RESOURCE ACCESS MANAGEMENT FOR A UTILITY HOSTING ENTERPRISE APPLICATIONS

J. Rolia, X. Zhu, and M. Arlitt
Hewlett-Packard Laboratories
Palo-Alto, CA, USA, 94304
jerry.rolia@hp.com, xiaoyun.zhu@hp.com, martin.arlitt@hp.com

Abstract:

In this paper we introduce a Resource Access Management (RAM) framework for resource utilities that facilitates Class of Service (CoS) based automated resource management. The framework may be used to offer resources on demand to enterprise applications that have time varying resource needs. The classes of service include guaranteed, predictable best effort, and best effort. The analytical apparatus we exploit requires the notion of application demand profiles that specify each application's resource requirements. These profiles may be statistical in nature. Consequently a policing mechanism is introduced to constrain each application's resource usage within its profile. A case study that exploits data from 48 data center servers, is used to demonstrate the framework. We show that our techniques are effective in: exploiting statistical multiplexing while providing service level assurances, limiting application demands in the presence of hostile application behaviour, and providing for differentiated service levels as planned.

Keywords: Resource management, Grid computing, Utility computing, Enterprise applications

1. Introduction

Enterprise applications, such as enterprise resource management, customer relationship management, and store fronts, can benefit from utility computing in the same way they currently benefit from storage and server consolidation exercises. Consolidation helps to decrease costs of ownership and increase return on investment. Resource utilities aim to provide such benefits on a large scale by automating management processes and increasing the effectiveness of consolidation. In this paper, we introduce a resource access management (RAM) framework for resource utilities that support enterprise applications.

Utility and Grid computing offer an appropriate starting point for automating consolidation processes for enterprise environments. Our approach relies upon: *utility computing*, where complex infrastructure is provided to enterprise applications on demand [2] [4] [10] [14] [19]; and *Grid computing*, which offers open services that help to match resource demands with supply, and that implements protocols for reserving, acquiring, and releasing resources adaptively [6] [7].

With this view, we assume that infrastructure service providers, which include data centers that may be internal to medium and large enterprises, will offer information technology resources as a utility service. Customers with applications, for example a department within an enterprise, will anticipate or characterize the demand of their applications and then discover, via Grid services, which utility is best able to satisfy the demands. The application will then be deployed and acquire and release resources as needed.

This paper presents a resource access management (RAM) framework for resource utilities that support Grids for enterprise applications. The framework helps to increase the effectiveness of consolidation processes by balancing utility asset utilization with Quality of Service (QoS) for resource access by applications. Differentiated QoS allows an infrastructure provider to be more flexible regarding the business models and services they offer to their customers. The framework provides Class of Service (CoS) based admission control and resource allocation for applications and implements a policing mechanism that governs access to resources. The CoS include guaranteed, predictable best effort, and best effort. For predictable best effort, the utility provides access to resources with a statistical assurance level θ. A case study demonstrates features of the framework.

The remainder of the paper is organized as follows. Section 2 describes related work. Our RAM framework is introduced in Section 3. A case study illustrates the behaviour of the framework for a hypothetical data center in Section 4. Summary and concluding remarks are given in Section 5.

2. Related Work

Grid resource management systems, such as LSF [22], Condor [11], and Legion [12], provide appropriate scheduling support for batch style computing jobs. However, they do not address the needs of enterprise applications. Enterprise applications operate continuously, have the potential for large peak-to-mean ratios in resource demands, and may have variable numbers of users. Resource management systems that support enterprise applications can increase asset utilization by exploiting the notion of statistical multiplexing. However, service level assurances are essential; enterprise applications must have confidence they will have access to resources when needed.

MUSE [3] is an example of a utility that treats hosted Web sites as services. All services run concurrently on all servers in a cluster. A pricing/optimization model is used to determine the fraction of cpu resources allocated to each service on each server. The over-subscription of resources is dealt with via the pricing/optimization model. When resources are scarce, costs increase thereby limiting application demand. Commercial implementations of such goal driven technology are emerging [5][18].

Garg *et al.* describe an SLA framework for QoS provisioning and dynamic capacity allocation [8]. They describe a mechanism that links application QoS, SLAs, and pricing. It includes the notion of penalties for utilities that do not live up to their obligations and incentives for applications to release resources when they are not needed. We also assume that applications have an incentive to relinquish resources that are not needed.

