. As a workload's demand increases its allocation increases, as a workload's demand decreases its allocation decreases. Such allocators typically can control the relationship between demand and allocation using a *burst factor* n, e.g., such that a workload's allocation will be approximately some real value $n \geq 1$ times its recent demand. The factor addresses the issue that allocations are adjusted using periodic utilization measurements. Utilization measurements over any interval are mean values that hide the bursts of demand within the interval. 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 need for a larger factor. The product of mean demand and this burst factor estimates the true demand of the application at short time scales and is used for the purpose of allocation. Furthermore, we assume that the scheduler can implement at least two priorities with all the demands associated with the highest priority satisfied first. Any remaining capacity is then used to satisfy the demands of the next priority.

We consider the problem of hosting enterprise applications in such resource pools. There are existing tools, e.g. HP Workload Manager[5], IBM Enterprise Workload Manager[6], that aim to support the resource management tasks in such resource pools. However, the process of setting and adjusting the parameters in these tools for managing required capacities is still a manual process. The hosted applications often operate continuously, have unique time varying demands and performance oriented quality of service (QoS) objectives. These complexities make it difficult and time consuming for human operators to make effective use of such pools. The applications can share individual resources such as a CPU or may require many resources. We take the following approach to this problem.

- The resource pool operator decides on the resource access QoS objectives for two classes of service for resources in the resource pool [7]. The first class of service is associated with the highest scheduling priority. The second is associated with the lower priority.

- Each application owner specifies its application workload's QoS requirement. This is specified as a range for the burst factor. The range specifies values for burst factor that correspond to ideal and simply adequate application QoS.

- The method we propose maps each application's workload demands onto the two classes of service. This mapping determines scheduling parameters for the workload.

- The Quartermaster [8] capacity manager [7] assigns application workloads to resources in the pool in a manner expected to satisfy the resource access QoS objectives for the pool.

The resource access QoS objectives specified by the resource pool operator govern the degree of overbooking in the resource pool. We assume that the first CoS offers guaranteed service. It corresponds to a scheduler's high priority service. The capacity manager ensures that the sum of the demands associated with this first CoS does not exceed the capacity of the resource. The second CoS offers a lower QoS. It manages overbooking, i.e., statistical multiplexing, for each resource.

This paper presents a new method for specifying how to partition an application's workload demands across the two classes of service to realize application level performance oriented QoS objectives. The method is motivated by portfolio theory [9]. A case study demonstrates the method and characterizes the per-application risk associated with sharing.

Section 2 describes the method. Case study results demonstrate the technique in Section 3. Summary remarks are offered in Section 4.

2 Portfolio Approach

This section describes the technique for mapping an application's workload demands across two classes of service to realize its application QoS objectives. The technique takes as input a characterization of an application's workload demands on the resource, the resource access QoS objectives for resources in the resource pool, and the application level QoS requirements as expressed using a range for the burst factor. As output it describes how the application's workload demands should be partitioned across the pool's two CoS. We now describe each of these inputs and output in more detail.

2.1 Trace-based characterization of workload demand

We employ a trace-based approach to model the sharing of resource capacity for resource pools [7]. Each application workload is characterized using several weeks to several months of demand observations, e.g., with one observation every 5 minutes. The general idea behind trace-based methods is that traces capture past demands and that future demands will be similar. Though we expect demands to change, for most applications they are likely to change slowly, e.g., over several months. By working with recent history we adapt to such change. Significant changes in demands, due to changes in business processes or application functionality are best forecast by business units and communicated to the operators of the resource pool so that their impact, e.g., scale demands up or down, can be reflected in the trace.

2.2 Resource access QoS

The Quartermaster capacity manager is a service that is used by the resource pool operator to assign workloads to specific resources and quickly assess the impact of future demands on the pool. It has an optimizing search method that supports consolidation (e.g., tight packing) and load levelling (e.g., load balancing) exercises. The service uses the traces of demands to assign workloads to specific resources such that when traces are replayed, demands associated with the first CoS are guaranteed and demands associated with the second CoS are offered with an operator specified resource access probability θ.

