Bounding the Resource Savings of Utility Computing Models

Artur Andrzejak, Martin Arlitt, Jerry Rolia
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2002-339
December 6th, 2002*

E-mail: Artur.Andrzejak@gmx.de, Martin.Arlitt@hp.com, Jerry.Rolia@hp.com

utility computing, resource savings, grid computing

In this paper we characterize resource usage in six data centers with approximately 1,000 servers. The results substantiate common knowledge that computing resources are typically under-utilized. Utility computing has been proposed as a way of increasing utilization and hence efficiency. Using an off-line integer programming model we bound the potential gain in efficiency for several different utility computing models. In a detailed study of a subset of the data center servers, we found that utility computing offered the potential to reduce the peak and mean number of CPUs needed by up to 53% and 79%, respectively.

* Internal Accession Date Only
Approved for External Publication

Bounding the Resource Savings of Utility Computing Models

Artur Andrzejak[*] Martin Arlitt Jerry Rolia

Internet Systems and Storage Laboratory

Hewlett Packard Laboratories

Palo Alto, CA 94304

Konrad-Zuse-Zentrum für Informationstechnik, Berlin (ZIB)

Takustraße 7, 14195 Berlin-Dahlem, Germany

Artur.Andrzejak@gmx.de, Martin.Arlitt@hp.com, Jerry.Rolia@hp.com

November 27, 2002

Abstract

In this paper we characterize resource usage in six data centers with approximately 1,000 servers. The results substantiate common knowledge that computing resources are typically under-utilized. Utility computing has been proposed as a way of increasing utilization and hence efficiency. Using an off-line integer programming model we bound the potential gain in efficiency for several different utility computing models. In a detailed study of a subset of the data center servers, we found that utility computing offered the potential to reduce the peak and mean number of CPUs needed by up to 53% and 79%, respectively.

[*]This work was done as Post-Doctoral research at Hewlett-Packard Laboratories.

1 Introduction

Data centers are large computing facilities with centralized resources and management infrastructure. They are often informally cited as having resource utilization rates of 20 to 30 percent. *Utility computing* has been proposed as a sharing mechanism to make more effective use of computing infrastructure. Typically computing resources are allocated to individual (or a small set of) applications and provisioned for anticipated peak-load conditions over multi-year planning horizons. This can lead to sustained periods where resources are under-utilized. Sharing resources across multiple applications can: increase asset utilization by keeping resources busy, improve business agility by making available resources on demand, decrease power consumption by shifting work from lightly loaded resources – and placing them in a power savings mode, and lower overall costs of ownership by reducing the quantity and space required for infrastructure and by increasing automation.

In this paper we study the load patterns of six data centers with approximately 1000 servers. We find that resource utilization levels for 80% of the servers were indeed in the 30% range. A subset of servers for one data center is studied in detail.

We describe several models of utility computing. Bounds for the potential resource savings of these utility computing models are given for the subset of servers of the data center under detailed study. The bounds are based on workload allocations found by an off-line integer programming model with a limited computation time. The allocations are not guaranteed to be optimal, yet are closer to optimal than are likely to be achieved in practice with an on-line algorithm. For the servers under study we found the potential to reduce the peak and mean number of CPUs by up to 53 and 79%, respectively. These represent significant resource savings and opportunities for increasing asset utilization with respect to the capacity that would be required by data centers without resource sharing.

The models of utility computing that we consider are summarized in Section 2. Section 3 discusses related work on utility computing. An analysis of loads for six data centers is presented in Section 4. The optimization methods used to bound the resource savings of the various models of utility computing are described in Section 5. The results of the analysis are presented in Section 6. Section 7 gives summary and concluding remarks.

2 Utility Computing Models

There are many approaches towards utility computing. Broadly they fall into two categories. The first is a *shared utility* model where server resources are exploited by multiple customer applications at the same time. The second is a more recent approach we define as a *full server utility* model where applications programmatically acquire and release servers as needed. A shared utility can act as an application to a full server utility. Inevitably both rely on network fabrics and possibly storage systems that are shared by

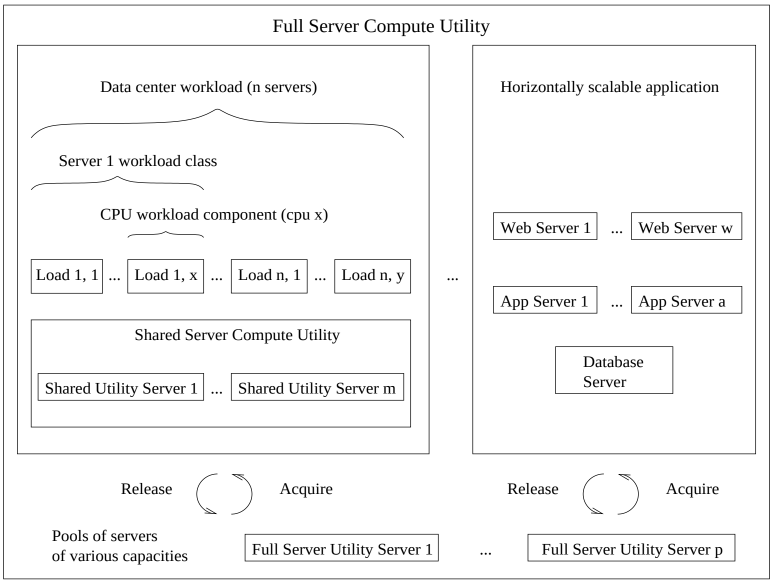


Figure 1: Shared and Full Server Models for Utility Computing

multiple applications. Figure 1 illustrates the relationships between these models.

There are several major differences between the two approaches. Security and performance interaction within a server are concerns for the shared utility approach but not the full server approach, because each server is only allocated to one application at a time. Application control systems are needed to acquire and release resources at appropriate times for the full server utility approach. Whereas utility (server cluster) scheduling mechanisms aim to achieve the same effect for the shared approach without the need for application control systems.

There are many kinds of application workloads that must be served by such utilities. We distinguish these as being either batch or business workloads. *Batch* workloads require resources for a certain amount of time, then terminate. Scientific applications for Grids are an example of batch workloads [7]. We loosely define *business* workloads as those that require resources continuously, but with resource requirements that vary over time. These are typically systems for enterprise resource management, customer relationship management, and store-fronts.

For business application systems we can further distinguish between those that are horizontally scalable and those that are not. *Horizontally scalable applications,* such as those that rely on clusters of Web and/or application servers as in Figure 1, are directly suited to the full server approach. Middleware for such applications is typically designed to have servers added/removed from clusters without interrupting the operation of the system. This feature enables scalability. Other applications may be less flexible. For example, adding or removing a server may require an application to be shut down and undergo a manual reconfiguration exercise. Some applications, or application tiers, may simply never require more than one multi-CPU server. The latter are most compatible with the shared utility model.