Our demand management approach differs from the above in the following way. Instead of relying on a dynamic pricing model we use historical and/or anticipated load information to specify a statistical demand profile (SDP) for each application [16]. An SDP bounds an application's expected resource requirements with a time ordered sequence of probability mass functions (pmfs), with one pmf per resource type. This enables support for statistical multiplexing and corresponding statistical assurances regarding resource availability. Furthermore, we separate the resource demand specifications of an application from mechanisms that control that demand. Each application is solely responsible for delivering an appropriate quality of service to its customers. It must translate a quality of service to a quantity of resources required to achieve that level of service [21]. The utility is responsible for providing resources on demand with a particular level of assurance (i.e. a probability) to its applications. We believe this separation of concerns is practical for many kinds of enterprise applications.

Urgaonkar *et al.* also consider quality of service issues [20] regarding resource access, but they do not characterize statistical assurance for a utility. Also they do not take into account the time varying nature of demands.

Hellerstein *et al.* illustrate the use of ANOVA [9] models and second order auto-regressive models [17] to characterize an application's request behaviour. Auto-correlation analysis is used to identify cycles of behaviour, such as weekly cycles versus daily cycles. ANOVA models give an additive workload model for daily, weekly, and monthly cycles. The second order auto-regressive models provide for a detailed characterization of residual behaviour once such cycles are removed. Their approach is to anticipate metric threshold violations over short time scales so that some preventative action can be taken before the violation occurs. The policing approach we present in this paper differs. It specifies directly when a utility may throttle an application's demand. It is a credit-based system, similar to a leaky-bucket approach [13], but based on time varying historical application behaviour.

In [16] we model and explore the impact of correlations among application demands on estimates for the number of resources needed for resource pools and the accuracy of our statistical assurance. In the next section we build upon [16] to introduce a RAM framework, classes of service, and a policing mechanism for a resource utility that supports enterprise applications.

3. Resource Access Management

This section describes our framework for Resource Access Management (RAM). The framework includes three components: admission control, policing, and CoS. *Admission control* decides whether an application will be permitted to execute within a utility. *Policing* mechanisms govern application requests to acquire and release resources as they execute within the utility. The CoS provide differentiated service to the admitted applications.

The RAM framework relies on SDPs. An SDP describes the expected resource usage of an application over time. Together, the SDPs of many applications include information required by a utility to estimate the number of resources required to satisfy their aggregate requirements while taking into ac-

count statistical multiplexing. SDPs are used to implement admission control and are reviewed in Section 3.1. Policing and CoS are introduced in Sections 3.2 and 3.3, respectively. Policing relies on the notion of application entitlement profiles (EP). These are used by the utility to decide whether an application's per-slot requests for resources are consistent with its SDP. If a request is not consistent then the utility can reject the resource request. The RAM framework as a whole is described in Section 3.4.

3.1 Statistical Demand Profiles (SDPs)

SDPs [16] represent historical and/or anticipated resource requirements for applications. For each unique resource type used by an application, we model the corresponding quantity of resources required as a sequence of random variables, $\{\mathbf{X}_t,\ t = 1, ..., T\}$. Here each t indicates a particular time slot, and T is the total number of slots used in this profile. For example, if each t corresponds to a 60-minute time slot, and $T = 24$, then this profile represents resource requirements by hour of day.

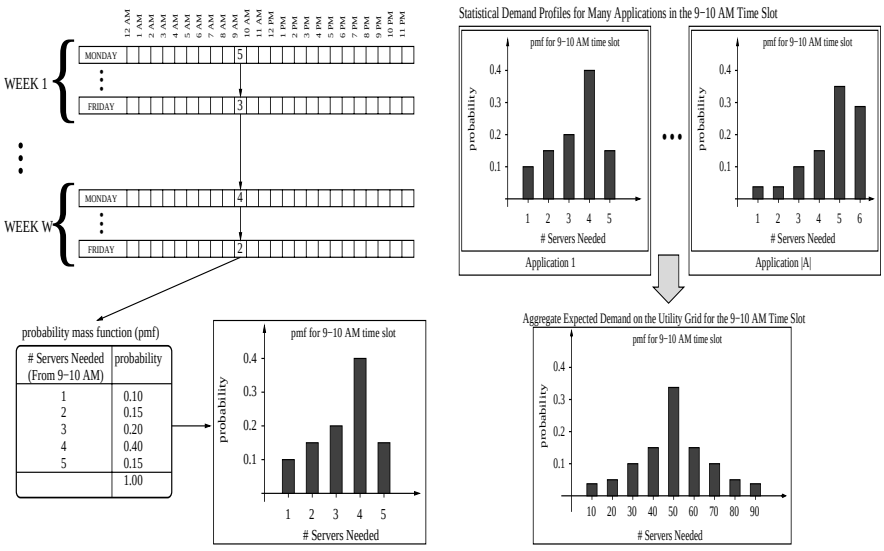
Our assumption here is that, for each fixed t, the behaviour of $\mathbf{X}_t$ is predictable statistically given a sufficiently large number of *observations* from historical data. This means we can use statistical inference to predict how frequently a particular quantity of resources may be needed. For each slot we use a probability mass function (pmf) to represent this information.

