



A Capacity Management Service for Resource Pools

Jerry Rolia, Ludmila Cherkasova, Martin Arlitt, Artur Andrzejak¹
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2005-1
January 4, 2005*

E-mail: {jerry.rolia,lucy.cherkasova,martin.arlitt}@hp.com; artur.andrzejak@zib.de

enterprise servers,
workload analysis,
performance
modeling, capacity
planning, capacity
management

Resource pools are computing environments that offer virtualized access to shared resources. When used effectively they can align the use of capacity with business needs (flexibility), lower infrastructure costs (via resource sharing), and lower operating costs (via automation). This paper describes the Quartermaster capacity manager service for managing such pools. It implements a trace-based technique that models workload (e.g., application) resource demands, their corresponding resource allocations, and resource access quality of service. The primary advantages of the technique are its accuracy, generality, support for resource access qualities of service, and optimizing search method. We pose general capacity management questions for resource pools and explain how the capacity manager helps to address them in an automated manner. A case study demonstrates and validates the method on empirical data from an enterprise application. We show that the technique exploits much of the resource savings to be achieved from resource sharing and is significantly more accurate at estimating per-server required capacity than a benchmark method used in practice to manage a resource pool. Finally, we explain how the problems relate to other practices regarding enterprise capacity management and software performance engineering.

* Internal Accession Date Only

¹Zuse Institute Berlin (ZIB) Takustrasse 7, 14195 Berlin-Dahlem, Germany

A Capacity Management Service for Resource Pools

Jerry Rolia, Ludmila Cherkasova, Martin Arlitt

Hewlett-Packard Laboratories

1501 Page Mill Road,

Palo Alto, CA 94304, USA

{jerry.rolia,lucy.cherkasova,martin.arlitt}@hp.com

Artur Andrzejak

Zuse Institute Berlin (ZIB)

Takustraße 7

14195 Berlin-Dahlem, Germany

artur.andrzejak@zib.de

Abstract

Resource pools are computing environments that offer virtualized access to shared resources. When used effectively they can align the use of capacity with business needs (flexibility), lower infrastructure costs (via resource sharing), and lower operating costs (via automation). This paper describes the Quartermaster capacity manager service for managing such pools. It implements a trace-based technique that models workload (e.g., application) resource demands, their corresponding resource allocations, and resource access quality of service. The primary advantages of the technique are its accuracy, generality, support for resource access qualities of service, and optimizing search method. We pose general capacity management questions for resource pools and explain how the capacity manager helps to address them in an automated manner. A case study demonstrates and validates the method on empirical data from an enterprise application. We show that the technique exploits much of the resource savings to be achieved from resource sharing and is significantly more accurate at estimating per-server required capacity than a benchmark method used in practice to manage a resource pool. Finally, we explain how the problems relate to other practices regarding enterprise capacity management and software performance engineering.

1 Introduction

The Quartermaster (QM) capacity manager is a management service for resource pools that support enterprise applications. It automatically searches for workload assignments that best meet objectives such as consolidation, load leveling, and problem solving. Consolidation packs workloads onto a small set of resources, load leveling distributes workloads across a fixed size resource pool to lessen the likelihood of service level violations, and problem solving recommends alternative assignments to lessen the impact of unexpected demands on a particular resource. The service evaluates alternative workload assignments more quickly and accurately than human operators. As a result it can lower administrative costs, reduce the risks associated with resource sharing, and improve the use of resource pool infrastructure.

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) that provide access to shares of capacity. Application workloads are assigned to the containers which are then associated with resources. A resource pool and its resources have capacity attributes, e.g., CPU, memory, I/O operation rates, and bandwidths, each with limited capacity.

Figure 1 provides some motivation for resource pools and the potential sharing they offer. It illustrates the average CPU utilization of servers at an enterprise computing site. Each bar gives the average CPU utilization of a server. The figure shows that many servers have low CPU utilizations. The site is typical of many enterprise

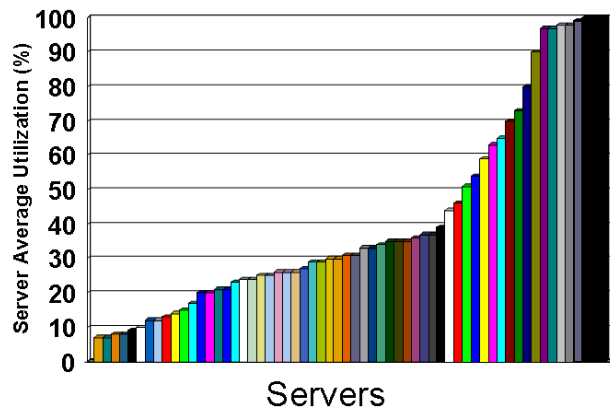


Figure 1: Average CPU Utilization of Servers at an Enterprise Data Center.

data centers. There is an opportunity to consolidate many workloads on a smaller number of servers. The figure also shows that some servers are over utilized. Ideally, it would be possible to make more resources available to workloads if they need them.

We consider the problem of hosting enterprise applications in such resource pools. These applications often operate continuously, have unique time varying demands, can share individual resources such as a CPU or may require many resources, and may have different service level expectations. These complexities make it difficult and time consuming for human operators to make effective use of such pools.

When managing resource pools there are numerous *capacity management questions* that must be answered to ensure that the resources are used effectively. For example: how much capacity is needed to support the current workloads? Which workloads should be assigned to each resource? What is the performance impact of workload scheduler and/or policy settings that govern sharing? How should workloads be assigned to make workload scheduler and/or policy settings most effective? What should be done when a resource doesn't have sufficient capacity to meet its workloads' needs? How many resources will be needed over a planning horizon? The QM capacity manager can be used to automate analysis steps that address these questions. We demonstrate the effectiveness of the method using a case study. The technique proves to be significantly more accurate at anticipating capacity requirements than a method, the *benchmark method*, used in practice.

The following sections describe related work, the QM capacity manager technique, our approach towards addressing the above questions, and a case study. We close with a summary of our work.