A formal definition for a resource access probability θ is as follows. Let A be the number of workload traces under 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 a measured 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 $D_{w,x,t}$ be the sum of the demands upon the attribute by the A workloads for week w, day x and slot t. We define the measured value for θ as follows.

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

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, $D_{w,x,t} - L' > 0$, are satisfied within the deadline. We express the deadline as an integer number of slots s.

2.3 Application QoS

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 the CPU capacity attribute. Though access to CPU capacity is not the only issue that can affect application quality of service [10], it is often responsible [10] and limits a workload's access to many other capacity attributes. A stress testing exercise is used to submit a representative workload to the application in a controlled environment[11]. Within the controlled environment we vary the burst factor that governs the relationship between application demand and allocation. We search for the value of burst factor $n_{ideal} \geq 1$ that gives the responsiveness required by application users (i.e., very good but not better than necessary), and the value n_{ok}: $1 \leq n_{ok} \leq n_{ideal}$ that offers adequate responsiveness (i.e., worse responsiveness would not be acceptable to the application users). These define an acceptable range of operation for the application on the resource.

These values for n bound lower and upper values for the utilization of an allocation:

$$U_{low} = \frac{1}{n_{ideal}}$$

and

$$U_{high} = \frac{1}{n_{ok}}.$$

Thus, the utilization of the allocation must remain in the range (U_{low}, U_{high}), where U_{high} is acceptable but not ideal.

2.4 Portfolio Approach

We aim to partition an application's workload demands across two classes of service, namely CoS 1 and CoS 2, to ensure that the application's burst factor remains within its acceptable range. CoS 1 offers guaranteed access to capacity. By associating part of the demands with CoS 1 we limit the resource access risk to be the demands associated with CoS 2. CoS 2 has a resource access probability of θ and a deadline s as 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. Planning exercises aim to avoid the third scenario [7].

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 U_{ideal}. In the case where demands exceed supply, the demands associated with CoS 1 are all guaranteed to be satisfied. However, the demands associated with CoS 2 are not guaranteed and will be offered with a 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 range (U_{ideal}, U_{ok}) to satisfy the application's QoS requirements.

Let p be a fraction of peak demand D for the CPU attribute for the application workload that is associated with CoS 1. The value $p\ D$ 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. This breakpoint value is the scheduling parameter that we automatically compute.

We solve for p such that in the second scenario the application workload's burst factor is no worse than n_{ok}. The range of allocations must be between $A_{ideal} = D \times n_{ideal}$ and $A_{ok} = D \times n_{ok}$. 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:

$$p = \frac{\frac{n_{ok}}{n_{ideal}} - \theta}{1 - \theta},$$

where $1 \geq \theta > 0$.

3 Case study

Our case study presents some general results regarding the portfolio approach and the implications of the results on 26 application workloads from a large enterprise order entry system [7].

Figure 1 presents our general results. Figure 1 (a) shows the general relationship between resource access probability θ for CoS 2, the burst factor range that describes an application's QoS requirement (expressed as a range of utilization of allocation), and the fraction of an application's peak demand that gets associated with CoS 2. Four curves are shown. These correspond to a utilization of allocation range of $(0.5, 0.6)$ which is a high QoS through to $(0.5, 0.9)$ which is a comparatively low QoS. The figure shows that even a low resource access probability of $\theta = 0.55$ permits between 30% and 100% of application demands to be associated with CoS 2. Figure 1 b) presents similar results in a manner that allows the Quartermaster capacity manager to automatically map between an application's acceptable utilization range, the operator selected value for θ, and the percentage value p needed find the breakpoint used to divide an application's workload across the two CoS.

