Capacity Management and Demand Prediction for Next Generation Data Centers

Daniel Gmach
Technische Universität München,
85748 Garching, Germany
daniel.gmach@in.tum.de

Jerry Rolia and Ludmila Cherkasova
Hewlett-Packard Laboratories
Palo Alto, CA, USA
{jerry.rolia,lucy.cherkasova}@hp.com

Alfons Kemper
Technische Universität München,
85748 Garching, Germany
alfons.kemper@in.tum.de

Abstract

Advances in server, network, and storage virtualization are enabling the creation of resource pools of servers that permit multiple application workloads to share each server in the pool. This paper proposes and evaluates aspects of a capacity management process for automating the efficient use of such pools when hosting large numbers of services. We use a trace based approach to capacity management that relies on i) *a definition for required capacity,* ii) *the characterization of workload demand patterns,* iii) *the generation of synthetic workloads that predict future demands based on the patterns, and* iv) *a workload placement recommendation service. A case study with 6 months of data representing the resource usage of 139 workloads in an enterprise data center demonstrates the effectiveness of the proposed capacity management process. Our results show that when consolidating to 8 processor systems, we predicted future per-server required capacity to within one processor 95% of the time. The approach enabled a 35% reduction in processor usage as compared to today's current best practice for workload placement.*

1 Introduction

In the distant past data centers were made up of small numbers of large mainframe computers that each hosted several application workloads with many users. Capacity planning experts helped to ensure that sufficient aggregate capacity was available just in time, as it was needed. With the advent of distributed computing new application workloads were typically assigned to their own smaller servers. The incremental cost of capacity from smaller servers was much less expensive than the incremental cost of capacity on mainframes. Capacity planners would often anticipate an application's workload demands two years in advance and pre-provision a new server with sufficient capacity so that the workload could grow into it. However, the explosive growth in both enterprise computing and Internet computing has led to server sprawl in data centers. Enterprise data centers are typically full of large numbers of lightly utilized servers that incur high cost of ownership including facilities cost, such as rent and power for computing and cooling, high software licensing cost, and high cost for human management activities.

Web services and service oriented computing further exacerbate the issues of server sprawl. In a service oriented environment there may be many services to manage either as workloads that run on servers or as services that share an operating system image. In our context, the services are assumed to be of coarse granularity, e. g., a database service that contributes to a customer relationship management service that supports many end users. We consider the resource demands of each service to correspond to an application workload.

Many enterprises are now beginning to exploit resource pools of servers supported by virtualization mechanisms that enable multiple application workloads to be hosted on each server. The primary motivation for enterprises to adopt such technologies is increased flexibility, the ability to quickly repurpose server capacity to better meet the needs of application workload owners, and to reduce overall costs of ownership. Unfortunately, the complexity of these environments presents additional management challenges. There are many workloads, a finite number can be hosted by each server, and each workload has capacity requirements that may frequently change based on business needs. Capacity management methods are not yet available to manage such pools in a cost effective manner.

The goal of our work is to provide a capacity management process for resource pools that lets capacity planners match supply and demand for resource capacity in a just in time manner. When managing resource pools there are numerous *capacity management questions* that must be answered to ensure that 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?

We use a trace based approach to address these capacity management questions. It relies on *i)* a definition for required capacity, *ii)* the characterization of workload demand patterns, *iii)* the generation of synthetic workloads that predict future demands based on the patterns, and *iv)* a workload placement recommendation service. Our process automates data gathering and analysis steps that address these questions. As a result it enables human operators to handle the questions more quickly and accurately with lower labor costs.

To demonstrate the effectiveness of our proposed capacity management approach, we obtained six months of data from an enterprise data center. The data describes the time varying demands of 139 enterprise applications. We demonstrate the impact of the various definitions for required capacity and demonstrate the effectiveness of the demand prediction service. The results show that when consolidating to 8 processor systems, we predicted per-server required capacity to within one processor 95% of the time when predicting per-server required capacity 5 weeks into the future while enabling a 35% reduction in processor usage as compared to today's current best practice for workload placement. The remainder of the paper presents our results in more detail.

2 Capacity Management Process

This section describes the capacity management process we envision and its corresponding services. The process relies on a combination of sub-processes that implement various use cases for resource pool operators. Examples of use cases include:

- determine resource pool capacity needed to support a number of workloads;
- add/remove a workload to a resource pool;
- add/remove capacity to a resource pool;
- rebalance workloads across resources in a pool;
- reduce load on a server resource in a pool by recommending new workload placements for some of its workloads;
- report significant changes in workload demand behaviors; and,
- adjust per-workload forecasts, trends or quality of service requirements.