Figure 1(a) shows the construction of a pmf for a given time slot (9-10 am) for an SDP of an application. In the example only weekdays, not weekends, are considered. The application required between 1 and 5 servers over W weeks of observation. Since there are 5 observations per week, there are a total of 5 W observations contributing to each application pmf. Figure 1(b) illustrates how the pmfs of many applications contribute to a pmf for the utility as a whole for a specific time slot. As with applications, the SDP for the utility has one sequence of pmfs per resource type. The aggregate demand on a shared pool for a particular resource type is modeled as a sequence of random variables, denoted as $\{\mathbf{Y}_t,\ t = 1, \ldots, T\}$.

A complete description of SDPs, confidence intervals for probabilities in pmfs, correlations between application demands, and a mechanism for estimating the size of a resource pool needed to offer a particular level of service is offered in [16].

3.2 Policing and Entitlement Profiles (EPs)

SDPs provide a mechanism for characterizing expected resource requirements for each time slot t. They are appropriate for sizing the resource pools needed by the utility. However to provide the expected levels of statistical assurance, applications must behave according to their SDPs. Though a pmf of an SDP limits the maximum number of resources an application is entitled to for its corresponding time slot, we still require a method to ensure that each application adheres to its pmfs over time. The purpose of our *policing* mechanism is to specify an application's entitlement to resources over both short and longer time scales, to recognize when an application exceeds its entitlement,



(a) pmf of an application SDP (b) pmf of a utility SDP

Figure 1. Creating statistical demand profiles

and to enable the framework to deal with surplus requests for resources in a systematic manner.

For short time scales, a *horizontal profile* characterizes the percentage of resource usage H_t^w that may occur over a sliding window of w slots ending at time slot t, for one or more values of w for all time slots. For example, H_t^w for time slot $t = 9$am to 10am and $w = 3$, describes the interval between 7am and 10am on the same day. If the maximum possible demand for the interval, based on the application's demand profile, is Δ units, then the horizontal entitlement H_t^w is a percentage of Δ, for example 80%.

For longer time scales, a *vertical profile* characterizes the percentage of resource usage V_t^w that may occur over a sliding window of w instances of the same time slot t, for one or more values of w, for all slots. For example, an application may be entitled to a total of Δ resource units over O instances of a 9am to 10am time slot. The vertical profile of V_t^w may specify that no more than 50% of the Δ units may be consumed in $w = O/3$ successive instances of a slot and no more than 100% of Δ units may be consumed in O successive instances of the slot. In this way the vertical component captures medium to long-term entitlements.

We define the H_t^w and V_t^w for an SDP's time slots for multiple values of w as an *entitlement profile* (EP). We use the EP to implement our *policing mechanism* that governs per-interval requests for resources. If an application requests more resources than it is entitled over any of the values for w then RAM can treat the requests according to some policy. For example it may reject the surplus requests without contributing to the utility's service level violations or treat them as best effort.

As with the construction of SDPs, the EP should be based on automated observation of the application's demand behaviour. They should be derived while constructing SDPs.

Lastly, it could be the case that an application acquires resources but holds them longer than expected thereby exceeding its next time slot's H_t^w or V_t^w for some w. In this case the application may be required to release some resources. We refer to this as *clawback*.

3.3 Classes of Service

This section describes classes of service for applications. Without loss of generality, in this paper we assume that an application acquires resources according to one CoS, and that its access to resources is constrained by its EP. In general, an application is expected to partition its requests across multiple classes of service to achieve the level of assurance it needs while minimizing its own costs.

