SUPPORTING APPLICATION QUALITY OF SERVICE IN SHARED RESOURCE POOLS

By JERRY ROLIA, LUDMILA CHERKASOVA, MARTIN ARLITT, *and* VIJAY MACHIRAJU

Dividing an application's workload demands to better manage resource capacity.

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 to 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 overprovision resources. Here, we describe our approach to automate the steps of a capacity self-management system that best matches resource supply with demand.

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.

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 life-cycle management (creation, relocation, termination) of resource containers (virtual machines or virtual disks [1, 2, 11, 12]). Workload managers for resources [3, 5, 6] provide containers with access to shares of resource capacity. Application workloads are associated with the containers; the containers are then assigned to resources in the pool.

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. Objectives express per-application requirements for responsiveness. 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 the capacity of resource pools are generally overbooked (the sum of per-application peak demands are greater than the capacity of the pool), and because different applications can have different QoS requirements that are affected by the applications' ability to obtain capacity when needed.

To address these challenging issues, we propose to replace the standard capacity management process with a self-managing system that governs access to capacity for resource pools. This article describes the system with a focus on a method for ensuring application QoS objectives. The method exploits workload manager allocation priorities to achieve an application's QoS objectives. Allocations are time-varying shares of resource capacity that become dedicated to each application. When demand exceeds supply, higher-priority allocation requests are dedicated capacity first. The method takes as input a characterization of the application's workload demands, its QoS requirement, and a measure of resource access QoS for resources that governs overbooking (statistical multiplexing) within the pool. As output, the method automatically specifies how to divide an application's workload demands across two workload manager allocation priorities in a manner expected to realize the application's QoS requirement.

Capacity Management Activities

Figure 1 illustrates capacity management activities for resource pools at 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 (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. Capacity management tools can be used to automate such a process. For example, our capacity management tool [8] takes into account detailed workload interactions and the overbooking of resources via statistical multiplexing to automatically decide which containers should share resources. Once resource containers are assigned to a resource, a workload manager for the resource [5, 6] 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.

Workload Managers

We assume that each resource in the pool has a workload manager. The manager monitors its workload demands and dynamically adjusts the allocation of capacity (for example, the 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. 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 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 (utilization), and hence the greater the need for a larger burst factor.

Further, let us assume that the workload manager implements two allocation priorities. Demands associated with the higher priority are allocated capacity first; they correspond to the higher class of service. Any remaining capacity is then allocated to satisfy lower-priority demands; this is the lower class of service.

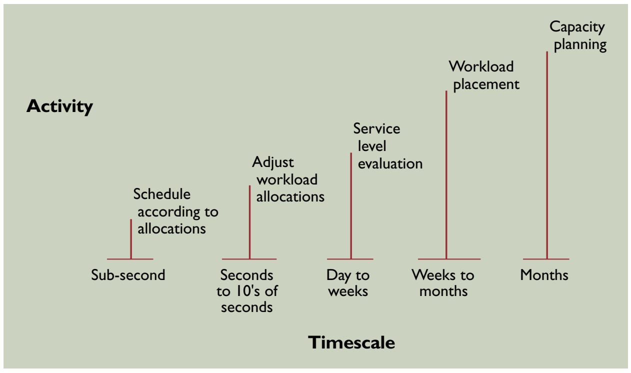

Figure 1. Capacity management activities and time scales.

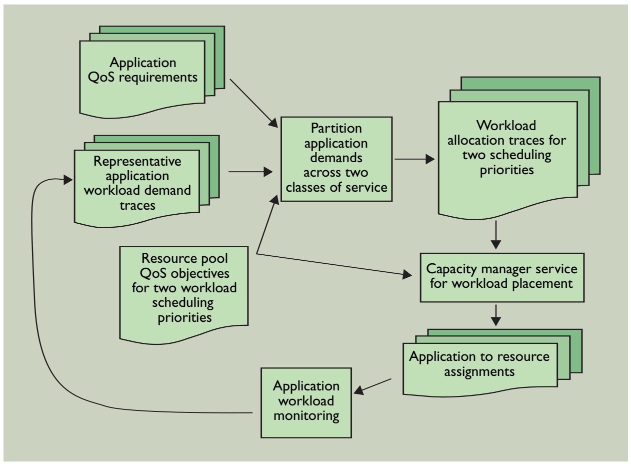
Application QoS requirements
Representative application workload demand traces
Resource pool QoS objectives for two workload scheduling priorities
Partition application demands across two classes of service
Workload allocation traces for two scheduling priorities
Capacity manager service for workload placement
Application to resource assignments
Application workload monitoring

Figure 2. Application QoS-aware capacity management process for resource pools.