In utility computing environments *load migration* enables applications to move from one utility server to another. With operating system support, applications of a shared utility may be moved quickly from one server to another without any significant disruption to service [4]. A shared utility with load migration can aim to keep as small a number of servers busy as possible. It can then acquire and release servers as needed from a full server utility.

There is a substantial literature on the scheduling of batch jobs within Grids [7]. In this paper we focus on business workloads and attempt to bound potential resource savings when they operate in utility computing environments. Both shared and full server utilities are considered. The ability of full server utilities to offer servers on demand to shared utilities is considered when reporting bounds for mean numbers of CPUs.

3 Related work

A number of approaches have been proposed to increase the utilization of computing resources [1, 3, 4, 7, 9, 10]. Each offers a notion of utility computing where resources can be acquired and released when/where they are needed.

An example of a shared utility environment is MUSE [3] in which hosted web sites are treated as services. All services run concurrently on all servers in a cluster. A pricing/optimization model is used to determine the fraction of CPU resources allocated to each service on each server. A special dispatcher, a level 4 load balancer, routes web page requests for all services only to those servers where the service has a CPU allocation. The optimization model shifts load to use as few servers as possible while satisfying service level agreements (SLA) for the services. A major goal of the work is to maximize the number of unused servers so that they can be powered down to reduce energy consumption.

Other mechanisms for shared utilities include cluster reserves for multiple-service performance isolation [2] and QoS-differentiation algorithms [11, 12]. The Grid community provides infrastructure for the support of batch jobs in utility environments [7]. [4, 10] are examples of commercial shared utility products that support business application workloads. These partition the resources of clusters of shared servers with specialized scheduling systems that aim to satisfy SLAs while better supporting increased resource utilization.

A fast migration technology is described in [4]. It quickly swaps an application image to disk. The technology exploits a shim that intercepts all operating system calls. The shim, combined with some operating system kernel modifications, provides host transparency so that the image can be reinstated on a different server that runs the same operating system image. Integration with load balancers enables fast migration for non-trivial applications.

A full server utility data center is described in reference [9] and has been realized as a product [6]. It exploits the use of virtual LANs and SANs for partitioning resources into secure domains called virtual application environments. The environment supports multi-tier as well as single-tier applications and can also contain systems that internally implement a shared utility model. A second example of a full server utility approach is Oceano [1], an architecture for an e-business utility.

Ranjan *et al.* consider QoS driven server migration for horizontally scalable applications operating within full server utility environments [8]. They provide application level workload characterizations and explore the effectiveness of an online algorithm called Quality of Infrastructure on Demand (QuID). The algorithm is an example of an application control system that decides when to acquire and release resources from a full server utility.

4 Workload Characterization for Data Centers

The goal of our study is to compare the impact of various models of utility computing on resource utilization for data centers. For the purpose of this study we were able to obtain CPU utilization information for a collection of approximately 1000 servers. These servers were distributed among six data centers in three different continents. These are identified as data centers A through F. The data was collected between September 2, 2001 and October 24, 2001. For each server, the average CPU utilization across all CPUs in the server was reported for each five minute interval. This information was collected using MeasureWare (Openview Performance Agent).

We regard the per-interval product of the number of CPUs on a server and their reported aggregate average utilization as a measure of the server's demand on the data center. The aggregate demand of all servers is the *data center workload*. All applications hosted on a server contribute to its demand. We define this as a *workload class*; it is the workload for the server. In some experiments, we regard each CPU's worth of demand as separate *CPU workload component* of the server and data center workloads. We note that in these cases a server with x CPUs would be the source of x identical CPU workload components. Figure 1 illustrates examples of a data center workload, a workload class, and a CPU workload component.

Though our approach is only an approximation of how applications hosted on the servers would actually use a utility environment, it is one that allows us to demonstrate the impact of various assignment policies. For example, we consider the impact of assignment alternatives where CPU workload components of a server must either be allocated jointly to the same server in the utility, i.e. *workload affinity*, or can be allocated to different servers. With our approach, changes in utilization for the utility mirror changes in workload intensity for the data center. In the following sections we look for similarities in behaviour across data centers and patterns of behaviour within a data center.

Although the empirical data we collected contained a lot of interesting information, there is a lot of useful information that is not available to us. For example, we do not know the total number of servers in each data center, so we cannot determine what fraction of servers are included in our analysis. Furthermore, we do not have any information on the type of applications running on each server. Demand characterization on a per-application basis would enable a more detailed analysis than we can offer in this paper. We have

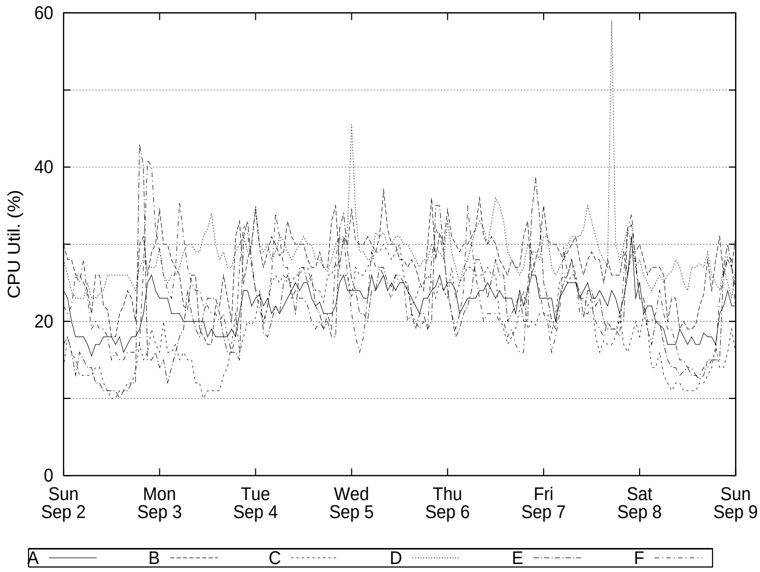


Figure 2: 80-percentile of CPU utilization Servers of Six Data Centers

no information about the hardware configuration of many of the servers. However, we were able to obtain detailed hardware configuration information for a subset of the servers from data center A. We refer to this subset as data center A'. We utilize this configuration information later in the paper.