We define the following CoS: *Guaranteed*, the application receives a 100 percent assurance it will receive the resources specified by its SDP; *Predictable Best Effort* (PBE) with probability θ, the utility offers resources to applications of this CoS with probability θ as defined in Section 3.1; and *Best Effort*, an application has access to these resources when they are available but must release these resources to the utility on demand.

For the PBE CoS, per-slot access to resources is governed by application EPs. For the Guaranteed CoS, an application's SDP need only contain the peak requirement for each time slot. It is always entitled to its per-slot peak requirement.

Consider a set of applications that exploit the utility. Using the techniques of Section 3.1, for the same applications, the utility will require a larger resource pool for a guaranteed CoS than for a predictable best effort CoS. Similarly, larger values of θ will require larger resource pools. In this way CoS has a direct impact on the cost of resources offered by the utility.

3.4 Resource Access Management Framework

This section describes the overall resource management framework. Figure 2 illustrates the admission control and policing process from the perspective of applications.

Figure 2(a) shows admission control and resource access steps for an application that aims to be hosted by the utility. The application presents its SDP and EP with a specific CoS. The utility performs an admission control test using the SDP to determine whether it has sufficient resources to accept the application while satisfying its obligations to existing applications. If accepted, the application may acquire resources, become deployed and begin its execution. Once in execution the application acquires and releases resources as needed but in accordance with its EP. As time progresses, each application's actual resource usage is characterized. For each time slot, for each sliding window, we compute each application's history of actual resource usage C. Actual resource usage is computed in the same manner as H_t^w and V_t^w but with recent measurements. C and its EP are presented for a policing test by the utility. An application is

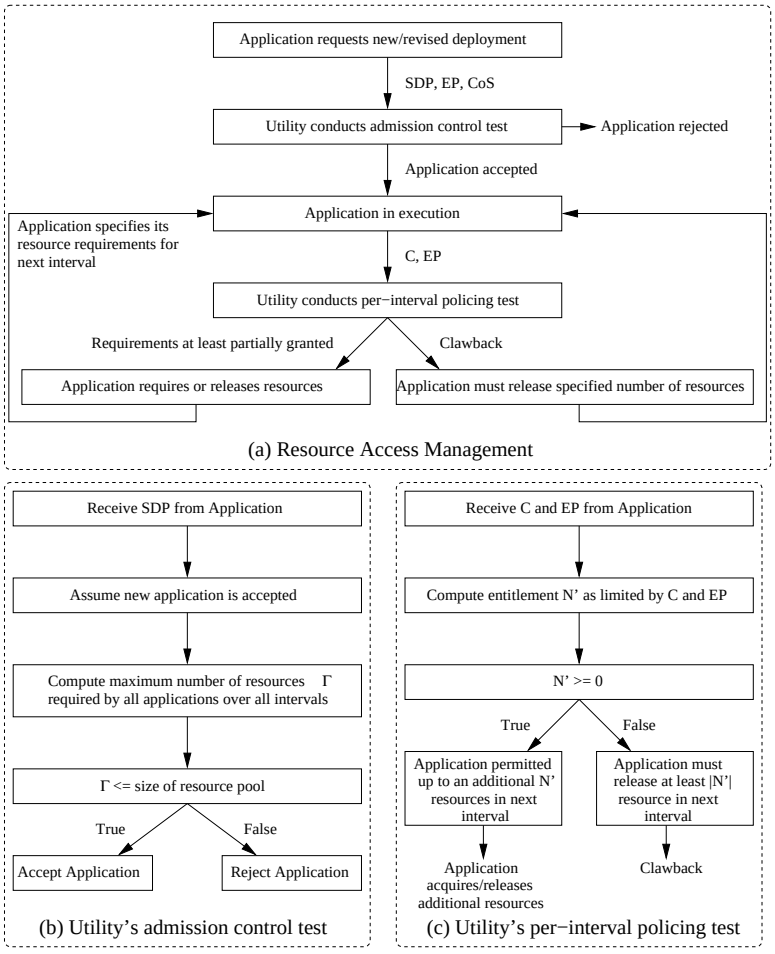


Figure 2. Resource access management processes

always permitted to release resources. The policing test decides the number of additional resources an application is entitled to or how many it must return (clawback). Even when permitted to request additional resources, depending on the application's CoS it may not actually receive all of the resources it requests. Receipt depends on resource availability, CoS based arbitration, and utility policy.