Application QoS-Aware Capacity Management

Our process for supporting application QoS in resource pools is shown in Figure 2. A resource pool operator decides on resource access QoS objectives for two classes of service for resources in the resource pool [8]; these are described further here. For each application workload, the application owner specifies its application's workload QoS requirement as a range for the burst factor. The range spans from ideal to simply adequate application QoS. This range and the resource access QoS objectives are used to map the application's workload demands onto the two classes of service. Finally, over the medium term, the capacity manager [8, 10] assigns application workload resource containers to resources in the pool in a manner expected to satisfy the resource access QoS objectives for the pool. Application workload monitoring maintains up-to-date views on application resource usage as feedback for this self-managing approach.

The resource access QoS objectives specified by the resource pool operator govern the degree of overbooking in the resource pool. We assume the first class of service offers guaranteed service. For each resource, the capacity manager ensures the sum of per-application peak allocation requirements 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 resources will be available for allocation when needed. The capacity manager finds workload placements such that both constraints are satisfied. Thus it manages overbooking for each resource (statistical multiplexing). Deciding on resource access QoS objectives is a long-term capacity planning task that takes into account the economics of providing resource pool capacity as a service and the resource access risk that application owners are willing to incur.

Partitioning an Application's Demands Across Two Classes of Service

We now describe our technique for mapping an application's workload demands across two classes of service (CoS) to realize its application QoS objectives. The proposed method is motivated by portfolio theory [4], 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.

Our method 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 (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 classes of service.

Trace-based Characterization of Workload Demand. We utilized a trace-based approach to model the sharing of resource capacity for resource pools [8]. Each application workload is characterized using several weeks to several months of demand observations (for example, with one observation every five minutes). The general idea behind trace-based methods is that traces capture past demands and that future demands will be approximately similar. Though we expect demands to change, for most applications they are likely to change slowly (such as 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, sales for e-commerce systems, or modified application functionality, are best forecast by business units; they must be communicated to the operators of the resource pool so their impact can be reflected in the corresponding traces. New applications, those without historical traces, need estimates for capacity. They may be placed in overprovisioned sand-box environments and observed until their workloads and demands are reflected in demand traces. We have found the trace-based techniques to be sufficiently accurate for ongoing capacity management in an enterprise environment [8].

Resource Access QoS Constraints and Application QoS. The resource access probability for a capacity attribute is defined in [9]. For each class of service of the resource pool, an operator specifies a threshold for the resource access probability. Application workloads that use a given CoS can thus assume they will receive resources with a given probability. Furthermore, we define a QoS constraint as the combination of a threshold value for the resource access probability and a deadline such that those demands that are not satisfied immediately are satisfied within the deadline.

Supporting application QoS by managing resource provisioning requires an understanding of how application QoS requirements relate to resource usage. The relationship is complex because it requires detailed knowledge of numerous application requests and transactions that is rarely known to people involved in capacity management. Furthermore, system measurements are typically collected at a coarse timescale, such as five minutes. These hide bursts in application activity that happen within measurement intervals. We employ empirical approaches to discover the relationship and express the relationship as a range for burst factors that relate demands to allocations.

We suggest two empirical approaches. As a first approach, a stress-testing exercise may be used to submit a representative workload to the application in a controlled environment [7]. Within the controlled environment, we vary the burst factor that governs the relationship between application demand and allocation. We then search for the value of the burst factor that gives the responsiveness required by application users (that is, good but not better than necessary). Next, we search for the value of a second burst factor that offers adequate responsiveness (worse responsiveness would not be acceptable to the application users). These define an acceptable range of operation for the application on the resource. The utilization of the allocation for a given workload must remain within this range. An alternative approach is to adjust the burst factors in an operational environment to find those values that support required and adequate responsiveness.