4.1 Characterization of CPU Utilization Across Data Centers

Figure 2 shows a graph of server CPU utilization across the six data centers, labeled in the figure as A through F, for the week of September 2-9, 2001. The graphs for the other weeks show similar results, and are thus omitted. The curve for each data center represents the CPU utilization (averaged over one hour intervals) of the 80th percentile server from that data center. In other words, 80% of the servers in that data center have a CPU utilization (averaged over one hour intervals) less than or equal to the plotted curve for that data center. Figure 2 reveals that across all of the data centers, the 80th percentile is typically in the 15 to 35% range. These results suggest that from a utilization perspective alone, there are significant opportunities for resource sharing in data centers.

4.2 Characterization of CPU Utilization in a Single Data Center

For Figure 2 we computed the cumulative distribution function (cdf) for the CPU utilization of all servers in the data center over one hour periods. As a result, the curves are smoother than if we had calculated the cdf over shorter intervals (e.g., 5 minutes). Calculating the cdf over a one hour period made Figure 2 more legible, enabling an easier comparison of the utilization across data centers. However, using longer intervals can hide short intervals of higher CPU utilization.

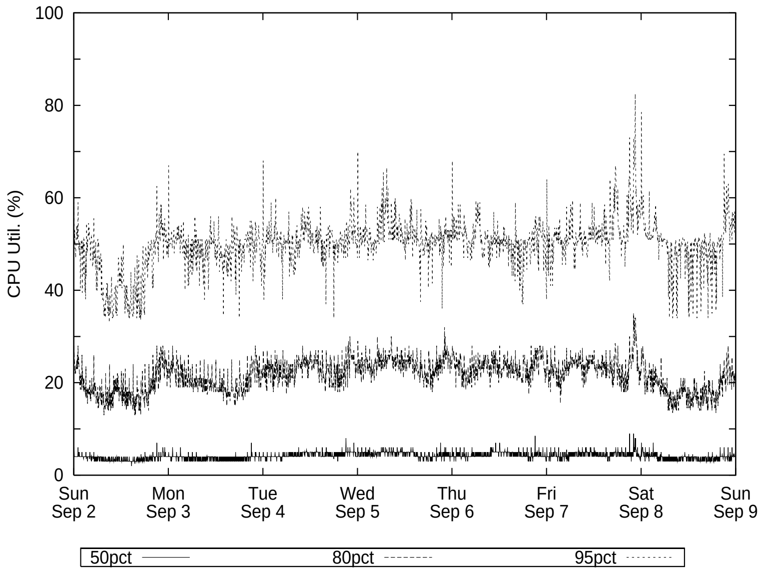


Figure 3: Detailed CPU Utilization for Data Center A

Figure 3 presents the CPU utilization information for the servers from Data Center A. In the figure, we show the cdf as computed over five minute intervals, the shortest possible interval length for the data set available to us. In Figure 3 three curves are present: the 50th percentile (the median), the 80th percentile, and the 95th percentile. The 50th percentile curve indicates that in any five minute interval half of the servers under study from Data Center A had an average CPU utilization of less than 10%. For this data center the 80th percentile is quite consistent in the 20% to 30% range. The 95th percentile curve indicates that not all servers in this data center have low utilizations; about 5% of servers in any five minute interval have an average CPU utilization in excess of 50%. These results indicate that care must be taken when attempting to share server resources, in order to prevent over-utilization. The results for the other data centers (not shown here) are similar to those for Data Center A.

Figure 4 shows the 80th percentile curves for the CPU utilization for Mondays in Data Center A over all 8 weeks of data. For this graph the 80th percentile has been computed over one hour intervals to make the graphs more legible. Several observations can be made from Figure 4. First, the CPU utilization behaviour is rather consistent over time; across all eight weeks of data the 80th percentile is typically between 20% and 30%. Similar observations can be made for the other days of the week, for this data center as well as the others (these additional graphs are not shown in the paper).

These results show that the workload intensity for these data center workloads appear to be consistent over time. However, we did not attempt to formally identify any patterns, such as increasing loads, over the weeks. We plan to investigate this as future work.

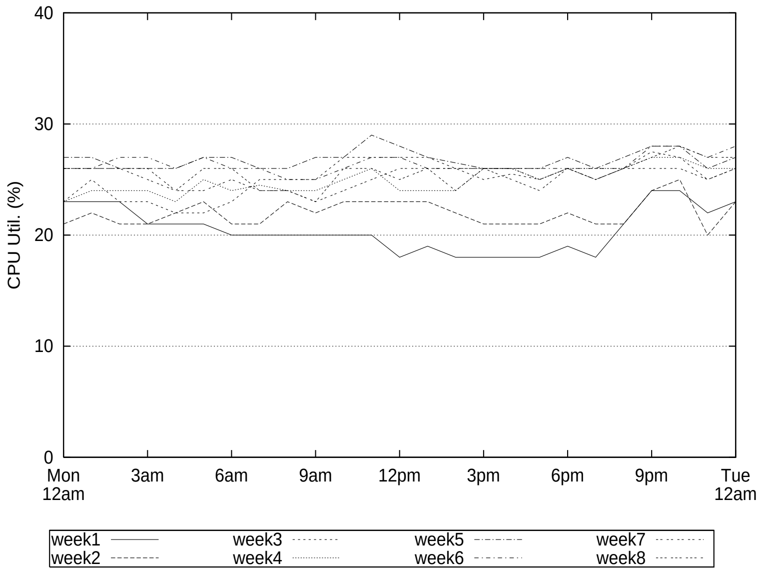


Figure 4: Day of Week (Monday), 80-percentile of CPU Utilization for servers for Data Center A

Group	Number of Servers	Server Class	CPU per Server	MHz	Memory (GB)
1	10	K-class	6	240	4
2	9	N-class	2-8	550	4-16
3	18	N-class	2-4	440	2-6
4	9	L-class	2-4	440	2-8

Table 1: Server Group Descriptions for Data Center A'

4.3 Selection of Servers for More Detailed Analyses

As was mentioned earlier, detailed hardware configuration information was available for a subset of servers in Data Center A. This information enabled us to create four groups of servers that had similar hardware configurations. We refer to these as data center A'.

Table 1 provides a breakdown of the four groups of servers that we use in Sections 5 and 6 for our examination of the different utility computing models. The most homogeneous is Group one, where all 10 servers have the same number (six) and speed (240 MHz) of CPUs, and the same amount of memory (4 GB). The other groups consist of servers from the same family with the same CPU speed, but vary in the number of CPUs and the amount of memory per server.

5 Bounding the Gain for Several Models of Utility Computing

This section describes our experimental design and the optimization methods used to assign the original workloads to servers in utility computing environments. In each case the goal of the optimization is to minimize the number of CPUs or servers needed per measurement interval or for the entire time under study. Our design explores the sensitivity of assignment to various factors. For example in some cases we assume that workload affinity is not necessary, even though it may be necessary, simply to explore the impact of affinity on assignment.