2 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 general 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. Furthermore, today's enterprise data centers can have hundreds of large shared servers and thousands of lightly utilized smaller server resources. To reduce management costs, enterprises are evolving their data center architectures to exploit resource pools. Resource pool automation and virtualization capabilities promise lower costs of ownership for similar quantities of capacity. However, methods are needed that systematically address the capacity management questions for such environments.

We employ a trace-based approach to model the sharing of resource capacity for resource pools. The general idea behind trace-based methods is that traces capture past demands and that future demands will be similar. Many groups have applied trace-based methods for detailed performance evaluation of processor architectures [1]. 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 on-going capacity management.

Traces have been used to support what-if analysis that consider the assignment of workloads to consolidated servers. AOG [3] and TeamQuest [4] now offer products that employ trace-based methods to support consolidation exercises. Our methods are similar, but go further by addressing issues including resource access Quality of Service (QoS) [5] and fast optimizing searches in support of on-going automated management [6]. The work described in this paper integrates the concepts of QoS and fast optimization heuristics and demonstrates the resulting service in a case study. Qualities of service are realized by associating workloads with resource access classes of service that have specific service level objectives.

Others have used traced-based methods to support the creation of performance models for software performance engineering exercises [7]. Clearly traces can be exploited at many levels of abstraction. We suggest that they are an excellent mechanism to bridge the divide between software performance engineering and capacity management.

The approach we consider is targeted towards enterprise application environments. In these environments workloads are often repetitive and predictable. Those responsible for the applications are typically risk averse. The approach we employ aims to reduce risk. We believe that the approach is more appropriate than pure priority and market based mechanisms for resource allocation that do not offer any assurances regarding qualities of service [10][11]. However, at this time our belief remains a subject of debate.

3 QM Capacity Manager Service

We have developed the Quartermaster Capacity Manager Service to address the capacity management questions. The service is part of the Quartermaster resource utility [8]. The tool has several inputs that are described below. These include traces of application workload demands on capacity attributes, an association of the demands with classes of service, resource access Class of Service (CoS) constraints for each class of service [2], constraints on application assignments (e.g., which applications may or may not be assigned to the same resource), and a specification (limits) for the capacity attributes of resources in the resource pool. The service implements several kinds of analysis including the simulation of a workload assignment (e.g., what-if) and an optimizing search to find an assignment of applications to resources that best satisfies an objective (e.g., consolidation, load leveling, problem solving) subject to the constraints. We now describe each of these inputs in greater detail. This is followed by a description of the components that implement the service.

Consolidation refers to an objective of minimizing the number of resources, e.g., computer servers, needed to support the workloads. Load leveling balances workloads across a resource pool of known size to reduce the likelihood of service level violations. Problem solving focuses on a single resource. It reports on the impact of moving one or more workloads from a resource, that may not be satisfying its service level objectives, to other resources and recommends which workloads to migrate. Combinations of these functions can be used to address the capacity management questions of Section 1. This paper focuses on the consolidation objective.

Each workload has demands upon capacity attributes such as CPU, memory, network and storage usage. Workload traces are gathered by existing monitoring systems such as HP's OpenView Performance Agent or the Windows Performance Monitor to record these demands. Such traces are gathered as a matter of best

practice in professionally managed enterprise environments to support capacity management exercises. Often they are gathered on a per-server basis, and sometimes on a per-application workload basis. If they are only gathered on a per-server basis we treat the aggregate load on the server as a single workload in our analysis.

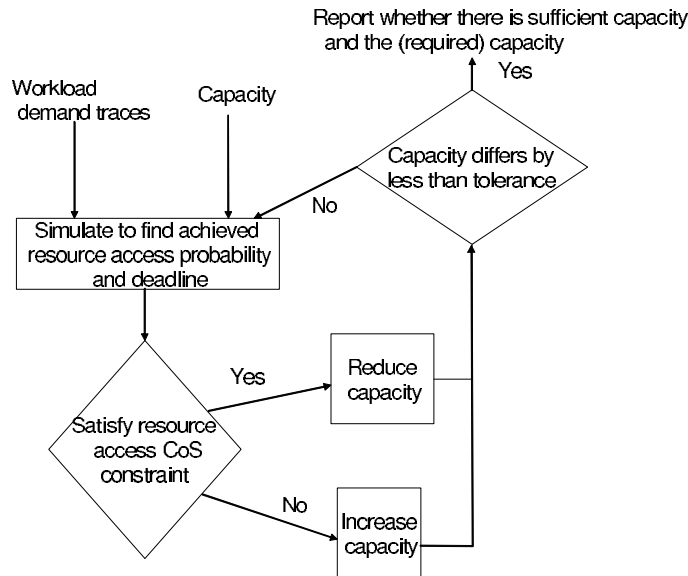


Figure 2: Simulation Algorithm

Each resource access CoS constraint includes a probability, θ , and a deadline, s . θ is the probability that a workload associated with the CoS receives a unit of capacity when requested. Those requests for units of capacity that are not received on demand are deferred. Deferred units of capacity must be made available to the workload within the deadline s . If either part of the constraint is unsatisfied then there is a service level violation. The demands in traces are used to compute the empirical value for θ for the traces and/or the minimum capacity required to ensure that a CoS constraint is satisfied. Multiple CoS are supported so that workloads with different needs can be accommodated. We refer to the minimum capacity needed to satisfy all resource access CoS constraints jointly as the *required capacity*. We define resource access CoS further in the next section.

Application placement constraints can also be important. For example, some workloads must be hosted on separate resources for the purpose of availability. Some workloads must be associated with specific resources because of licensing constraints. The capacity manager supports these kinds of placement constraints.

The capacity manager has two components. A simulator component emulates the assignment of several applications to a single resource. It traverses the traces of observed demands to estimate a required capacity that satisfies the resource access QoS constraints. The required capacity can be compared with resource capacity limits to support what-if analysis. An optimizing search algorithm examines many alternative assignments and reports the best solution found for an objective such that the placement constraints are satisfied. These components are described in the following sections.

3.1 Simulator and resource access CoS constraints

Figure 2 illustrates the simulator algorithm. The simulator considers the assignment of a set of workloads to a single resource. It replays the workload traces, compares the aggregate demand 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 constraints 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 constraints are satisfied. Next, we consider the computation of the CoS statistic θ in detail.