To support such use cases we must start with a definition of required capacity. *Required capacity* is the minimum amount of capacity needed to satisfy resource demands for workloads on a server resource. Section 3 gives a more formal definition and motivates our particular method for expressing required capacity. Given a definition for required capacity, we implement

- an admission control service,
- a workload placement service, and
- a workload demand prediction service.

The *admission control service* decides whether a resource pool has sufficient resources to host a new workload. If so it reports which server the workload should be assigned to. We consider workloads that exploit multiple resources as a collection of individual workloads possibly having workload placement constraints that must be addressed by the workload placement service.

The *workload placement service* we employ [13] recommends where to place application workloads among servers in the pool to reduce the number of servers used or to balance workloads across the servers. The service implements a trace based approach for characterizing resource demands and for recommending solutions. Basically, each workload is characterized using a time varying trace of demands for its key capacity attributes such as processor usage and memory usage. The workload placement service includes greedy algorithms for consolidating workloads onto a small set of servers and for balancing the workloads across some fixed number of servers. It also includes a genetic algorithm based optimizing search that aims to improve upon the greedy solutions. In each case the algorithms simulate multiple assignment scenarios. Each scenario considers the placement of zero or more workloads on each server. The aggregate demand of the workloads assigned to a server is characterized using a trace that is the sum of its per-workload time varying demands. The required capacity of the time varying aggregate demands is compared with the capacity of the resource to decide whether the workloads fit. The service recommends the best workload placement it can find over all servers, either for consolidation or for load leveling, that fits. Finally, *the service accepts additional constraints* on workload placements that include affinity between workloads, e.g., workloads must or must not be placed on the same physical server, and affinity between workloads and a list of one or more specific servers.

The *workload demand prediction service* has three purposes:

- it helps to recognize whether a workload's demands change significantly over time;
- it supports the generation of synthetic traces that represent future demands for each workload to support capacity planning exercises; and,
- it provides a convenient model that can be used to support forecasting exercises. The service implements pattern discovery techniques that we describe in Section 4.

The capacity management process relies on the key concept of a *capacity management plan*. A capacity management plan is a calendar based data store that keeps track of: workload identities, forecasts, and resource access quality of service requirements; resources that are associated with a pool; and assignments of workloads to resources. As a calendar based data store, the plan keeps track of such information as a function of date and time and uses it to support capacity planning.

Figures 1 (a) - (c) show several aspects of our capacity management processes. "Configure resource pool size" is used to reduce capacity fragmentation by periodically repacking, i.e., consolidating, workloads in a pool. "Find placement" balances loads, i.e., divides load evenly, across resources. It has two stages. If no resource is able to support the resulting capacity requirements then we may either attempt a larger scale re-balancing of workloads, adjust workload quality of service requirements, or combine the two approaches. "Add workload" reports whether a placement can be found for a new workload.

To summarize, the capacity management process we envision relies on the subprocesses we have described. Some

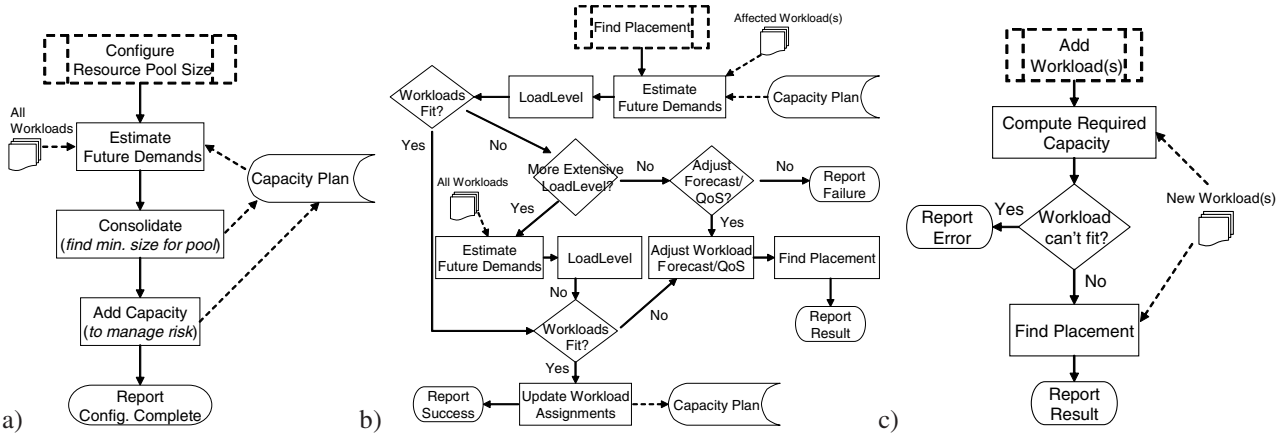

Figure 1. Examples of Capacity Management Processes: a) Configure Resource Pool Size; b) Find Placement; c) Add Workload.

of the steps may require resource pool operator or workload owner intervention or may be policy driven. We expect such processes to support a greater Information Technology service delivery framework [10].