Portfolio Approach. We aim to partition an application's workload demands across two classes of service, CoS_1 and CoS_2, to ensure 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 the demands associated with CoS_2. The resource access probability of CoS_2 is chosen by the resource pool operator. Consider

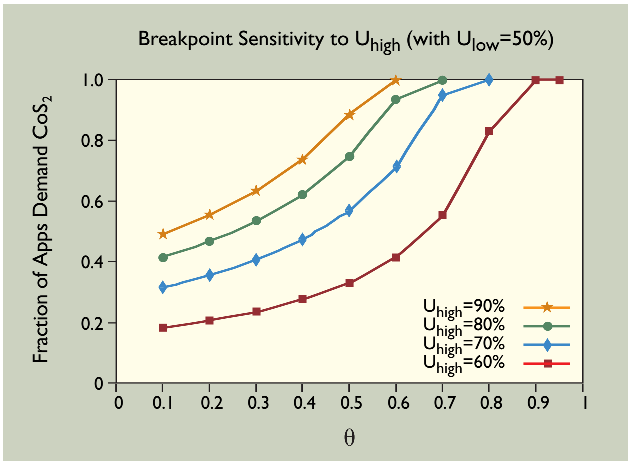

Figure 3a. Sensitivity: Resource access probability, range for utilization of allocation, and percentage of demand for CoS_2.

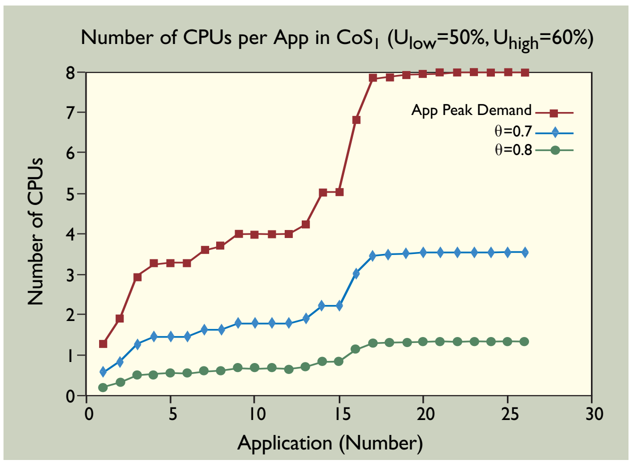

Figure 3b. Application workload usage of CoS_1.

three operating scenarios for a resource: it has sufficient capacity to meet its current demands; demand exceeds supply but the resource is satisfying its resource access constraint; and demand exceeds supply and the resource is not satisfying its resource access constraint. We consider the first two scenarios here; workload placement techniques can be used to avoid and react to the third scenario [8].

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. 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. We aim to divide workload demands across these two classes of services while ensuring the utilization of allocation remains in the acceptable range defined previously 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 product of p and 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. We solve for p such that in the second scenario the burst factor offered to the application is bounded by the value deemed to give adequate application QoS [9].

Case Study

Next, we present some results regarding the portfolio approach and the implications of these results on 26 application workloads from a large enterprise order entry system [8]. Figure 3a shows the relationship between resource access probability, denoted as Θ, for CoS_2 and the fraction of an application's peak demand that is associated with CoS_2. Four curves are shown. Each corresponds to a particular utilization of allocation range with a lower bound of 50% and upper bounds of 60% through 90%, respectively. The range [0.5, 0.6] corresponds to the highest application QoS, whereas [0.5, 0.9] corresponds to the lowest application QoS. The figure shows that even a low resource access probability of Θ=0.6 permits between 40% and 100% of application demands to be associated with CoS_2 for the highest and lowest application QoS scenarios, respectively, thereby increasing opportunities for sharing.

Figure 3b illustrates the impact of this approach on the number of CoS_1 CPUs needed by the 26 applications for an application utilization of allocation range of [0.5, 0.6]. The figure has three curves: the top curve shows the peak number of CPUs needed by each application; the bottom two curves show the number of CPUs needed for the scenarios with the resource pool resource access probability of Θ=0.7 and Θ=0.8, respectively. 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. For greater values of Θ, the use of the shared portion of each resource increases, which may increase the utilization of resources in the pool. From more detailed results [9], we found that a value of Θ=0.9 puts virtually all application workload demands in CoS_2.