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.

To define resource access CoS more formally, consider one CoS 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, 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 a deadline. We express the deadline as an integer number of slots s .

As an example, suppose we replay the A workload traces with respect to a server with L' CPUs that has a scheduler that allocates/re-allocates CPUs to workloads each slot based on the current use of capacity. All the workloads are associated with a CoS with resource access probability θ . The CoS offers CPUs to its workloads with probability no worse than θ during each slot as evaluated on a weekly basis. To give an intuitive feel for the impact of the resource access probability, an aggregate workload that requires a CPU week (10,080 minutes) worth of CPU time and receives a resource access probability of 0.999 receives all but 10 minutes of CPU time when requested. A resource access probability of 0.99 implies that all but 100 CPU minutes are received when requested. Suppose that one of the A workloads, that we name *unlucky*, has a demand of 100 CPU minutes while the remaining workloads we consider have a total demand of 9,980 minutes. In the degenerate case, all of unlucky's demands may get deferred while the other workloads may receive all of their demands when requested. If this is a likely yet unacceptable scenario then the workload should be placed in its own CoS.

Note that system CPU schedulers often re-allocate shares of resources on the time scale of sub-seconds to 15 seconds for more transparent processor sharing. We rely on system schedulers to ensure that workloads do not starve, e.g., become unlucky, during the many scheduling cycles that contribute to an observation. Also, the measurements provided by monitoring tools are typically an average value over the entire interval. As a result the θ value applies to the m minute slot time scale as evaluated on a weekly basis.

Next, we contrast the use of required capacity L' with the use of a utilization metric alone for workload placement. In our earlier work [2], we limited the assignment of workloads to servers by constraining the maximum aggregate utilization of the target servers. For example, we assigned workloads subject to the constraint that the maximum aggregate CPU utilization never exceeds a limit u , e.g., $u = 80\%$. We show

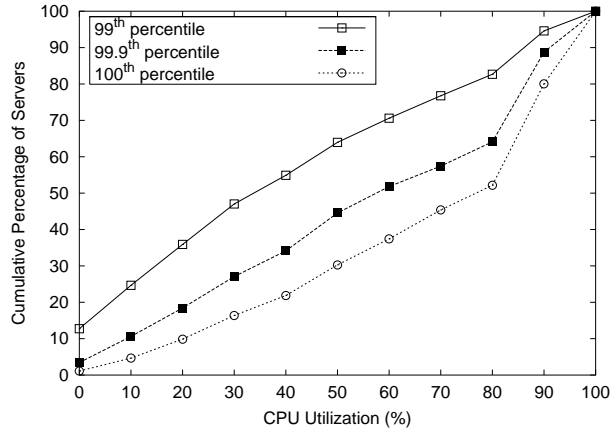


Figure 3: Cumulative Distribution Function for percentiles of CPU Utilization for 1459 Servers

that resource access probability with a deadline provides a more practically useful constraint than utilization constraints alone.

Figure 3 gives a cumulative distribution function (CDF) for the percentiles of CPU utilization for 1459 enterprise servers from three data centers over a one month period in 2001. For each server, a CPU utilization measure is recorded every five minutes for the month giving 8,640 measurements. For each server, we sort these measurements in increasing order. We present a CDF for the percentiles of these utilization values for the servers. We present results that correspond to the 100th, 99.9th and 99th percentiles of the sorted values. The illustration is with respect to buckets that show the percentage of servers that have a CPU utilization u in the range $0\% \leq u < 10\%$, $10\% \leq u < 20\%$, up to $90\% \leq u < 100\%$, and $u = 100\%$ where u is the CPU utilization for the given percentile. The 100th percentile curve, as read from the 70% CPU utilization value on the x axis, shows that 45% of servers each have a maximum utilization that is greater than or equal to 0% and less than 80%. It follows that 55% of servers each have one or more CPU utilization measurements greater than or equal to 80%. The 99.9th percentile curve indicates that approximately 42% of servers each have 8 or more 5-minute observations with a CPU utilization greater than or equal to 80%. The 99th percentile curve reveals that 23% of servers each have 100 or more CPU utilization measurement values that are greater than or equal to 80%.

The figure shows that a constraint that limits target server utilization to 80% causes us to omit approximately two thirds of the servers from workload placement exercises. Nearly one quarter of the servers each have one hundred or more utilization values greater than or equal to 80% over the month. Simple utilization constraints may omit too many workloads from a consolidation analysis. This problem is particularly acute when the workloads are being assigned to target servers that have similar capacity. This is the typical case for the on-going management of a resource pool. We use resource access probability and the deadline to capture the notion that a quantified amount of demand may be deferred for a limited amount of time.

Workloads can exploit the CoS parameters in the following way. An interactive workload may require a very high probability and may not benefit much from the notion of a deadline. Degraded access to resources will likely result in higher application response times rather than any deferral of work. A batch workload may be more flexible regarding when it receives its resources. It could have a lower resource access probability, e.g., $\theta = 0.5$ which gives a window of time that is approximately double the demand to complete the work, but could rely on the deadline to ensure its demands are not deferred by too much.

When multiple CoS are involved, the simulation component schedules access to capacity in the following way. Capacity is assigned to the highest priority, e.g., greatest θ value, CoS first. If a CoS does not have a probability of 1.0, then it is only offered its specified probability of access with remaining demands deferred to future intervals. The remaining capacity is assigned to the next lower CoS. This repeats for other lower classes of service.

The required capacity of each attribute is found as follows. If the current capacity satisfies the CoS constraints then the algorithm reduces the capacity value for the attribute. If the current capacity does not satisfy the constraints, the algorithm increases the value for capacity up to the capacity limit L of the attribute. The algorithm completes when it is found that the constraints 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 constraints are satisfied (for each attribute). If so, the resulting value for capacity is reported as the required capacity (for each capacity attribute).

3.2 Optimizing search

Figure 4 illustrates the optimizing search algorithm. An objective, constraints, workload 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 3.1. The results of the simulations are then used to compute a score for the objective's corresponding objective function. If there is little improvement in the score then the algorithm reports a configuration that achieved the best score and terminates. Otherwise a new configuration is enumerated using a genetic algorithm [6] and the simulation process is repeated.