Figure 2 illustrates the impact of this approach on the 26 applications of the large enterprise order entry system. In this scenario the application utilization of allocation is in the range $(0.5, 0.6)$. The figure shows the peak number of CPUs needed by each application, and for $\theta = 0.8$ and $\theta = 0.7$, how many CPUs must be provisioned using

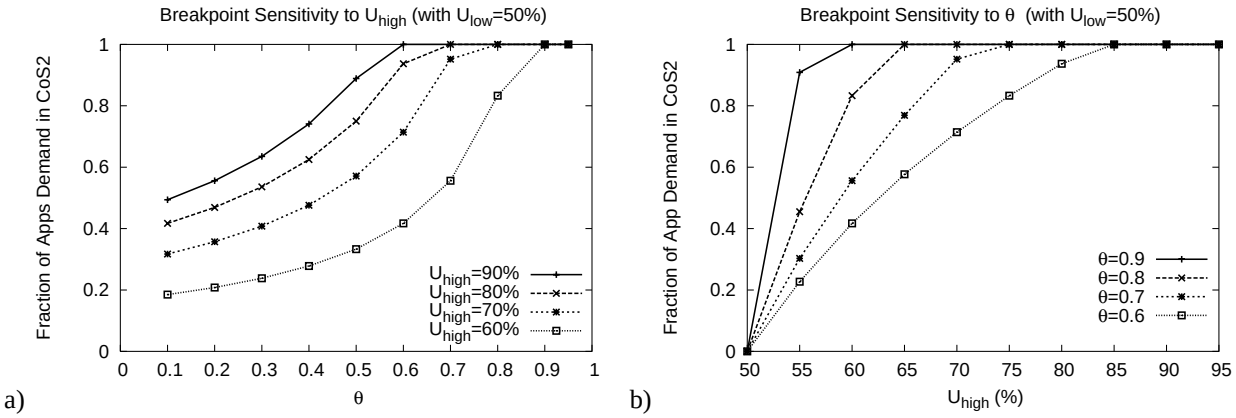

Figure 1: Sensitivity: Resource Access Probability, Range for Utilization of Allocation, and Percentage of Demand for CoS 2.

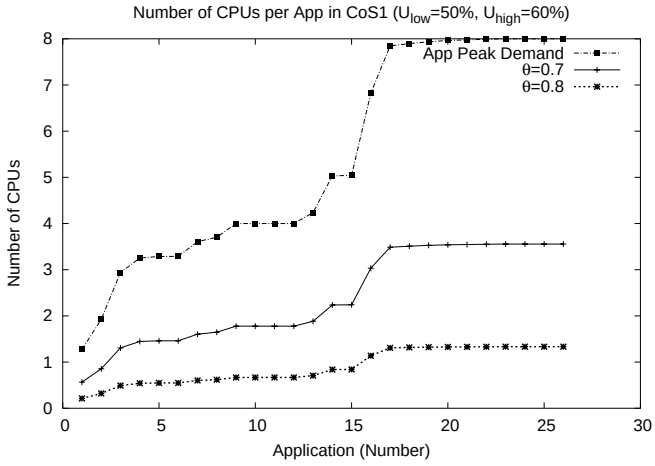

Figure 2: Application Workload Usage of CoS 1.