5.1 Experimental Design

We consider the following models for utility computing.

A. Large multi-processor server: we assume that the target server in the utility computing environment is a single server with a large number of CPUs (e.g. 32, 64). Each CPU workload component is assigned to a CPU for exactly one measurement interval. We assume processor affinity, i.e. the workload component cannot be served by any other CPU in this measurement interval. However, the CPU hosting a workload component is re-evaluated at the boundaries of measurement intervals.

B. Small servers without fast migration: here the target is a cluster of identical smaller servers with s CPUs each, where $s = 4, 6, 8, 16$. The workload classes, or CPU workload components, are assigned to the servers for the whole computation, but we assume no processor affinity, i.e. a workload might receive concurrent service from its server's CPUs.

C. Small servers with fast migration: this case is the same as case **B**, but the workload classes, or CPU workload components, are assigned to the target servers for a single measurement interval only. This implies that a workload class might migrate between servers at the measurement interval boundaries. We do not consider the overhead of migration in our comparison.

For the above cases, we consider several additional factors.

- *Target utilization M* per CPU. We assign the workloads to CPUs or servers in such a way that a fixed target utilization M of each CPU is not exceeded. The value of M is either 50% or 80%, so each CPU retains an *unused capacity* that provides a soft assurance for *quality of service*. Since our input data is averaged over 5-minutes intervals, the unused capacity supports the above-average peaks in demand within an interval. Figure 3 shows that only 5% of hosts had CPU utilizations greater than 50% for any five minute interval. If input utilization data exceeds the value of M for a measurement interval, it is truncated to M. Otherwise the mathematical programs can not be solved since no allocation is feasible. We note this truncation never occurs for the 16 CPU cases since none of the original servers had more

Factor	Symbol	Level	Applicable Cases
Target Utilization	M	50%, 80%	**A, B, C**
Interval Duration	D	5 min, 15 min	**A, B, C**
Workload affinity	-	true, false	**B, C**
Fast migration	-	true, false	**C**
# of CPUs per server	s	4, 6, 8, 16	**B, C**

Table 2: Summary of Factors and Levels

than 8 CPUs. When truncation is used, we assume that the unused CPU capacity accomodates the true capacity requirement.

- *Interval duration D.* In addition to the original data with measurement intervals of 5 minutes duration, we average the workload demands of three consecutive intervals for each original server, creating a new input data set with measurement interval length of 15 minutes. This illustrates the impact of measurement time scale and, for the case of fast migration, the impact of migrating workloads less frequently.

- *Workload affinity.* For cases **B** and **C**, we consider two scenarios: either all CPU workload components of a workload class must be placed on the same target server, or they may be spread among different servers. The first case, designated as *workload affinity*, makes sense for highly-coupled applications that have significant inter-process communication or that share memory. With workload affinity, CPU workload components behave as separate workloads.

- *Number s* of CPUs in each small server. For cases **B** and **C**, we assume that the number s of CPUs of a single server is 4, 6, 8 or 16. Note that if s is smaller than the number of CPUs of the original server and if we also assume workload affinity then certain cases are not feasible and are left out.

Table 2 summarizes the the factors and levels, and indicates when they apply.

5.2 Optimization methods

This section describes the integer programming model used to bound the resource savings of the various alternatives presented for utility computing.

For technical reasons, we introduce a notion of an *idle workload*. When the idle workload is assigned to a CPU or server then no *real* workload can be assigned to that CPU or server. The integer program as formulated uses the idle workload to force the real workload onto as small a set of CPUs or servers as possible. The demand of the idle workload is set to M for case **A** and $s \cdot M$ for cases **B** and **C**.

We designate:

$I = \{0, 1, \ldots\}$ as the index set of workload classes or CPU workload components (in the cases with workload affinity), where 0 is the index of the idle workload, these are the workloads under consideration,

$J = \{1, \ldots\}$ as the index set of CPUs (case **A**) or target servers (cases **B** and **C**),

$T = \{1, \ldots\}$ the index set of measurement intervals (with durations of either 5 or 15 minutes),

$u_{i,t}$ the demand of a workload with index $i \in I$ for measurement interval with index $t \in T$. As mentioned above, for all $t \in T$ we set $u_{0,t} = M$ for case **A** and $u_{0,t} = s \cdot M$ for cases **B** and **C**.

Furthermore, for $i \in I$ and $j \in J$ let $x_{i,j}$ be a 0/1-variable with the following interpretation: 1 indicates that the workload with index i is placed on a CPU or server with index j, 0 when no such placement is made. Note that $x_{0,j} = 1$, $j \in J$ indicates that the CPU or server with index j is hosting the idle workload, i.e. it is unused.

The following class of 0/1 integer programs cover our cases. The objective function is:

- Minimize the number of used CPUs (case **A**) or servers (cases **B** and **C**):

$$\text{minimize} \quad \sum_{j \in J} (1 - x_{0,j}).$$

With the following constraints:

- Let the unused CPUs (case **A**) or servers (cases **B** and **C**) be those with highest indices. For each $j \in J \backslash \{1\}$:

$$x_{0,j-1} \leq x_{0,j}.$$

- Each non-idle workload $i \in I$ is assigned to exactly one CPU (case **A**) or server (cases **B** and **C**). For all $i \in I \backslash \{0\}$:

$$\sum_{j \in J} x_{i,j} = 1.$$

- For case **A**, each target CPU must not exceed target utilization M in each time interval t. Each $t \in T$ gives rise to a new integer program to be solved separately. For all $j \in \mathbf{J}$:

$$\sum_{i \in \mathbf{I}} u_{i,t} x_{i,j} \leq M.$$

- For case **B**, each target server must not exceed target utilization M for each CPU for all measurement intervals $t \in T$ *simultaneously*. For all $j \in J$ *and* all $t \in T$:

$$\sum_{i \in \mathbf{I}} u_{i,t} x_{i,j} \leq sM.$$

- For case **C**, each target server must not exceed target utilization M for each CPU for each measurement interval t. Each $t \in T$ gives rise to a new integer program. For all $j \in J$:

$$\sum_{i \in \mathbf{I}} u_{i,t} x_{i,j} \leq sM.$$

As mentioned above, for cases **A** and **C** we solve one integer program for each measurement interval. Case **B** requires one integer program that includes all time intervals. For case **A** we have 16 experiments. Cases **B** and **C** each have 128 experiments.