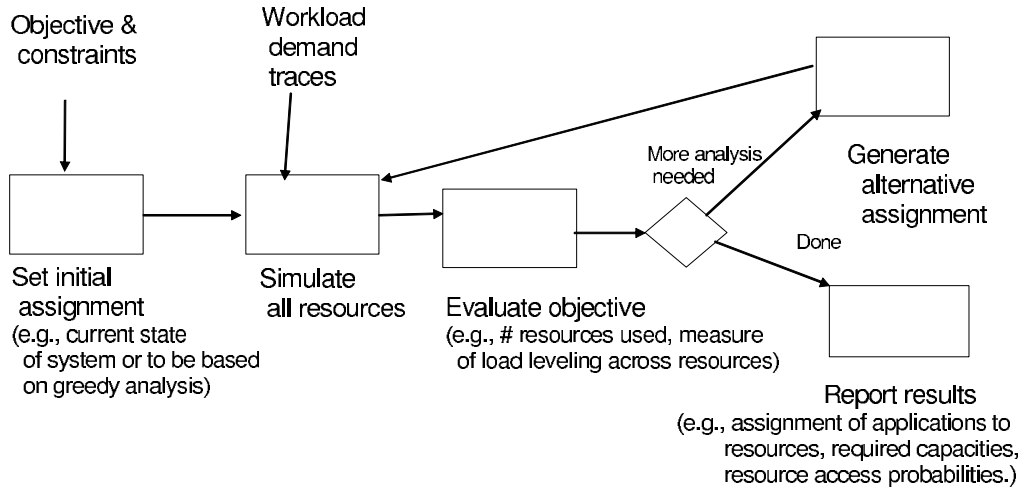


Figure 4: Optimizing Search

We have two methods for setting the initial configuration for the system. The first method takes the current configuration of a system. It is most appropriate in support of what-if analysis and where solutions are required that have only small numbers of changes to the environment.

The second approach applies a greedy algorithm to choose the initial placement. The greedy approach is a clustering algorithm that starts with each workload in its own cluster. The algorithm then simulates the merging of pairs of clusters to determine the amount of sharing between workloads in the clusters. A sharing distance $\Delta_{i,j}$ between a pair of clusters i and j is defined as the difference between the required capacity of

the merged cluster $L^{i,j}$ and the sum of the two required capacities L^i and L^j of the original clusters. It is computed as follows.

$$\Delta_{i,j} = \frac{L^{i,j}}{L^i + L^j}$$

The lower the sharing distance the greater the sharing of resources between workloads in the clusters. For each iteration, we merge the pair of clusters with the lowest sharing distance, such that placement and resource access CoS constraints are satisfied. A weighting value is specified to factor the importance of each capacity attribute in this equation. The more important it is to share a resource, e.g., CPU or memory, the greater its weight. The greedy algorithm is most appropriate when consolidating to a new resource pool or when many changes are permitted, e.g., as supported by virtualization mechanisms that enable the migration of a workload from one resource to another with minimal disruption to the service that it provides and minimal human effort.

The consolidation objective causes a search for a workload assignment that satisfies constraints and uses a small number of servers. The number of servers of an intermediate solution is used as its score. The solver reports a solution that had the lowest score. The solution gives the assignment of each workload to each server and the required capacity of each server. Other objectives (e.g., load leveling) have different expressions for assigning scores. They are also implemented within the same framework.

3.3 Assumptions and Validation Method

This section considers several assumptions of the trace based approach. The technique assumes that the traces are of sufficient duration to capture the correlations between workload demands, and that each workload’s demand trace is representative of its future demands. These assumptions support our use of the capacity manager for forward looking management. With regard to specific resource pools, we assume that the simulator’s approach for scheduling access to resources is compatible with the scheduling mechanism in the resource pool. Last, we assume that the workloads operate in an additive manner, i.e., when the workloads are reassigned to different resources we assume that their demands will not change significantly. We now discuss each of these assumptions in more detail.

The capacity manager considers the sensitivity of required capacity to correlations in workload demands by implementing a technique that exaggerates correlations. The trace data we exploit typically has an observation for each workload for five minute slots. We can also perform the analysis at more coarse time scales that exaggerate correlations. For example, we may perform the analysis with 1 hour slots where the demands for each slot are the peak capacity attribute values of the 12 corresponding 5 minute observations. As a result, the aggregate demand is greater than or equal to per-5 minute aggregate demand values for the same hour. This allows for the possibility that future per-workload demands will be similar but may shift slightly in time causing greater peaks. The approach leads to more pessimistic results. We rely on a validation method to help decide what slot duration is most appropriate for studying a particular system’s behavior.

We envision the capacity manager being used in an on-going capacity management scenario where recently gathered data is used to prepare a capacity plan for the near future. The capacity management plan may call for the re-assignment of some workloads to make better use of the resource pool. We assume this process is repeated periodically, e.g., weekly, bi-weekly or monthly. This scenario also corresponds to the validation method we present in the case study.

The validation method is as follows. Given a large amount of workload data, e.g., several months, the analysis is applied to contiguous subsets of the data. For each subset, a part is used as training data to report a solution. This solution gives an assignment of workloads to servers and the *predicted required capacity* for each server. Then the remaining part of the subset of data is used to validate how that solution would have behaved in practice. It gives the *actual required capacity* for each server for the remaining data. This is sometimes referred to as a *walk-forward test* in the literature. We define per-server error as the predicted required capacity minus the actual required capacity. A characterization of errors over time gives a measure of confidence for the results of the capacity manager for a particular set of workloads.

As a simulation-based approach, the resource scheduler of the simulation component can model any scheduling mechanism that operates on data offered in the workload traces. For example, it is compatible with the HP-UX Process Resource Manager (PRM) scheduler [9] when the QM capacity manager’s highest priority CoS is given a resource access probability of one.