3 Required Capacity

A definition for required capacity manages the level of overbooking in the resource pool, i. e., the extent to which demand for capacity is deemed to be permitted to exceed the supply of capacity. Demands may exceed supply if we place multiple workloads on a common server and the sum of per-workload peak demands exceeds the capacity of the server. Workload placement algorithms use a definition for required capacity to decide whether a set of workloads fit on a resource. They are deemed to fit if the resource has sufficient capacity across all attributes to support the aggregate workload demands.

Our definitions rely on the concept of a *unit of capacity*. The magnitude of a unit is somewhat arbitrary, but it must be used in a consistent manner to express resource demands and the capacity of resources. For processors, we define a unit of capacity as one percentage of utilization of the processor.[1] A server with n-processors would have n hundred units of capacity. For memory, we define each unit of capacity as 1 MB. Similarly, we can define units of demand for other capacity attributes. We define utilization as demand divided by supply over some time interval.

Consider one capacity attribute. We define a workload's trace of demands, L, as N contiguously measured demand values for the attribute for intervals of constant duration d. Let t_n be the time that corresponds to interval n in L and let $l(t_n)$ be the measured demand in units of capacity. We can write $L = (l(t_n))_{1 \le n \le N}$. Our definitions for required capacity rely on fixed and sliding windows of intervals.

We define *a fixed window trace* with c contiguous intervals per window with window duration $s = c \cdot d$ as $L^{F_s} = \left(l^{F_s}(t_n)\right)_{1 \le n \le N/c}$, where $l^{F_s}(t_n)$ is the average demand over the window of duration s.

We define *a sliding window trace* with c contiguous intervals per window of window duration $s = c \cdot d$ as $L^{S_s} = \left(l^{S_s}(t_n)\right)_{1 \le n \le N-c+1}$, where $l^{S_s}(t_n)$ is the average demand over the windows of duration s.

3.1 Fixed Windows and Probabilities

Fixed windows provide an intuitive way to express constraints on required capacity. With this approach we may state multiple simultaneous constraints for fixed windows with different sizes. Consider Z constraints, of the form (s_i, U_i, P_i), for $i = 1 \ldots Z$, where:

- s_i, a fixed window with c_i intervals of duration d so that $s_i = c_i \cdot d$,
- U_i, a limit on the percentage of utilization of capacity for a window,
- P_i, the percentage of windows permitted to have utilizations that exceed U_i.

We solve for a required capacity such that the tightest constraint is satisfied. For example, let: $s_0 = 30$ minutes, $U_0 = 100\%$, and $P_0 = 100\%$; and $s_1 = 5$ minutes, $U_1 = 100\%$, and $P_1 = 95\%$. The first constraint captures the intuitive requirement that demand for capacity should not exceed supply for too long, e.g., 30 minutes. The second constraint limits how often demand is permitted to exceed supply at a shorter timescale, e.g., 5 minutes. This limits the impact of overbooking on application workloads at shorter timescales.

A deficiency of this approach is that it does not clearly bound the impact on any demand that is not satisfied in an interval. For example, for those 5 minute intervals where demand exceeds supply we don't know how much greater demand was than supply. Furthermore, as a fixed window approach, the result for required capacity will depend on which interval starts the first fixed window. We now consider a sliding window approach.

3.2 Simple Sliding Window

Our *simple sliding window* definition for required capacity defines the required capacity for a capacity attribute as the minimum number of units of capacity needed so that demand for capacity doesn't exceed the supply of capacity

[1] We don't discuss the scaling of capacity requirements between servers with different processor speeds or architectures in this paper.

for more than an *overload epoch s* as expressed in minutes. If a unit of demand is not satisfied because demand exceeds supply, i.e., an overload, then that unit of demand propagates forward in time until there is available capacity to satisfy the demand. For a performance critical environment s may be chosen as 0, which means all demand must always be satisfied. For a more typical data center, where service levels may be monitored on an hourly basis, $s = 30$ minutes may be a reasonable value. We report the required capacity as the smallest capacity such that no epoch has demand greater than supply for more than s minutes at a time. This is a sliding window approach where the overload epoch is defined as the window duration s.

3.3 Per-Unit-of-Demand Sliding Window

We now consider our preferred definition for required capacity that also uses the sliding window. With this approach, we find the smallest capacity such that no unit of demand is propagated from one interval to the next for more than s minutes. Additionally, we specify a *resource access probability* θ that a unit of demand will be satisfied upon demand, and hence not propagated. This directly relates the definition of required capacity to its impact on workload demands.