Figure 2(b) illustrates the admission control test. To begin, we assume that the application will be accepted. We unfold the SDP of the application onto the time slots of the utility's calendar, and then determine the total number of resources, Γ, needed to support all applications of all CoS over the future time slots. [16] explains how the pmfs of SDPs can be used to bound the potential impact of correlation in application demands on Γ when admitting a new ap-

plication to a utility. Γ is then compared with the actual size of the resource pool to determine whether it is possible to accept the additional application. If so, then the changes to the utility's calendar are made permanent; otherwise, the request for admission is rejected and the application does not receive any resources. For the PBE CoS we use the techniques of [16], based on the SDPs of Section 3.1, to compute the number of resources required by each PBE CoS resource pool separately. For the Guaranteed CoS we need a pool large enough to satisfy peak application demands on an interval by interval basis. The number of resources needed by the utility, Γ, is chosen as the sum of the number of resources needed by each of its CoS separately.

Figure 2(c) illustrates the policing test. Given an application's history of resource usage C and its EP we compute the maximum number of additional resources N' to which the application is entitled. If N' is less than 0, the application must return $|N'|$ resources to the utility (clawback).

4. Case study

This section presents a case study that demonstrates how the RAM framework may be applied in practice to: quantify application demand requirements, explore the impact of policing, observe the impact of service differentiation, and illustrate resource availability. For this case study, we consider cpu utilization information from 48 servers in an enterprise data center. Sections 4.1 through 4.5 describe our hypothetical resource utility and demand characterization, the classes of service chosen for our study, our experimental design, and results.

4.1 Hypothetical Resource Utility

For the purpose of our study we were able to obtain cpu utilization information for a collection of 48 servers. The servers have between 2 and 8 cpus each, with the majority having either 4 or 6 cpus. The data was collected between September 2, 2001 and October 24, 2001. For each server, the average cpu utilization across all processors in the server was reported for each five minute measurement interval. The information was collected using MeasureWare (Openview Performance Agent) [1].

We *interpret* the load of each server as an application for a resource utility for enterprise applications. Whereas the groups' servers have multiple cpus in our study we assume the utility has many servers with one cpu each. If a server only required one cpu in an interval then its corresponding application requires one server in the utility. We exploit the fact that changes in server utilization reflect real changes in required activity for its applications. Our applications have the same changing behaviour. However since our purpose is simply to validate our techniques we are not concerned with whether it is feasible for the loads on the multi-cpu servers to be hosted on many single cpu servers. Similarly we do not scale the loads with respect to processor speed and/or memory capacity. Further details and results for our study are available in [15].

4.2 Statistical Demand Profiles and Entitlement Profiles

For this case study we chose to characterize the application SDPs by weekday and by hour of day. We consider all weekdays to be equivalent, and we omit data from weekends. As a result, our profile consists of 24 one hour time slots. Since we have utilization data for 35 days, there are $O = 35$ data points that contribute to each pmf in each application's SDP. We used the maximum of the utilizations observed at 5 minute intervals as the utilization value for a corresponding hour.

Values for H_t^w are based on observation of the 100-percentile of total servers used over a window size of $w = 4$, which contains four 60 minute time slots. The values for H_t^w are typically in the 70-90% range. For example, for a sliding window of $w = 4$ slots, the sum of the peak values for the corresponding pmfs may be δ whereas the observed maximum number of resources used for the window may have been only 0.7δ. Values for V_t^w are 100% of the total demand Δ as observed over the $w = 35$ observations used to characterize the SDP.

4.3 Classes of Service

For our case study we consider the following classes of service: guaranteed, predictable best effort with $\theta = 0.999$, denoted as PBE(0.999); and predictable best effort with $\theta = 0.99$, denoted as PBE(0.99).

We use a simulator that generates streams of resource requests according to the pmfs of application SDPs [16]. We submit applications to the utility for the three classes. Each class is an instance of the 48 applications. All applications are accepted. Their SDPs, along with their classes of service, determine the total number of resources for the simulated resource pool. We keep track of unused resources and assume these are offered as part of a best effort service to batch style jobs or other applications that can tolerate resource clawback. For each experiment, we simulate 1000 days of resource access.