The computations were performed on an 8-CPU server with 4 GB of memory running the AMPL 7.1.0 modeling environment and CPLEX 7.1.0 optimizer [5] under HP-UX. CPU time limits were established for each run of the CPLEX tool. For each run, we report the best solution found within its time limit. For scenarios in cases **A** and **C**, time limits were generally 128 seconds. Case **B** scenarios had limits of 4096 seconds. There were a few scenarios in **B** that did not yield good allocations. These were cases where we deemed the bounds for the solution, as reported by CPLEX, to be too wide. They then received 131072 seconds of computation time. Several case **C** scenarios received 2048 seconds of processing time. Unfortunately, we only had limited access to the modeling environment; some runs did not lead to satisfactory allocations.

6 Case Study

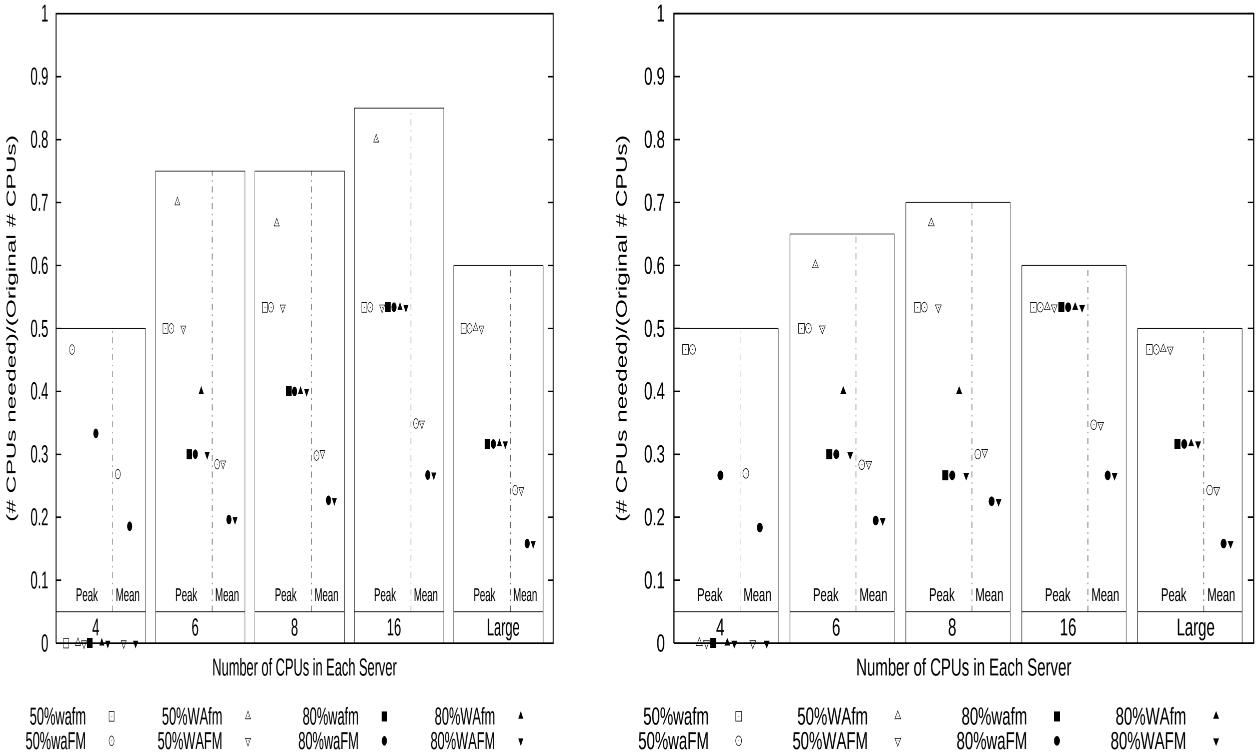
This section presents the results of the optimization method for the experimental design for data center A'. As stated earlier the optimization method is an off-line algorithm that attempts to find an optimal allocation of work. There is no guarantee that an optimal allocation is found; the quality of the solution depends on the time permitted for the computation.

Figures 5 through 8 illustrate the results for Groups one through four, respectively. Since the discussions of cases **A**, **B** and **C** rely directly on these figures, we first describe how to interpret the graphs.

The y-axis on each of the graphs illustrates the number of CPUs required for a particular scenario relative to the original number of CPUs in the group. For example, a data point at 0.5 on the y-axis means that 50% of the original CPUs in the group were required. Similarly, a data point at 1.2 on the y-axis would indicate that 20% more CPUs were required than in the original group of servers. Scenarios that were infeasible are plotted at zero on the y-axis.

Within each graph are five histogram bars (or columns). These columns are used to distinguish the cases with $s = 4, 6, 8$ and 16 CPUs, as well as the case for a large multi-processor system. Each column is divided into two parts with a dashed vertical line. On the left side of each column are the peak results; the right side of the column provides the mean results.

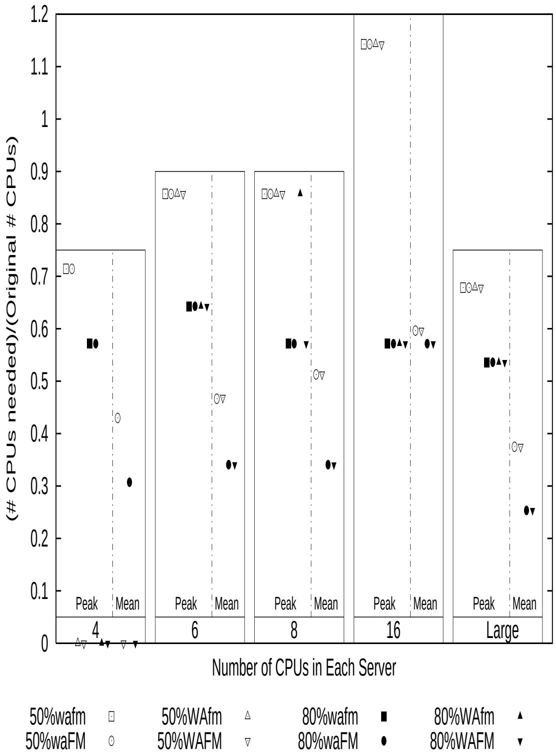
Within each column, data points are plotted for the eight different combinations of factors and levels. The workload affinity and fast migration factors are labeled as *WA* and *FM* when they apply, and *wa* and