Consider the computation of θ that takes place with the per-unit-demand approach during workload placement exercises. Let A be the number of workload traces under consideration. Each trace has W weeks of observations with T intervals per day as measured every d minutes. Without loss of generality, we use the notion of a *week* as a time period for service level agreements. Other time periods could also be used. Time of day captures the diurnal nature of interactive enterprise workloads (e.g., those used directly by end users); we note that some time intervals may be more heavily loaded than others. For 5 minute measurement intervals we have $T = 288$ intervals per day. We denote each interval using an index $1 \leq t \leq T$. Each day x of the seven days of the week has an observation for each interval t. Each observation has a measured value for each of the capacity attributes considered in the analysis.

To define θ, consider one attribute that has a capacity limit of R 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 interval t. We define the measured value for θ as follows.

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

Thus, θ is reported as the minimum resource access probability received any week for any of the T intervals per day.

Furthermore, let R' be the required capacity for an attribute. The required capacity R' is the smallest capacity value, $R' \leq R$, to offer a probability θ' such that $\theta' \geq \theta$ and those demands that are not satisfied upon request, $D_{w,x,t} - R' > 0$, are satisfied within the overload epoch of s minutes.

The primary advantage of this approach over the fixed window approach is that demand not satisfied within an interval is modeled as propagated to the next interval. Compared to the simple sliding window approach, the overload conditions are defined with respect to units of demand that are not satisfied rather than a simple number of contiguous overload epochs. It has been shown that such a value for θ can be used to decide workload manager scheduler settings for workload managers that support two priorities of service [6]. The approach can be used to automatically partition a workload's demands across the two scheduling priorities to manage the risks of resource sharing on a per-workload basis. The higher priority can be used as a guaranteed class of service and the lower priority as a class of service that offers capacity with a statistical guarantee of θ.

4 Workload Demand Prediction

This section describes the techniques we use for the workload demand prediction service. Section 4.1 shows the extraction of workload patterns and trends. These are used to generate synthetic workload traces in Section 4.2.

4.1 Extracting Workload Patterns and Trends

We use a three stage approach to recognize a likely pattern for a workload. In the *analysis phase*, many hypothetical patterns are found. In the second phase, *trends* are computed. Finally, in the third phase, the hypothetical patterns are evaluated and a recommendation is made regarding the most likely pattern for the workload. The recommendation may be that the workload is periodic with a certain cycle time or a-periodic such that no clear cycle time was found.

Given a historic workload trace $L = (l(t_n))_{1 \leq n \leq N}$ which is represented by N contiguous demand values $l(t_n)$ we extract a demand pattern $P = (p(t_m))_{1 \leq m \leq M, M \leq N/2}$ with M contiguous demand values $p(t_m)$ with the assumption that the workload has a cyclic behavior. This assumption is evaluated later in the classification phase. According to a classical additive component model, a time series consists of a trend component, a cyclical component, and a remainder, e. g., characterizing the influence of noise. The trend is a monotonic function, modeling an overall upward or downward change in demand.

To identify the cyclical component that describes the periodic characteristics of the workload we determine the yet unknown duration M of the pattern. For this, we make a combined evaluation of the *periodogram function* and the *auto-correlation* [4] and, thus, determine a set of hypothetical pattern durations. Workloads from enterprise data centers typically show a periodicity which is a multiple of hours, days, weeks, and so forth. Due to unavoidable computational inaccuracies and influences of irregular events and noise, the wavelengths can diverge slightly from these typical periods. Thus, we perform a comparison to calendar specific periods and determine for every wavelength candidate the best matching multiple of hours, days, and weeks. After that, we select the best candidate wavelength λ' from the set of wavelength candidates. For more details on the calculation of wavelength candidates and the

selection of the best candidate we refer to [17]. The pattern length M is defined as λ'/d intervals where d is the duration of each interval.

The chosen value for the pattern length of M intervals is used to calculate the pattern $P = (p(t_m))_{1 \le m \le M}$ for the workload. First we define occurrences for the pattern and then define the pattern's demand values $p(t_m)$. Given M, we divide the workload L into N/M complete occurrences and possibly one partial occurrence. Let O be the occurrences of the pattern for $o \le N/M+1$. Thus, occurrence o is a subtrace of the trace L with values $l^o(t_m) = l(t_{m+o \cdot M})$ for each $1 \le m \le M$. For every interval t_m in the pattern we calculate a weighted average $p(t_m)$ for the interval. The weighted average is computed using intervals t_m from the occurrences O of the pattern. We define a weight for each occurrence o and interval m as:

$$w_{o,m} = \frac{l^o(t_m)}{\Sigma_o l^o(t_m)}$$

With these weights we compute the weighted average demand for each interval t_m as $p(t_m) = \sum_o w_{o,m} \cdot l^o(t_m)$. We use the weighted average to emphasize the importance of larger values over smaller values for capacity management.