We note as well that demands in traces are typically averages. The PRM scheduler supports a goal mode to ensure application level QoS for interactive workloads at the sub-five minute timescale. It has the same role for this capacity management approach as the queuing analysis step described in the related work section. The basic idea is that the scheduler allocates sufficient resources to maintain a particular target utilization level within an allocation. For example, if the target utilization, g , of an allocation is 50% and a workload has a demand of one CPU then the scheduler allocates two CPUs. The QM capacity manager models the goal mode by scaling each CPU demand in a trace, d , using the target utilization goal, g , to an allocation value $\frac{d}{g}$. The allocation values are used when computing required capacity. This ensures sufficient resources are available for interactive workloads.

We have not yet addressed the additive assumption in our work. This is in part a function of the virtualization mechanisms that are used to enable sharing, and in part affected by the detailed interactions of application demands below the sub-five minute boundary. We regard this as future work.

4 Case Study

To assess the accuracy and benefits of our method we obtained demand traces from an enterprise application environment. The environment is a large order processing system. It has application components that tend to require resources at the same time. We chose this application because it allows us to explore the benefits of resource sharing for applications that have highly correlated demands. We demonstrate the effectiveness of the QM capacity manager for the environment.

The data set spans 32 weeks, collected between October 2003 and June 2004. It includes traces that describe 26 application servers each with either 6 or 8 CPUs. We conduct our study with respect to a particular reference processor speed. The applications are hosted on the equivalent of approximately 138 of these reference CPUs.

The order processing system includes interactive and batch workloads. We use a target CPU utilization of $g = 50\%$ for the interactive workloads. This results in an allocation that is double the observed demand. The purpose is to keep the interactive response times low. The batch workloads have a target CPU utilization of 100%. Thus for batch, the allocations are the same as the observed demands. The original system requires 207 CPUs when its demands are scaled in this way.

There are both production and non-production (e.g., test and development) workloads in our data set. The production workloads are placed in the highest CoS. The non-production workloads are associated with a second CoS. This lets us associate the workloads with different resource access constraints.

We present three sets of results. The first set considers the simulation component only. It demonstrates the sensitivity of required capacity to slot size, resource access CoS constraints, the stability of predicted required capacity based on the number of weeks of data used for prediction, and compares the prediction of required capacity with the actual required capacity. The second set of results demonstrates the optimizing search method for the consolidation objective. The third set of results focuses on on-going capacity management. We apply the consolidation method to successive subsets of trace data and report how accurately the technique predicts required capacity over time. We conduct our analysis for the CPU demand attribute only to simplify the presentation of results and report required capacities as real values rather than integers for the purpose of illustration.

4.1 Sensitivity Analysis

The first set of results rely on the simulation component to determine the required capacity for the order processing workloads as a whole. This assumes a sufficiently large server that is able to share its resources among all workloads. Though this is an unlikely consolidation scenario, it gives a lower bound for required capacity. We use this case to show the sensitivity of required capacity to slot size, e.g., 5 minutes, 15 minutes or 1 hour, the impact of resource access CoS, the stability of predicted required capacity based on the number of weeks of data used for prediction, and the prediction of required capacity with actual required capacity.

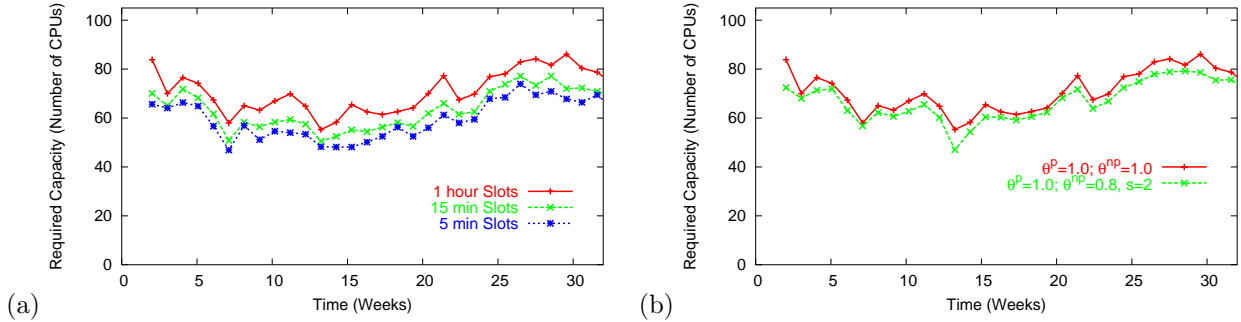


Figure 5: (a) Predicted Required Capacity for Several Slot Sizes, (b) Impact of QoS on Required Capacity.

For the first scenario, both CoS have a resource access probability of 1.0. Figure 5(a) shows the required capacity for 5 minute, 15 minute, and 1 hour slot sizes. The values are reported weekly as computed using the past two weeks of data. The corresponding peak required capacities are 79, 80, and 86 CPUs, respectively. As expected, as slot duration increases the correlation in demands also increases leading to larger values for required capacity. Note that these peaks for required capacity are significantly lower than both the 207 CPUs for the comparable original system, i.e., with allocations scaled based on utilization goals, and the 138 CPUs of the original system.

Figure 5(b) considers 1 hour slots. It shows the required capacity for the case where both CoS have resource access probability of 1.0, as referred to in the figure as $\theta^p = 1.0$ for production and $\theta^{np} = 1.0$ for non-production, and for the case where the non-production workloads have a resource access probability of $\theta^{np} = 0.8$ and a deadline of $s = 2$ slots. This means that non-production workloads can have some demand deferred by up to 2 slots. This is a reasonable scenario in that priority is given to production workloads yet the non-production loads still receive significant capacity when requested. From the figure we see that required capacity drops from a peak value of 86 CPUs to 79 CPUs. This information is helpful in several ways. It may serve to lower management or software licensing costs if fewer servers are needed to support the workloads, or may simply provide a quantitative assessment of the impact of current capacity, e.g., 79 CPUs, on the lower CoS workloads.