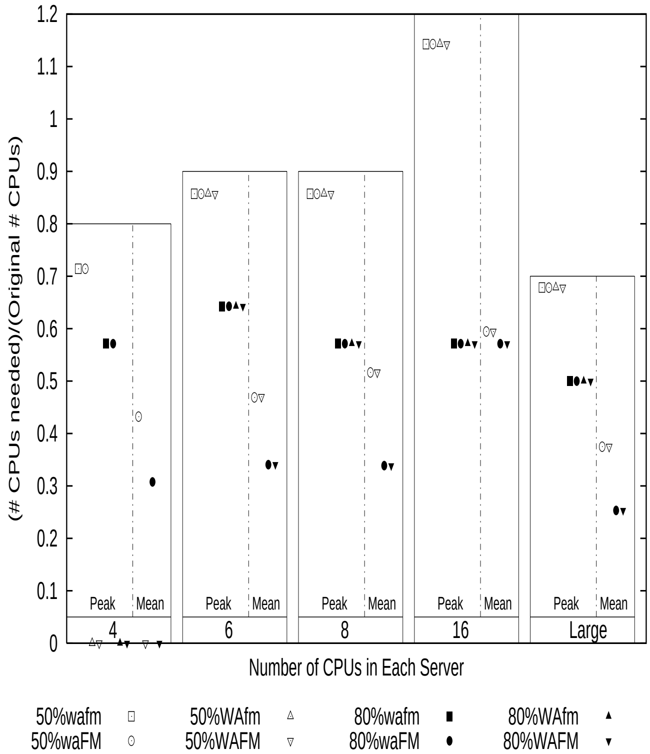
(a) $D = 5$

(b) $D = 15$

Figure 5: Group One Bounds on Number of Servers

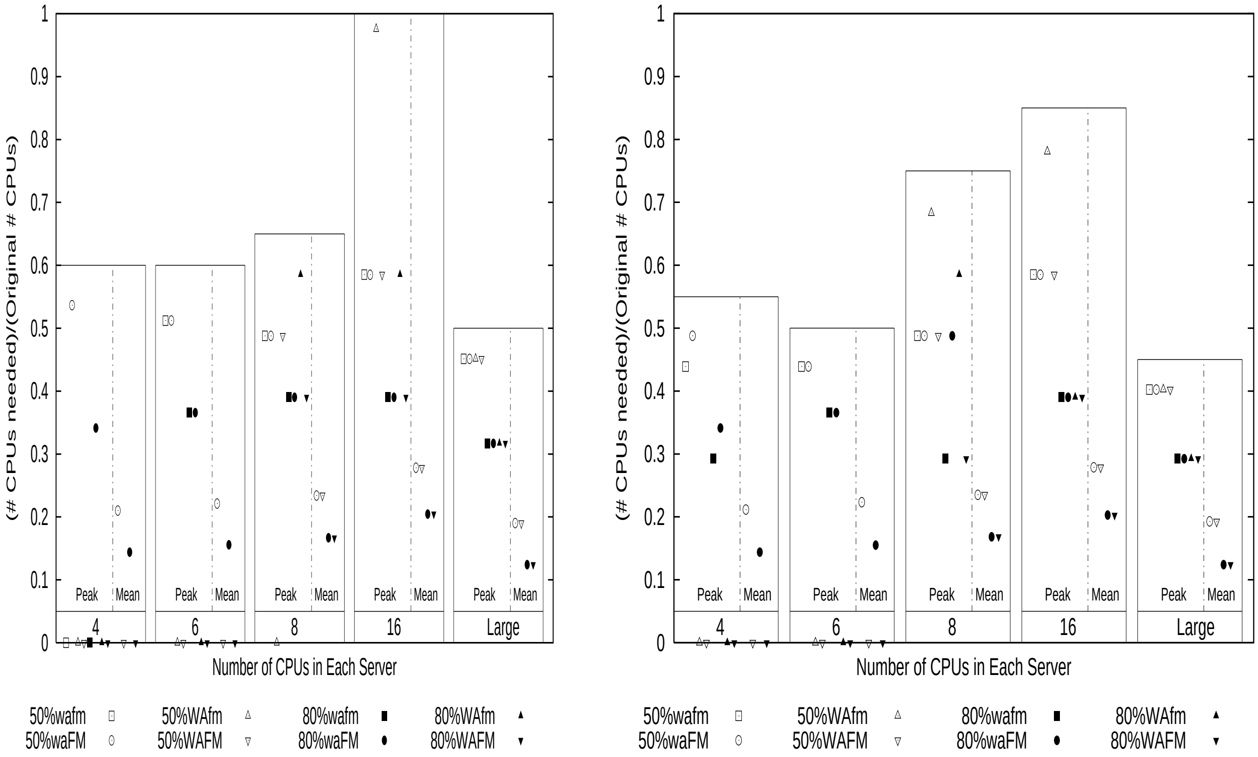


(a) $D = 5$



(b) $D = 15$

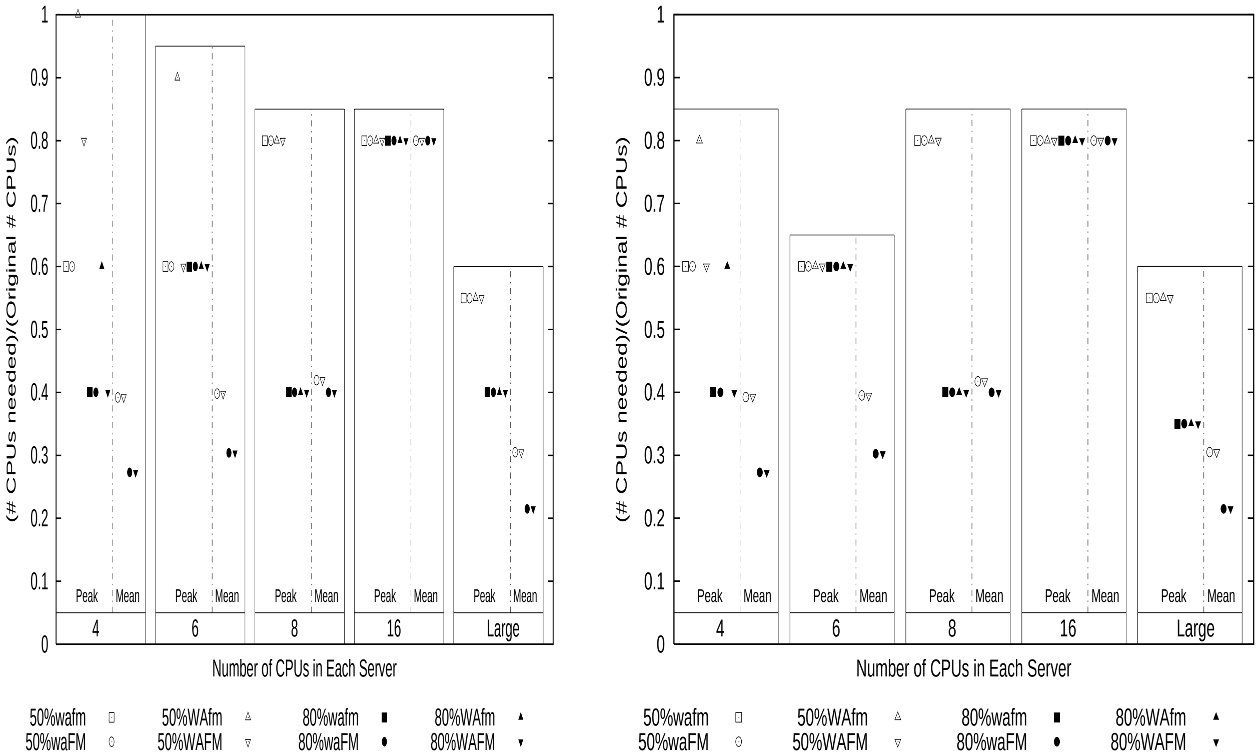
Figure 6: Group Two Bounds on Number of Servers