In the second phase, we analyze the trend of the workload. For this we calculate the overall deviation of each occurrence of the pattern from the original workload L. Let c_m^o be the difference between the $p(t_m)$ and the demand value for interval t_m in the occurrence o. We define c^o as the aggregate demand difference of occurrence o with respect to the pattern P as: $c^o = \sum_{1 \le m \le M}(p(t_m) - l^o(t_m))$. Further, we define the trend τ as the gradient of the linear least squares fit [7] through the values c^o for the occurrences O as ordered by time. The trend τ estimates the rate of change of demand over time with respect to the pattern.

Finally, in the classification phase we decide which workloads exhibit periodic behavior. The classification is based on two measures for the quality of the pattern. The first measure is $\bar{\rho}'$ that is the average value at multiples of the λ' in the auto-correlation function. Larger values for $\bar{\rho}'$ imply a better quality of fit. The second measure characterizes the difference between occurrences O and the pattern. The difference is computed as the average absolute error $\zeta = \frac{\sum_{1 \le m \le M, o} |p(t_m) - l^o(t_m)|}{N}$ between the original workload and the pattern P. Smaller differences suggest a better quality of pattern. To classify the quality of patterns for a large number of workloads, we employ a *k means cluster algorithm* [9] with clustering attributes ζ and $\bar{\rho}'$. The algorithm partitions the patterns into three groups that we interpret as having strong, medium, or weak patterns. Weak patterns are regarded as a-periodic because no clear cycle could be deduced for the trace.

4.2 Generating Synthetic Workload Traces and Forecasting

We now consider a process for generating a synthetic trace to represent a future workload demand trace L' for some time period in the future. Typically, we generate traces to represent demands for a time period that is several weeks or months into the future. Our goal for a synthetic trace is to capture the highs and lows of demand and contiguous sequences of demand. These are critical for modeling a workload's ability to share resource capacity with other workloads and to model required capacity for the workload.

To generate an occurrence o' for L' we rely on the historical pattern occurrences O. A value $l^{o'}(t_m)$ is chosen randomly from the corresponding t_m values from O. Given a sufficiently large number of future occurrences O', we will obtain the same time varying distribution of demands as in O. This gives us a pattern of demands that captures the lows and highs of demand in a representative way. To better model required capacity we must take into account sequences of contiguous demands in the trace L. We accomplish this by randomly selecting blocks of b intervals $t_m, t_{m+1}, \ldots, t_{m+b}$ at a time from the occurrences O. Furthermore, we note that the occurrences may have a trend τ. For the sequence of historical pattern occurrences we normalize the demand values so that the trend is removed with respect to the last occurrence before constructing O'.

Demands $l^{o'}(t_m)$ in the synthetic trace are augmented to reflect the trend τ. We assume an additive model. For each future occurrence o', we compute an absolute value based on τ that must be added to each demand in occurrence o'. The further o' is into the future the greater the change with respect to the historical data, assuming τ is not zero.

In our capacity management process, we repeat our analysis steps using multiple randomly generated instances of L' to better characterize the range of potential behavior for the overall system. Multiple instances better characterize interactions in demands among multiple workloads.

Finally, a workload pattern P provides a convenient way to express what-if-scenarios and business forecasts that are not observable to us from historic data. Suppose we have a pattern P with O occurrences and we require a change to the pattern. Then, we can express a change once with respect to P rather than once for each of the possibly many occurrences.

5 Case Study

To evaluate the effectiveness of our methods and processes we obtained six months of workload trace data for 139 workloads from a data center. The data center specializes in hosting enterprise applications such as customer relationship management services and supply chain management services for small and medium sized businesses. Each workload was hosted on its own server so we use resource demand measurements for a server to characterize the workload's demand trace. The measurements were originally recorded using vmstat and sar. Each trace describes resource usage, e. g., processor demands, as measured every 5 minutes starting January 1st, 2006.

Our case study considers:

- the impact of alternative definitions for required capacity;
- general results for the data center using results from workload demand pattern analysis method;
- a validation of the trending and synthetic workload generation techniques; and

Workload	Simple Sliding Window	100-p	99-p	95-p	Fixed window	Per-unit-demand sliding window
w_1	848	946	769	621	855	839
w_2	538	580	402	296	553	533
w_3	75	75	75	74.7	75	75
w_4	32	51.6	40.4	9.2	36	22
$w_{1\cdots4}$	1193	1356	1036	827	1240	1164
w_5	464	539	406	317	449	444
w_6	180	180	180	179	180	180
w_7	41	47	38.7	26.1	40	40
w_8	77	120	62	51	81	70
$w_{5\cdots8}$	581	672	528	431	607	558

Table 1. Required Capacity in Units of Demand

- a walk-forward test that employs the pattern matching, trending, and synthetic workload generation methods.