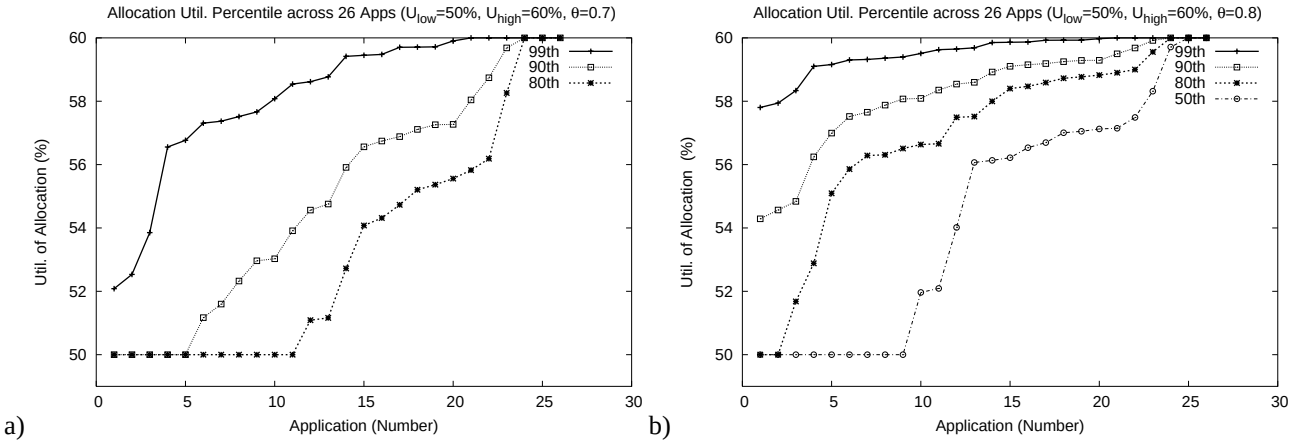

Figure 3: Distribution of Application Utilization of Allocation.

the guaranteed CoS 1 (as determined by the corresponding breakpoint). As expected, a higher value for θ means a lower breakpoint so that less demand is associated with CoS 1 and more with CoS 2. The higher value for θ increases the use of the shared portion of each resource which may increase the utilization of resources in the pool. However, there are diminishing returns. A value of $\theta = 0.9$ puts virtually all application workload demands in CoS 2.

Figure 3 provides insight into how the selected breakpoint affects each of the applications. By choosing a breakpoint based on an application's peak CPU demand we are provisioning for the scenario where the application requires its peak demand. Figure 3 (a) and (b) illustrate the distribution of utilization of allocation for each application for the $\theta = 0.7$ and $\theta = 0.8$ scenarios, respectively. Figure 3 (a) shows that 11 of 26 applications spend 80% of their time at 50% utilization of their allocation, i.e., U_{low}. For much of the time the application's demands are less than the breakpoint so all of its demands are satisfied by CoS 1. These 11 applications spend 99% of their time below the 59% utilization of allocation. Figure 3 (b) has $\theta = 0.8$, which introduces greater opportunity for sharing. In this scenario only 2 of 26 applications spend 80% of their time at 50% utilization of their allocation. It is interesting to note that increasing the resource access probability for CoS 2 puts more application demands in CoS 2 and therefore puts the application at greater risk of operating closer to the higher end of its utilization of allocation, U_{high}.

Finally, recall that an application will only operate near the high end of its allocation when it is both at risk of doing so, as illustrated in Figure 3, and aggregate demand from the many application workloads assigned to the same resource exceeds the capacity of the resource. From our pessimistic definition of θ, and our use of the capacity manager to assign workloads in an appropriate manner, we therefore expect the application to operate near U_{high} infrequently.

4 Summary

This paper presented a method motivated by portfolio theory for selecting application workload specific scheduling parameters for resource pool environments. The approach lets an application owner specify application QoS requirements using a range for a burst factor for the CPU demand attribute. This range along with resource pool resource access QoS determine how much of the application's demands must be associated with a guaranteed CoS and how much with a second CoS that offers resources with probability θ. The more workload that is associated with the second CoS the greater the opportunity for the resource pool to overbook resources. Case study results demonstrate the technique and illustrates the risks of sharing for applications in a large enterprise order entry system. Future work includes completing the characterization of application risks of sharing based on aggregate application demands on a resource and using this information to further manage the resource pool.

References

[1] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. Proc. of ACM SOSP, October 2003.

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

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

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

[5] HP-UX Workload Manager.
`http:www.software.hp.com/portal/swdepot/displayProductInfo.do?productNumber=T1302AA`

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

[7] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak, A Capacity Management Service for Resource Pools, HP Labs Technical Report, HPL-2005-01, 2005.

[8] S. Singhal, S. Graupner, A. Sahai, V. Machiraju, J. Pruyne, X. Zhu, J. Rolia, M. Arlitt, C. Santos, D. Beyer, and J. Ward, Quartermaster: A Resource Utility System, HP Labs Technical Report, HPL-2004-152. To appear in the proceedings of IM 2005.

[9] E. J. Elton and M. J. Gruber, Modern Portfolio Theory and Investment Analysis, John Wiley & Sons, 1995.

[10] I. Cohen, M. Goldszmidt, T.P. Kelly, J. Symons, and J. Chase, Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control, 6th Symposium on Operating Systems Design and Implementation (OSDI '04), December, 2004.

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