(a) $D = 5$

(b) $D = 15$

Figure 7: Group Three Bounds on Number of Servers



(a) $D = 5$

(b) $D = 15$

Figure 8: Group Four Bounds on Number of Servers

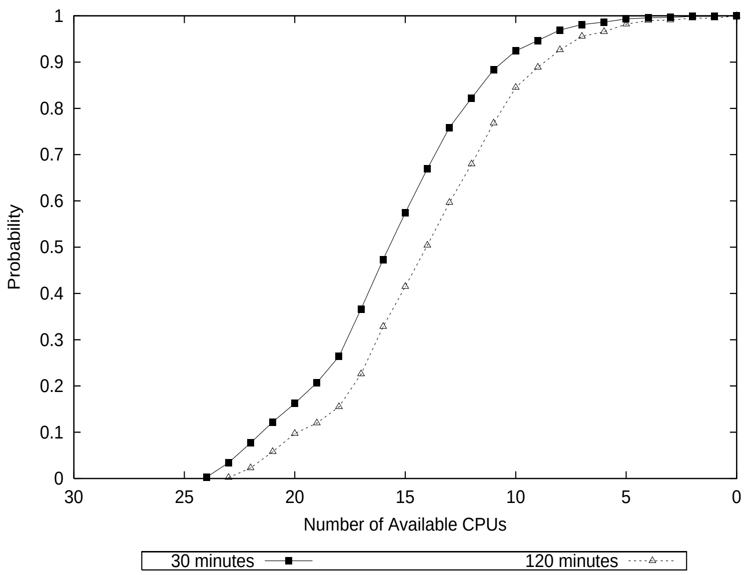


Figure 9: Remaining CPU Availability for Group one with $M = 50$ and $D = 5$

fm when they do not. The two levels for the target utilization M are 50% and 80%. The data points for experiments with a target utilization of 50% have an outlined point type (e.g., $\circ$) while experiments with a target utilization of 80% have a solid point type (e.g., $\bullet$). This means that in order to evaluate the effect of a change in M, it is only necessary to compare the outlined versus the solid point types of the same shape.

The labels in the legend of each figure describe the levels of each factor for a particular experiment. For example, the label $50\% waFM$ corresponds to the scenario of a 50% target utilization without workload affinity but with fast migration. Finally, the left hand graph in each figure corresponds to five minute measurement intervals (i.e., $D = 5$), while the right hand graph provides the results for aggregated 15 minute intervals ($D = 15$).

6.1 Case A

Case **A** allocates the work of each group to a large multi-processor system. In this scenario, CPUs may be re-allocated at the end of each measurement interval. We assume that each CPU has uniform access to all memory on the server. For this reason, workload affinity and fast migration do not affect the results for these cases. The purpose of this optimization is to estimate the peak and mean number of CPUs required. The results are presented in the *Large* column of Figures 5 through 8.

First we consider the case of the $M = 50\%$ utilization target. The results correspond to the outlined data points. For the five minute interval duration, $D = 5$, the peak number of CPUs for the four groups drops by 50%, 32%, 55%, and 45%, respectively. For $D = 15$, the peak number of CPUs required drops by 53%, 32%, 60%, and 45%, respectively. The results for these large server cases should give the lowest numbers of CPUs required over all our scenarios. These percentages identify potential resource savings, but must be realized by on-line scheduling algorithms. Thus, actual savings are likely to be less.

As expected when the target CPU utilization is increased to 80% (illustrated using the solid data points) the potential savings are greater still. This utilization target may be an appropriate target for non-interactive workloads where latency is not a significant issue. For this case, for Groups one through four, the savings range from 46% to 68% for $D = 5$ and from 50% to 71% for $D = 15$.

Our results indicate that increasing D tends to reduce the peak number of CPUs needed. This is because of a smoothing of *observed* bursts in application demand, since we only consider utilization as measured over the longer time scale. As a side effect, this may reduce the quality of service offered to applications. Inversely, increasing D may increase the mean number of CPUs required. For example, we observe that Group three had a small increase in the mean number of CPUs required at the 50% utilization level. The mean increased from 15.2 to 15.8 as D increased from 5 to 15 minutes. A larger value for D causes less frequent, and possibly less effective, re-assignments of resources.

It is interesting to note that even with such drastic reductions in CPU requirements there are still significant numbers of CPUs not being utilized much of the time. Figure 9 considers the case of Group one with a peak of 30 CPUs, $D = 5$, and $M = 50$. It illustrates the probability of gaining access to a number of CPUs, for either 30 or 120 minutes of computation, over all 5 minute intervals. For example, Figure 9 shows that it is possible to acquire access to 10 CPUs for 120 minutes starting at over 80% of the 5 minute intervals. In other words, the figure shows that the peak number of resources are used infrequently. Low priority scientific jobs or other batch jobs could benefit from such unused capacity.

6.2 Case B

Cases **B** and **C** consider smaller servers with various numbers of CPUs. The number of CPUs is identified in each column of Figures 5 through 8. The goal of the optimization method is to determine the peak number of small servers needed to support each group's workload. Only whole numbers of servers, i.e. integers, are considered. For these cases workload affinity is an issue. In this section, we focus on the peak number of servers that are required.

Case **B** considers the scenarios with and without workload affinity and no fast migration. These are illustrated using △s and □s, respectively. With workload affinity, all of a server's demand, a workload class, must be allocated to the same shared utility server. Without workload affinity, each CPU workload component may be allocated separately giving greater allocation opportunities.

We note that no solution was found for the Group one, $D = 5$, 4 CPU 50%*wafm* case. This is indicated by showing the corresponding □ at zero. A longer solution time limit may have led to a solution. For the cases with 6, 8, and 16 CPUs per server, we see significant differences in the reported ratio for 50%*wafm* and 50%*WAfm*. As expected, this shows that workload affinity can diminish possible savings. However, as the utilization threshold changes to $M = 80\%$, and as D changes to $D = 15$ minutes, the differences are diminished. Both of these factor values enable a greater utilization within the corresponding shared utility. With the greater potential for utilization, workload affinity matters less because there are fewer returns to be gained or lost with respect to the additional (no-affinity) allocations.