Finally, we expect that through the automation of capacity management practices, planned application demands will rarely exceed the capacity of a resource. Most often a resource pool will provide a resource access probability that is greater than the value spec-

THIS APPROACH CAN SUPPORT THE CONFIGURATION OF A SELF-MANAGING SYSTEM FOR MANAGING THE CAPACITY OF RESOURCE POOLS.

ified by the resource pool operator. As a result, most applications will operate toward their ideal application QoS, that is, lower utilization of allocation, much of the time.

Conclusion

We have presented a method for dividing an application's workload demands across two workload manager allocation priorities. This can be used to satisfy application QoS objectives in shared resource environments. Application owners specify application QoS requirements using a range for a workload manager burst factor for the CPU demand attribute. This range, along with resource pool resource access QoS, determines how much of the 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. The more workload that is associated with the second class of service, the greater the opportunity for the resource pool to overbook resources.

Experimental results validate our technique. This approach can support the configuration of a self-managing system for managing the capacity of resource pools. In the future, we plan to complete the characterization of application risks of sharing, based on aggregate application demands on a resource, and to use this information to further improve the management of the resource pool. **C**

References

1. Banga, G., Druschel, P., and Mogul, J. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd Symposium on Operating System Design and Implementation (OSDI '99)*, New Orleans, LA, 1999.
2. Dragovic, B., Fraser, K., Hand, S. et al. Xen and the art of virtualization. In *Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, (Bolton Landing, NY, Oct. 2003).
3. Duda, K. and Cheriton, D. Borrowed-virtual-time (BVT) scheduling: Supporting latency-sensitive threads in a general purpose scheduler. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999)* (Kiawah Island Resort, SC, Dec. 1999).
4. Elton, E.J. and Gruber, M.J. *Modern Portfolio Theory and Investment Analysis*. Wiley, 1995.
5. HP-UX Workload Manager; www.hp.com/products1/unix/operating/wlm/.
6. IBM Enterprise Workload Manager; www.ibm.com/developerworks/autonomic/ewlm/.
7. Krishnamurthy, D. Synthetic workload generation for stress testing session-based systems. Ph.D. Thesis, Carleton University, January 2004.
8. Rolia, J., Cherkasova, L., Arlitt, M. and Andrzejak, A. A capacity management service for resource pools. In *Proceedings of the 5th International Workshop on Software and Performance (WOSP 2005)* (Palma, Spain, July 2005), 229–237.
9. Rolia, J., Cherkasova, L., Arlitt, M. and Machiraju, V. An automated approach for supporting application QoS in shared resource pools. In *Proceedings of the 1st International Workshop on Self-Managed Systems and Services (SelfMan 2005)* (Nice, France, May 2005).
10. Singhal, S., Graupner, S., Sahai, A. et al. A resource utility system. In *Proceedings of the 9th International Symposium on Integrated Network Management (IM 2005)* (Nice, France, May 2005).
11. VMware VirtualCenter 1.2; www.vmware.com/products/vmanage/vc_features.html.
12. Whitaker, A., Shaw, M. and Gribble, S. Scale and performance in the Denali isolation kernel. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI 2002)*, (Boston, MA, Dec. 2002).

Jerry Rolia (jerry.rolia@hp.com) is a senior scientist in the Internet Systems and Storage Laboratory at Hewlett-Packard Laboratories in Palo Alto, CA.

Ludmila Cherkasova (lucy.cherkasova@hp.com) is a senior scientist in the Internet Systems and Storage Laboratory at Hewlett-Packard Laboratories in Palo Alto, CA.

Martin Arlitt (martin.arlitt@hp.com) is a senior scientist in the Internet Systems and Storage Laboratory at Hewlett-Packard Laboratories in Palo Alto, CA.

Vijay Machiraju (vijay.machiraju@hp.com) is a senior scientist and project manager in the Internet Systems and Storage Laboratory at Hewlett-Packard Laboratories in Palo Alto, CA.