4.4 Experimental design

For our experiments, we examine how effectively our policing mechanisms curtail bad behaviour. Policing determines which entitlement mechanisms are used. *Bad behaviour* is defined as the case where certain applications demand more servers than which they are entitled. Finally, in our experiments we exploit pool sharing. Each PBE class is given access to unused surplus servers of the other PBE class. The number of *surplus servers* for a CoS for a time slot t is defined as the difference between the total size of the server pool for that CoS (over all slots) and the number of servers needed to offer the assurance level θ for slot t. PBE(0.999) applications are given higher priority of access than PBE(0.99).

For bad behaviour, we consider the cases where all applications of the PBE(0.99) CoS run at the peak of their pmfs between time slots 10 and 18 (9am to 6pm). These are the slots with the greatest aggregate demand. Applications start their bad behaviour at random within the first 35 simulated days. Though

this is not the worst possible behaviour, we believe it represents significant hostile behaviour with respect to applications in a resource utility.

For policing, we chose entitlement profiles as described in Section 4.2. In this study, when servers are clawed back an application must return them immediately.

In each case unused servers from the guaranteed and predictable best effort classes of service are made available to applications via a best effort CoS. Note that as a result the number of servers available for best effort service is time varying.

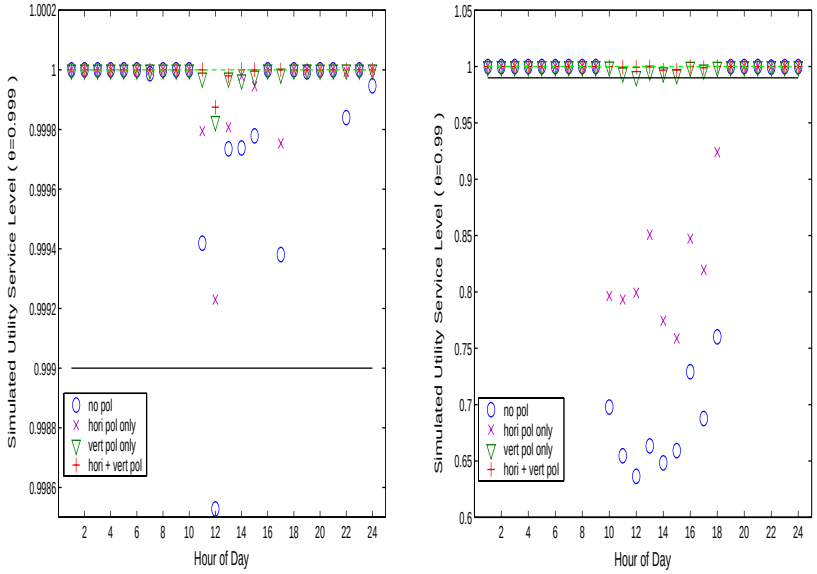
We illustrate and evaluate the RAM framework based on the the following metrics: utility service level violations for the predictable best effort classes (θ achieved for the utility), and, application service levels for the predictable best effort classes (θ perceived by each application). These metrics verify that the utility provides the levels of service that are planned, that applications receive qualities of service distinguished based on CoS, and that policing has the desired impact. We also show that that there remain significant opportunities for exploiting servers via a best effort CoS.

4.5 Results

Figures 3(a) and (b) illustrate service levels achieved by the utility over the 1000 day time scale with applications of the PBE(0.99) CoS exhibiting bad behaviour between the 10th and 18th time slots. The impact of horizontal, vertical, and a combination of horizontal and vertical policing are shown. The horizontal line at θ represents the boundary between acceptable and unacceptable performance. Without policing, there are a large number of service level violations for the PBE(0.99) class. Vertical policing is effective as a mechanism to limit application demands. The combination of horizontal and vertical policing appears to further reduce service level violations. The PBE(0.999) is spared from the bad behaviour because its applications have higher priority access to resources.