5.1 Required Capacity

We now compare the following methods for computing a required capacity as described in Section 3:

- simple sliding window definition with overload epoch of $s = 30$ min;
- 100, 99 and 95-percentiles of demand (as estimates for required capacity);
- a fixed window computation with a single constraint $U = 100\%$, $P = 100\%$ and $s = 30$ minutes; and,
- a per-unit-demand sliding window computation with $s = 30$ min.

For the per-unit-demand technique we do not constrain the value of the resource access probability θ. We include the percentiles of demand to explore the impact of the overload epoch.

Table 1 gives results for a five week subset of data for eight workloads, $w_1 \cdots w_8$, and two aggregate workloads $w_{1\cdots4}$ and $w_{5\cdots8}$ that include workloads $w_1 \cdots w_4$ and $w_5 \cdots w_8$, respectively. We consider aggregate workloads as well because we compute the required capacity for aggregate workloads during workload placement recommendation exercises.

The second column of the table gives the results for the simple sliding window that propagates unsatisfied demands.

The 100-percentile is the peak demand of a workload, it is always larger than the other estimates. While the 99 and 95 percentiles are often low compared to the sliding or fixed window approaches they provide no insight regarding the duration of any resulting overload epochs. The percentile approach typically underestimates required capacity, and as a result, it can not offer a controlled level of overload.

The fixed window approach typically results in higher required capacity values, i. e., more over-provisioning, compared to sliding window definitions. This is because the fixed window is more constraining than the sliding window; when $P = 100\%$ all demands must be satisfied within each fixed window. Neither the percentile nor the fixed window approach characterizes the impact of overload epochs on units of demand. However, such a characterization is helpful and necessary when deciding workload manager scheduler settings [6].

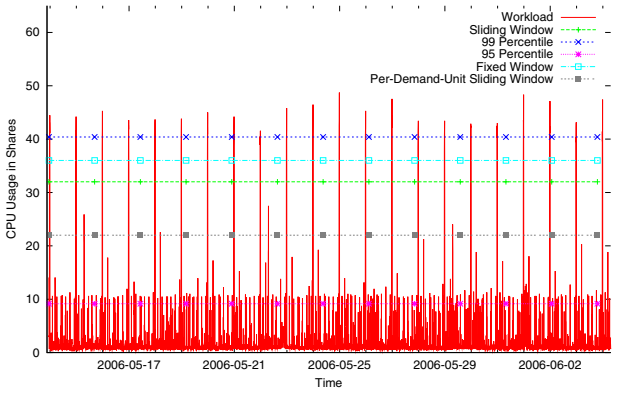

Figure 2. Workload and Required Capacity

As expected, the per-demand-unit sliding window approach typically results in a slightly lower required capacity value, i. e., more efficient capacity usage, than the simple sliding window definition as it permits longer periods of overload as long as no unit of demand remains unsatisfied after $s = 30$ minutes. Furthermore, we can observe a value for θ that is typical for workload placements in the data center. The resource pool operator can offer that as part of a service level commitment for the resource pool.

For workload w_4 the per-demand-unit approach results in a significantly lower estimate for required capacity, i. e., more efficient user capacity. This particular workload is very bursty, which is also shown by the relatively large difference between its 99 and 95 percentiles of demand. Figure 2 illustrates the computed required capacity values for w_4 with respect to the workload demand trace. It is interesting that the range of results for required capacity for the various methods is so large. The per-unit-demand sliding window would clearly permit longer overload epochs than the than the simple sliding window approach for this bursty workload. This may be acceptable if the clearly cyclic bursts in demand correspond to batch jobs and if the elapsed time requirements for the batch jobs can tolerate the additional delay. If the resource pool has workload managers that can offer multiple classes of service then a workload owner can choose a higher class of service for those workloads with demands that should not be deferred [6]. Due to these additional performance advantages and flexibility we adopt the per-unit-demand definition for required capacity and use it for the remainder of the case study.

5.2 Walk-Forward Test

In this section, we exploit the workload demand prediction service as part of the capacity management process. We conduct a walk-forward test over the six months of data to emulate how well our capacity management process would have served the data center for the six months.

- Starting with the first week, a window with w weeks of data is used to recommend a consolidated configu-

ration C_1, i.e., each workload is assigned to a specific server, for the system. The configuration reports expected required capacity values for each server in the configuration.

- The next y weeks of data are then simulated with respect to C_1. This simulation gives the actual required capacity for the next y weeks.
- The difference between a server's estimated and actual required capacity gives the absolute error for the estimate of required capacity. The negative errors reflect "under-estimated" capacity while the positive errors correspond to "over-estimated" capacity. We use a special CDF that reflects both types of errors for the walk-forward test.
- The steps in the walk-forward test are repeated iteratively with w weeks of data but now starting with weeks 2, 3, and so on.
- Let i be the step number in the walk-forward test. Step i computes a new configuration C_i and a new set of differences between estimated and actual required capacity values for each server.