Similarly, workload affinity had the greatest impact on peak CPUs required for Groups one, three, and four. Group two, which reports the highest CPUs needed ratio, i.e. it gained the least from consolidation, was not affected by workload affinity at all.

For all groups, when allocation solutions were found, smaller servers were able to approach the savings of case **A**, the larger server scenario. However, as the number of CPUs per server increased, the savings were reduced. This is because there can be unused CPUs on the last server. Ideally, these resouces could be used for other purposes.

We note that the peak number of CPU's required, summed over all servers, should be greater than or equal to the peak number of CPUs for the large server scenario of case **A**. Group one had one anomaly, for 4 CPUs per server, 80% CPU utilization, $D = 15$, no workload affinity and fast migration (80%*waFM*). For this case an allocation was found that only required 16 CPUs, whereas the best corresponding allocation for the large server scenario was 19 CPUs. If we permitted a longer integer program solution time for the large server scenario, we would expect to find a corresponding or better allocation.

6.3 Case C

In the previous subsection we considered cases without fast migration. This subsection considers the impact of fast migration and workload affinity. Here we assume at the end of each period D work can migrate between servers. We assume a non-disruptive fast migration mechanism [4] and do not consider migration overhead in this study. This section considers both peak and mean numbers of servers.

Fast migration with and without workload affinity are shown using ∇s and $\circ$s, respectively. We note that throughout Figures 5 through 8, whenever there are solutions for cases both with and without affinity, the peak server requirements are the same. The additional allocation opportunities presented by fast migration do indeed overcome the limitations of workload affinity. Note that the impact of fast migration appears to diminish as we increase the number of CPUs per server. This is likely because of increased opportunities for sharing resources within each server.

We now compare the mean and peak number of CPUs required for the 50% CPU utilization case with no workload affinity. Groups one through four have means that are 34 to 57% lower than their corresponding peaks. For the cases with workload affinity, the means are between 34 and 71% lower than their corresponding peaks. These differences could be exploited by a shared utility by acquiring and releasing resources from a full server utility as needed. We note that for Group four, with 16 CPUs per server, the peak and mean number of servers is the same. This is because the entire Group four data center workload is satisfied using one 16 CPU server.

The greatest total potential reduction in CPU requirements is for Group three. With four CPU servers, it has the potential for up to a 79% reduction in its mean number of CPUs with respect to the original number of CPUs. In general, the cases with small numbers of CPUs per server and fast migration require close to the same mean number of CPUs as in the corresponding large server cases.

Fast migration presents shared server utilities that are made up with small servers with an advantage over large multi-processor servers. They can scale upwards far beyond any single server, and, can also scale downwards as needed, making resources available for other full server utility workloads that need them. However, the number of CPUs the shared utility can offer to any single workload class with affinity is limited to its number of CPUs per server.

Fast migration also provides a path towards the full server utility model for non-horizontally scalable applications. By shifting work from server to server, a shared utility can improve its own efficiency by acquiring and releasing servers from a full server utility over time.

6.4 Summary

The results of this section suggests there is significant opportunity for reducing numbers of CPUs required to serve data center workloads. Utility computing provides a path towards such savings. For the data center under study and a 50% target utilization, a shared utility model has the potential of reducing the peak number of CPUs up to 47% and 53%, with and without workload affinity, respectively. A large multi-processor system had the potential to reduce the peak number of CPUs by 55%. Fast migration combined with a full server utility model offered the opportunity to reduce the mean number of CPUs required by up to 79%.

Workload affinity can restrict allocation alternatives, so it tends to increase the number of small servers needed to satisfy a workload. The impact appears to be less significant when the target utilization for shared utility CPUs is higher. Fast migration appears to overcome the effects of workload affinity altogether. Measurement interval duration has more of an impact on the peak number of CPUs needed than on the mean, though as D increases, it can also reduce allocation alternatives thereby increasing the mean number of CPUs needed. This suggests that the advantages of fast migration are present even for larger values of D.

7 Summary and Conclusions

This paper presents the results of an investigation into the effectiveness of various utility computing models in improving CPU resource usage. Initially we examined the CPU utilization of approximately 1000 servers from six different data centers on three continents. The results of this analysis confirmed that many servers in existing data centers are under-utilized.

We then conducted an experiment to determine the potential resource savings that the utility computing models under study could provide. For this analysis we used an optimization method on the utilization data from several groups of servers from one of the data centers. The results of our study found that there is indeed a significant opportunity for CPU savings. The large multi-processor and shared utility model with many smaller servers reduced peak numbers of CPUs up to 55% for the data center under study. With fast migration and a full server utility model there was an opportunity to reduce the mean number of CPUs required by up to 79%. These represent signifcant resource savings and opportunities for increasing asset utilization with respect to the capacity that would be required by data centers that do not exploit resource sharing.

The work described in this paper represents only an initial investigation into the effectiveness of utility computing. Additional work is needed to determine the achieveable savings of on-line algorithms, which are necessary for implementing utility computing. Other issues that need to be addressed include security and software licencing.

We intend to extend our work in several ways. We plan to collect data from additional data centers, and from a larger fraction of the servers in each data center. In addition to collecting measures of CPU utilization, we intend to collect information on network and storage utilization, in order to get a more complete picture of data center usage. We also plan to explore on-line algorithms for dynamically adapting the number of servers required by shared utility environments when operating within full server utility environments.

References

[1] K. Appleby, S. Fakhouri, L. Fong, M. Goldszmidt, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano – SLA based management of a computing utility. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proceedings of ACM SIGMETRICS 2000*, June 2000.

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

[4] Ejasent. Utility computing white paper, November 2001. `http://www.ejasent.com`.

[5] R. Fourer, D.M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press / Brooks/Cole Publishing Company, 1993.

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

[7] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software-Practice and Experience*, 32(2):135–164, 2002.

[8] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *IWQoS*, pages 3–12, Miami, USA, May 2002.

[9] J. Rolia, S. Singhal, and R. Friedrich. Adaptive Internet Data Centers. In *SSGRR'00*, L'Aquila, Italy, July 2000.

[10] Sychron. Sychron Enterprise Manager, 2001. `http://www.sychron.com`.

[11] H. Zhu, B. Smith, and T. Yang. A scheduling framework for web server clusters with intensive dynamic content processing. In *Proceedings of the 1999 ACM SIGMETRICS Conference*, Atlanta, GA, May 1999.

[12] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation for cluster-based network servers. In *Proceedings of IEEE INFOCOM'2001*, Anchorage, AK, April 2001.