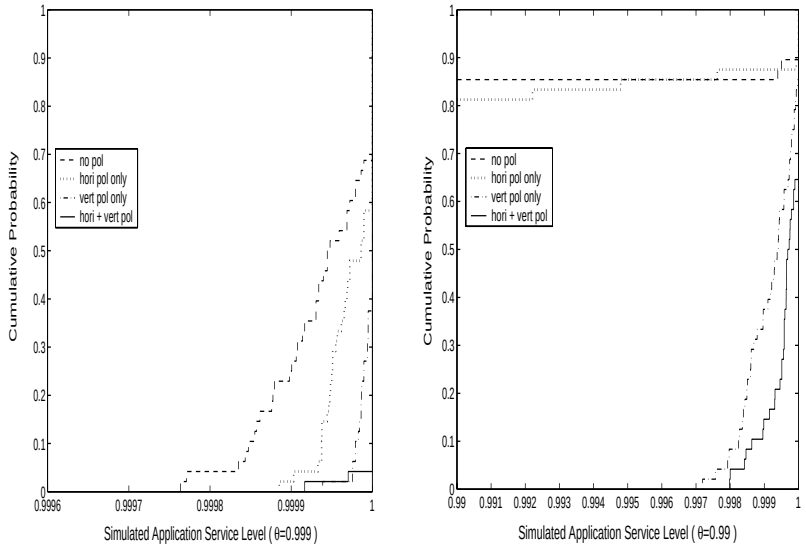
Figure 4 provides Cumulative Distribution Functions (CDF) to illustrate service levels obtained by the individual applications. For the figure, x-axis labels have been chosen on a case by case basis to best illustrate density between θ and 1. The figures correspond to the scenarios of Figure 3. As in Figure 3, Figure 4(b) shows that vertical policing is an effective mechanism for ensuring application service levels. The combination of horizontal and vertical policing further reduces service level violations. Policy mechanisms for the utility may be used to ensure that an application does not receive a better quality of service than its entitled to even if resources are available.

Figure 5 provides CDFs for the number of servers available for the best effort CoS. These represent the unused servers from the Guaranteed and PBE classes of service. Figure 5(a) shows resource availability with pool sharing for the scenarios without bad behaviour. We note that over all time slots, there are between 150 and 170 servers that are unused – which is approximately 30% of the total resource pool. These can be used to support true best effort workloads, where the servers may be clawed back on demand, or they could be used by the utility to support unexpected demands by applications. Figure 5(b) shows



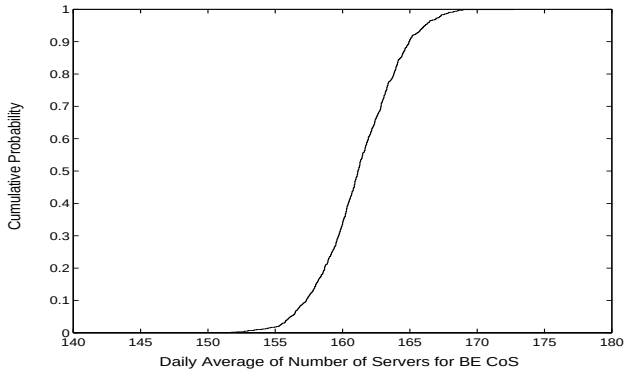
(a) $\theta = 0.999$, PBE(0.99) Bad behaviour (b) $\theta = 0.99$, PBE(0.99) Bad behaviour

Figure 3. Impact of bad behaviour and policing on utility service levels

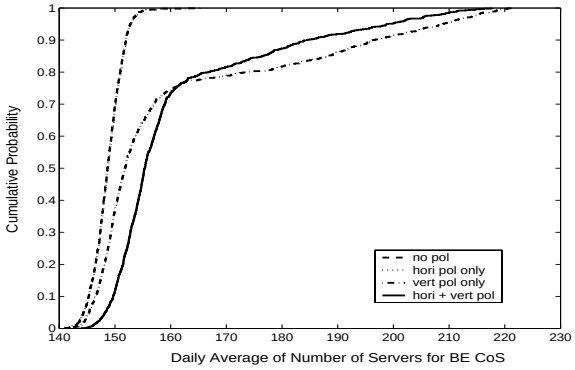


(a) $\theta = 0.999$, PBE(0.99) Bad behaviour (b) $\theta = 0.99$, PBE(0.99) Bad behaviour

Figure 4. Impact of bad behaviour and policing on application service levels



(a) No bad behaviour



(b) Bad behaviour in both PBE classes of service

Figure 5. Best effort server availability

results for the case with *both* PBE classes of service exhibiting bad behaviour between the 10th and 18th time slots. It shows that policing limits demand thereby increasing the number of unused servers. From the figure, we note that the no policing and horizontal policing only scenarios overlap while the combination of vertical and horizontal policing is the most effective.