We consider an ongoing process where the workloads are repeatedly consolidated onto a number of powerful servers over time. The servers have 8 processors. In general, the consolidation required 13 or 15 of these servers at a time. To evaluate the effectiveness of workload demand prediction methods we consider several different scenarios for generating synthetic workloads. The scenarios include:

a) use pattern analysis and trending;
b) use pattern analysis alone;
c) all workloads are associated with daily pattern; and,
d) all workloads are associated with a 30 hour pattern (specifically chosen to be incorrect).

For our study we use $w = 5$ weeks of historic input for the process and predict required capacity $y = 1$ week and $y = 5$ weeks into the future. Figures 3 and 4 show CDFs of errors in predictions for required capacity for the scenarios over the entire walk-forward test. A negative error suggests that a method estimates less capacity than is actually required for a server.

Figure 3 shows the results for the one week prediction. Scenarios *a)* and *b)* are pretty much indistinguishable. Trending avoided two large but similar negative errors. A fixed daily pattern without trending, scenario *c)*, caused several larger negative errors than *a)*, i. e., values less than -1 processor. The clearly incorrect 30 hour pattern from scenario *d)* caused severe errors.

Figure 4 shows that the results for predicting required capacity 5 weeks into the future are very similar. The only difference is errors were a little lower for scenario *b)*, i. e., without trending, than *a)* with trending. This is reasonable. Our historic window of 5 weeks of data is not likely to be sufficient for predicting trends 5 weeks into the future for all workloads for all steps in the walk-forward test.

For both 1 week and 5 week predictions, Scenario *a)* estimates per-server required capacity to within one processor (out of eight processors) 95% of the time.

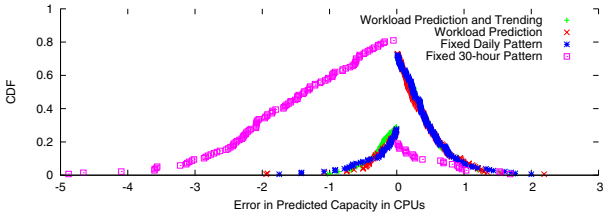

Figure 3. Predicting Capacity for 1 Week

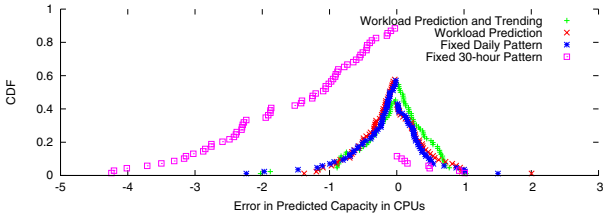

Figure 4. Predicting Capacity for 5 Weeks

The current best practice for server consolidation in industry determines the peak demand of each application workload and consolidates the applications to a small number of servers such that the sum of peak demands for applications at each server is less than the capacity of the server. If we choose to recommend workload placements based on peak per-application demands then we require 23 8-processor servers. By using the trace based method described in this paper we require only 15 servers. The approach enables a 35% reduction in processor usage as compared to a workload placement that stacks applications based only on their peak demands. The accuracy of predictions for required capacity suggests that such resource savings can be achieved with little risk.

6 Related Work

We employ a trace-based approach to model the sharing of resource capacity for resource pools. Many groups have applied trace-based methods for detailed performance evaluation of processor architectures [11]. They can also be used to support capacity management on more coarse data, e. g., resource usage as recorded every five minutes.

Some early work that evaluated data center efficiency relied on traces of workload demands to predict opportunities for resource sharing in enterprise data centers [1]. The demand prediction we consider predicts demands days, weeks, and months into the future. We distinguish the methods we employ from those that are typically used to predict demands several seconds or minutes into the future. Techniques for very short term predictions often use other approaches such as ARMA [4] or GARCH [8, 3] models. While these approaches may be appropriate for the very short term their predictions quickly converge to a mean value for the time scales of interest to us. [17] also describes methods for predicting workload demand patterns that exploit periodograms and auto-correlation. They are similar to the methods we propose, but do not consider trends, or synthetic workload generation as we developed in this paper.

Traces have been used to support what-if analysis that consider the assignment of workloads to consolidated servers. AOG [2] and TeamQuest [16] offer products that employ trace-based methods to support consolidation exercises. AutoGlobe [14] proposes a self-organizing infrastructure where the available hardware is virtualized, pooled, and monitored. They introduce a fuzzy logic based controller to supervise all services running on the hardware platform. If the controller recognizes an exceptional situation it triggers actions to remedy the situation automatically. In addition to that, they introduce a static optimization module that uses historical demand information to compute workload placement recommendations. They calculate the recommendation using a greedy heuristic.