Next, we show the impact of the choice of the number of weeks of historical data used for analysis on the stability of predicted required capacity. Figure 6 shows predicted values for required capacity for 1 hour slots with 3, 6, and 12 week subsets of data. For the 3 week subsets, 2 weeks of training data are used to predict required capacity for the third week. The analysis is repeated weekly. The 6 week subsets use 4 weeks of training data to predict the required capacity of the next 2 weeks with the analysis repeated every two weeks; the 12 week subsets use 8 weeks of training data to predict required capacity for the following 4 weeks with the analysis repeated every four weeks. Predicted values for required capacity are computed using hourly slots. Actual values for required capacity are computed using 5 minute slots. The more weeks that are used for prediction the greater the likelihood of larger peak values arising. From the figure we see that longer data set sizes tend to yield larger estimates for required capacity. The 4 and 8 weeks subsets offer more stable estimates for required capacity than the 2 week subsets. Hence we choose to use 4 week subsets for the remainder of our study. In general, the training portion of the subset should have sufficient data to capture peaks that arise due

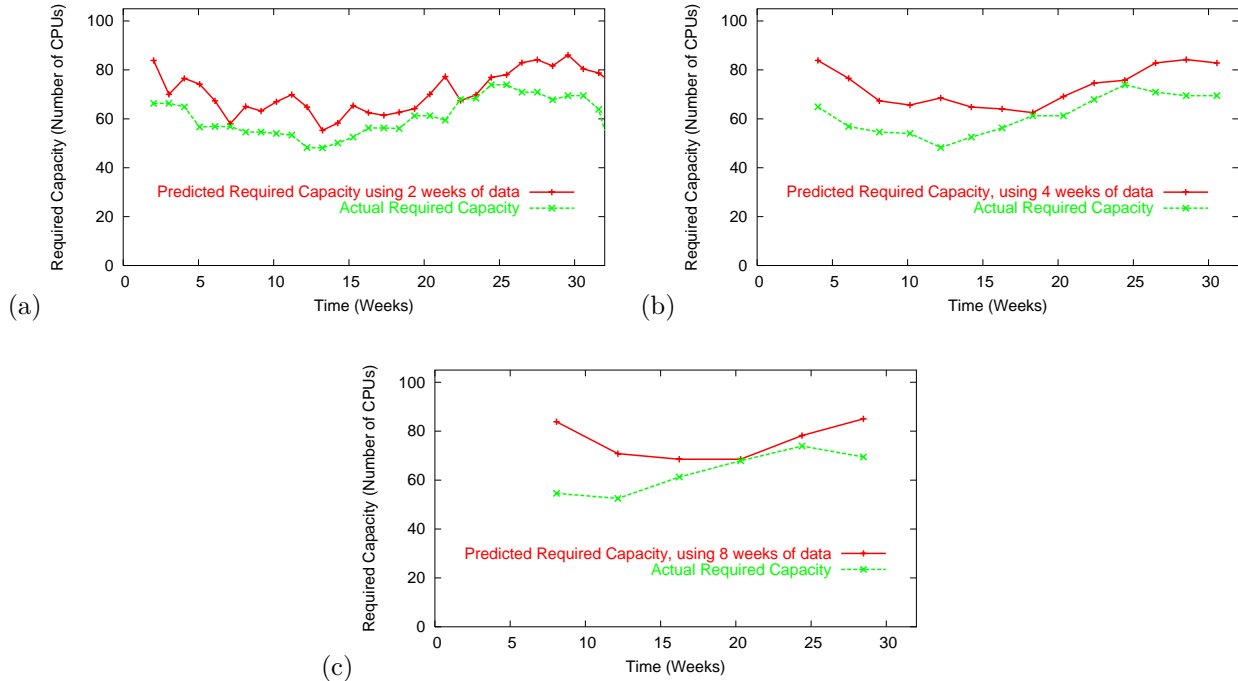


Figure 6: (a) 3-week subset: 2 weeks predicting 1 week, (b) 6 week subset: 4 weeks predicting 2 weeks, (c) 12 week subset: 8 weeks predicting 4 weeks.

to predictable but time varying business needs. For example, if end of month capacity requirements are always larger than mid-month requirements, the training portion of the subset of data should always include end of month data.

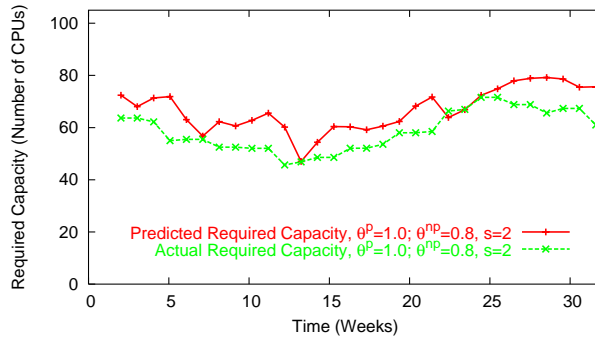


Figure 7: Predicted and Actual Values for Required Capacity with QoS

Figure 7 compares predicted values for required capacity with actual values. Predicted values for required capacity are computed using 1 hour slots. Actual values for required capacity are computed using 5 minute slots. This figure corresponds to the scenario where the production workloads have a resource access probability of $\theta^p = 1.0$ and the non-production workloads have a resource access probability of $\theta^{np} = 0.8$ with a deadline of $s = 2$ slots. With hourly slots the predicted values are typically greater than the actual values. Other figures not included in the paper show similar accuracy for the cases where both CoS have a resource access probability of 1.0. We note that the results we present do not take into account any trends that can be observed from the data. Nor do they consider any business planning information that can be used to predict changes in demands more directly. Such information is likely to improve predictions.

4.2 Consolidation Method

Figure 8(a) shows the sum of per-server required capacity for the assignment of the workloads to servers with CPU capacity limits of 16, 32, 64 and 128 CPUs. The required capacity values are reported for four week subsets of data. Both CoS have a resource access probability of 1.0. The peak required capacities over all subsets of data are 101.5, 90.6, 85.9, and 84.2 CPUs, respectively. As the number of CPUs per server increases from 16 to 32 we see a drop in the required capacity values. This is because larger servers provide greater opportunities for resource sharing. The 16 CPU servers suffer the most from *resource sharing fragmentation*. The required capacity values for 32 and 64 CPU servers drop, but with diminishing returns. Improvements are smaller still for the single large 128 CPU server. Most of the sharing that can be achieved is realized by the 64 CPU servers. The 16 CPU servers realize all but 20% of the savings possible from sharing, with respect to the 128 CPU server. We note that when a server is used, not all of its CPUs are exploited. For each scenario, the total number of CPUs for the resource pool is the product of the maximum number of servers needed and the number of CPUs per server. The total number of CPUs needed for the 16, 32, 64, and 128 CPU per server scenarios are 112, 96, 128, and 128 CPUs, respectively. The difference between a server’s per-server capacity limit and its required capacity is available to support additional work. Often, unused CPUs can be disabled to reduce server costs and software licensing costs. These can then be enabled on-demand if and when they are needed.