5. Summary

This paper presents a Resource Access Management (RAM) framework for enterprise applications that access resource utilities. The framework supports admission control, CoS, and policing. A case study used measurements from 48 servers in a data center to demonstrate the effectiveness of our approach for a hypothetical utility.

Based on our case study we find that the hypothetical utility is able to offer differentiated services including predictable best effort with multiple levels of assurance. Applications in these classes of service receive service in propor-

tion to the service levels of their class. Our credit-based policing mechanism reduces service level violations in the presence of bad behaviour from applications. Policing improves the service levels yet maintains the property of service differentiation between the two predictable best effort classes. Finally, even with resource sharing and statistical assurances there remain significant opportunities for making resources available as part of a best effort service.

We find that statistical demand profiles and entitlement profiles are complementary in their support of RAM. The SDPs help to size resource pools. An EP specifies the time scales and manner in which an application must conform to its SDP. Correspondingly for a PBE CoS, a utility must also provide a time scale over which it provides its assurance level.

Last, we fully expect that from time to time applications will require more resources than expected. Our RAM framework provides the basis to decide when its possible to support such requests. Our future work includes exploring the use of multiple time scales with vertical policing, dealing with long term growth in demands (trending), exploiting EPs to look ahead at resource entitlements when considering resource allocation issues, supporting multiple resource types, exploring issues of time scale and statistical assurances, and providing quantitative support for models that take into account pricing and penalties.

References

[1] HP Openview Performance Agent. `www.openview.hp.com/products/performance`.

[2] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar. Oceano – SLA based management of a computing utility. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[3] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[4] Think Dynamics. `www.thinkdynamics.com`.

[5] Ejasent. Utility computing white paper, November 2001. `www.ejasent.com`.

[6] I. Foster and C. Kesselman. The grid: Blueprint for a new computing infrastructure, July 1998. ISBN 1-55860-475-8.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, www.globus.org, January 2002.

[8] R. Garg, R Singh Randhawa, H. Saran, and M. Singh. A sla framework for qos provisioning and dynamic capacity allocation. In *Proceedings of IWQoS 2002*, pages 129–137, Miami, USA, May 2002.

[9] J. Hellerstein, F. Zhang, and P. Shahabuddin. A statistical approach to predictive detection. *Computer Networks*, January 2000.

[10] Hewlett-Packard. HP utility data center architecture. `www.hp.com/solutions1/infrastructure/solutions/utilitydata/architecture`.

[11] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, June 1988.

[12] A. Natrajan, M. Humphrey, and A. Grimshaw. Grids: Harnessing geographically-separated resources in a multi-organisational context. In *Proceedings of High Performance Computing Systems*, June 2001.

[13] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. A scalable architecture for fair leaky-bucket shaping. In *IEEE INFOCOM (3)*, pages 1054–1062, 1997.

[14] J. Rolia, S. Singhal, and R. Friedrich. Adaptive Internet Data Centers. In *Proceedings of SSGRR'00*, L'Aquila, Italy, `www.ssgrr.it/en/ssgrr2000/papers/053.pdf`, July 2000.

[15] J. Rolia, X. Zhu, and M. Arlitt. Resource access management for a utility hosting enterprise applications. Technical Report HPL-2002-346, HP Labs, Palo Alto, CA, Dec 2002.

[16] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak. Statistical service assurances for applications in utility grid environments. In *IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 247–256, Fort Worth, TX, October 2002.

[17] D. Shen and J. Hellerstein. Predictive models for proactive network management: Application to a production web server. In *IEEE NOMs*, 2000.

[18] Sychron. Sychron Enterprise Manager, 2001. `www.sychron.com`.

[19] TerraSpring. `www.terraspring.com`.

[20] B. Urgaonkar, P. Shenoy, and T. Roscoe. Overbooking and application profiling in shared hosteing platforms. In *Fifth Symposium on Operating Systems Design and Implementation (ODSI)*, Dec. 2002.

[21] D. Xu, K. Nahrstedt, and D. Wichadakul. Qos and contention-aware multi-resource reservation. *Cluster Computing*, 4(2):95–107, 2001.

[22] S. Zhou. Lsf: Load sharing in large-scale heterogeneous distributed systems. In *Workshop on Cluster Computing*, 1992.