We believe the workload placement service we employ has advantages over other workload placement services described above. It supports both consolidation and load balancing services as needed in a comprehensive capacity management process and is supported by a genetic algorithm that tends to improve upon greedy workload placement solutions. Furthermore, the workload placement methods go further than the other methods by addressing issues including classes of service and placement constraints.

Finally, the required capacity approach we present improves upon other approaches we have seen for resource pools. It provides for a resource access probability that can be used to partition each workload's demands across two classes of service [6]. Some researchers propose to limit the capacity requirement of an application workload to a percentile of its demand [15]. This does not take into account the impact of sustained performance degradation over time on user experience as our sliding window constraint does. Others look only at objectives for resources as a whole [14] rather than making it possible for each workload to have an independently specified objective.

7 Conclusions and Future Work

We describe a capacity management process for resource pools. The process relies on services that automate and simplify management for resource pool operators. We focused on definitions for required capacity and on a workload demand prediction technique. A case study exploited six months of data for 139 enterprise applications to evaluate the effectiveness of our methods. The automated methods predicted the required capacity of servers hosting the workloads to within one processor out of eight 95% of the time when predicting required capacity five weeks into the future while reducing aggregate processor requirements by 35% without significant risks. Such advance knowledge can help resource pool operators to decide whether to order additional capacity for their pools.

The workload demand prediction service relies on pattern and trend recognition methods. Our case study results show that trend prediction can be helpful as long as we do not exaggerate how far into the future we expect trends to continue. We believe that workload demand prediction methods are advantageous for a capacity management process. They recognize odd patterns which may interact in the future, e.g., 3 day and 5 day patterns may interact every 15 days, and can help to report when a workload's demands appear to deviate from past behavior.

Our future work includes: developing on-line automated methods for monitoring and reporting unplanned changes to workload characteristics; better exploiting notions of confidence and risk regarding predictions for future required capacity; and better integrating business forecasts for change into our approach. Further work is also needed to characterize workload demand patterns for enterprise services exposed as Web services. The usage patterns for these services may differ from the patterns characterized in this paper.

References

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

[2] http://www.aogtech.com/

[3] T. Bollerslev. Generalized Autoregressive Conditional Heteroskedasticity. *Journal of Econometrics*, 31:307–327, 1986.

[4] G. E. P. Box, G. Jenkins and G. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Upper Saddle River, NJ, USA, third edition, 1994.

[5] I. Chakravarti, R. Laha, J. Roy.Handbook of Methods of Applied Statistics. Vol. I, John Wiley and Sons, pp.392-394, 1967.

[6] L. Cherkasova and J. Rolia: R-Opus: A Composite Framework for Application Performability and QoS in Shared Resource Pools. Proc. of the Intl. Conf. on Dependable Systems and Networks (DSN'2006), Philadelphia, USA, 2006.

[7] N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley & Sons, New York, NY, USA 1998.

[8] R. F. Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, 50(4):987–1008, 1982.

[9] J .A. Hartigan and M. A. Wong. A K-Means Clustering Algorithm. In *Applied Statistics*, vol. 28, pp. 100–108, 1979.

[10] ITIL: IT Infrastructure Library. www.itil.com.uk/

[11] J. Pieper, A. Mellan, J. Paul, D. Thomas and F. Karim. High Level Cache Simulation for Heterogeneous Multiprocessors. Proc. of the 41st Annual Conference on Design Automation, San Diego, CA, 2004.

[12] J. Rolia, X. Zhu, M. Arlitt and A. Andrzejak, Statistical Service Assurances for Applications in Utility Grid Environments. Performance Evaluation Journal, vol. 58, 2004.

[13] J. Rolia, L. Cherkasova, M. Arlitt and A. Andrzejak. A Capacity Management Service for Resource Pools. Proc. of the 5th Intl. Workshop on Software and Performance (WOSP'2005), Spain, 2005.

[14] S. Seltzsam, D.Gmach, S. Krompass and A. Kemper. AutoGlobe: An Automatic Administration Concept for Service-Oriented Database Applications. Proc. of the 22nd Intl. Conference on Data Engineering (ICDE'2006), Industrial Track, Atlanta, GA, 2006.

[15] B. Urgaonkar, P. Shenoy and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. Proc. of the 5th Symp. on Operating System Design and Implementation (OSDI'2002), Boston, MA, December, 2002.

[16] http://www.teamQuest.com

[17] M. Wimmer, V. Nicolescu, D. Gmach, M. Mohr, A. Kemper and H. Krcmar. Evaluation of Adaptive Computing Concepts for Classical ERP Systems and Enterprise Services. *Proc. of the Joint Conference on E-Commerce Technology and Enterprise Computing, E-Commerce and E-Services (CEC'06 and EEE'06)*, San Francisco, CA, 2006.