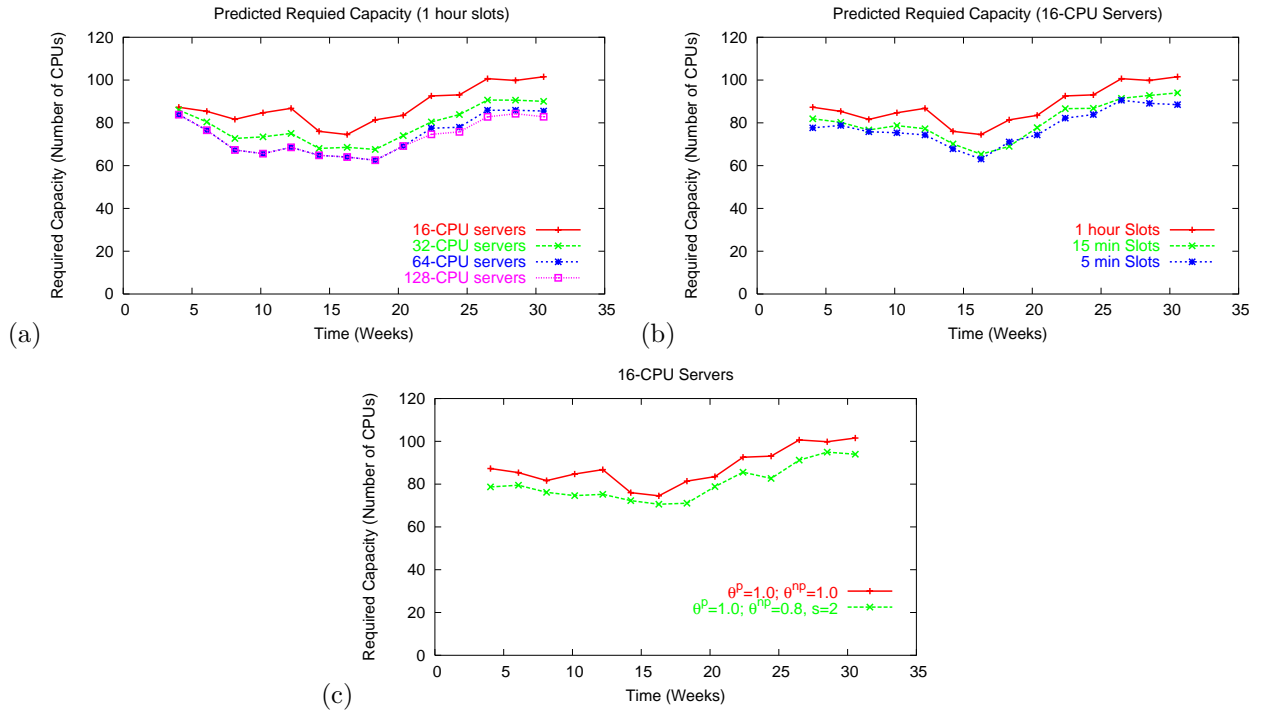


Figure 8: (a) Consolidation to 16 CPU servers, (b) Impact of Slot Size on Required Capacity, (c) Impact of QoS on Required Capacity

Figure 8(b) shows the impact of slot size on the sum of required capacity for the 16 CPU server case. The slot size has less of an impact than in the large server case. This is reasonable as the search algorithm aims to co-locate workloads with smaller or negative demand correlations onto the same server. Thus, our method of increasing slot duration to exaggerate correlations has less of an impact because there are fewer positive correlations to exaggerate as compared to the large server scenario where all the workloads are assigned to the same server.

Figure 8(c) shows the sum of required capacity values for the 16 server scenario, for 1 hour slots, for the case where both CoS have a $\theta = 1.0$ and for the case where the production workloads have a $\theta^P = 1.0$ and the

non-production workloads have a $\theta^{np} = 0.8$ with a deadline of $s = 2$ slots. The peak of required capacity values are 101.5 and 95, respectively. The sum of required capacities drops by 6.5 CPUs case where both CoS have a resource access probability of 1.0. The large server case differed by 7 CPUs. This shows that most of the advantages of the quality of service specification are achieved on the 16 CPU servers.

4.3 On-going Capacity Management

This section focuses of a validation of the capacity manager for on-going management. We apply the optimizing search consolidation method to 6 week subsets of data to assign and re-assign workloads to servers over time. For each subset, we use 4 weeks of training data to compute a predicted required capacity for 1 hour slots. This is then compared with the computation of the actual required capacity for the next two weeks as computed using 5 minute slots. We define error for a server as its predicted required capacity minus its actual required capacity. Positive errors correspond to an over-estimate of required capacity. Negative errors correspond to an underestimate. Our figures show the distribution of errors for the individual servers.

We compare the accuracy of the technique with a benchmark method used to manage a resource pool. Its advantage is that it is straightforward for an operator to calculate without significant effort. It computes a ratio between the sum of per-application peak demands and the observed aggregate demand for all the workloads on all servers as a whole. We compute this ratio using the four most recent weeks of data and use it to predict the demand for each server for the next two weeks.

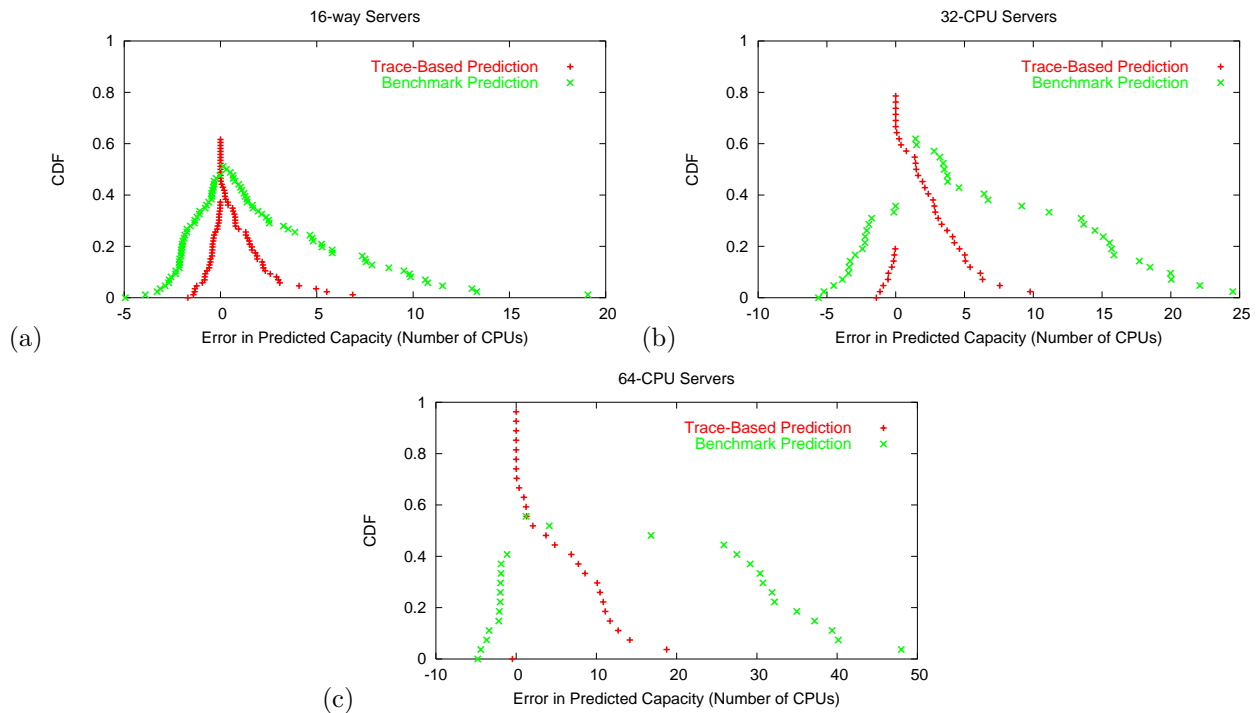


Figure 9: Error: Predicted Required Capacity minus Actual Required Capacity

Figure 9(a) shows the distribution of errors for the QM capacity manager’s trace-based method for the case study data on 16 CPU servers. 38% of the errors are negative, for these cases the error was never greater than 2 CPUs. 62% of the errors are greater than zero. These are over-estimates for required capacity. Together, the estimates are within 2 CPUs of per-server required capacity values 85% of the time. In the remaining 15% of cases, the capacity manager over-estimates required capacity by more than 2 CPUs. From the figure, we see

that the benchmark approach has errors that are significantly larger and more frequent than the trace-based approach.

Figures 9(b) and (c) show the distribution of error values for servers with 32 and 64 CPU, respectively. The trace-based method becomes more optimistic as the server size increases. This suggests that an analysis using slots of less than 60 minutes is more appropriate for these servers. Again, the benchmark approach has errors that are larger and more frequent than the trace-based approach.

The accuracy of the trace-based method provides confidence to system operators. It offers insights regarding how much workload can be assigned to a server and what the impact is likely to be of moving workloads from one server to another.

5 Summary and Conclusions

This paper describes a trace-based approach to capacity management for resource pools. We define an empirical measure of required capacity and resource access quality of service constraints. We describe an optimizing search method with a focus on consolidation and re-consolidation for the purpose of on-going management as an objective for the search. Results from approximately 1500 servers show that our approach to resource access quality of service is superior to our earlier utilization based method, which is too constraining for capacity management, in particular for on-going management. Case study results show that our consolidation approach realizes much of the opportunity for resource sharing for a large order processing system. It automates the process of deciding workload assignments for resource pools and is much more accurate than a benchmark approach for capacity management used in practice.

Though our focus has been on a consolidation objective for the optimizing search, other objectives including load leveling and problem solving are also supported. Together these automate functions needed by operators to address the capacity management problems described in Section 1.

The trace-based approach provides confidence to system operators as they manage resource pools. However many challenges still remain. Changing business conditions, new applications, changing applications, and issues such as run-away processes in applications can all have big impact on past and future resource usage. To improve the approach, methods are needed to systematically obtain capacity requirement estimates from business planning systems, and from software design, development, and maintenance processes. Software performance engineering exercises could provide initial estimates for the capacity requirements of new or modified applications so that their impact on resource pools can be anticipated prior to deployment. Methods are needed to assure the quality of historical trace data. These are all requirements for further automation of capacity management for resource pools.

6 References

- [1] 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, 2004, pages 287-292.
- [2] A. Andrzejak, J. Rolia and M. Arlitt, Bounding Resource Savings of Utility Computing Models, HP Labs Technical Report HPL-2002-339, 2002.
- [3] www.aogtech.com
- [4] www.teamQuest.com
- [5] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, Statistical service assurances for applications in utility grid environments. *Performance Evaluation Journal* 58(2004) pp 319-339.
- [6] J. Rolia, A. Andrzejak, and M. Arlitt, Automating Enterprise Application Placement in Resource Utilities, 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2003, Ed. M. Brunner and A. Keller, Springer Lecture Notes in Computer Science, Volume 2867, 2003, pp. 118-129.

- [7] C.E. Hrischuk, C.M. Woodside, and J. Rolia, Trace-Based Load Characterization for Generating Software Performance Models, *IEEE Trans. on Software Engineering*, v 25, n 1, pp 122-135, Jan. 1999.
- [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] www.hp.com/go/PRM
- [10] Appleby K., Fakhouri S., Fong L., Goldszmidt G. and Kalantar M.: Oceano – SLA Based Management of a Computing Utility. Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, May 2001.
- [11] 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.