Tailoring resources: the energy efficient consolidation strategy goes beyond virtualization

Jordi Torres, David Carrera, Vicenç Beltran, Nicolás Poggi, Kevin Hogan, Josep Ll. Berral, Ricard Gavaldà, Eduard Ayguadé, Toni Moreno and Jordi Guitart.

Barcelona Supercomputing Center (BSC) - Technical University of Catalonia (UPC) - Barcelona (Spain)
torres@ac.upc.edu

Abstract

Virtualization and consolidation are two complementary techniques widely adopted in a global strategy to reduce system management complexity. In this paper we show how two simple and well-known techniques can be combined to dramatically increase the energy efficiency of a virtualized and consolidated data center. This result is obtained by introducing a new approach to the consolidation strategy that allows an important reduction in the amount of active nodes required to process a web workload without degrading the offered service level. Furthermore, when the system eventually gets overloaded and no energy can be saved without loosing performance, we show how these techniques can still improve the overall value obtained from the workload. The two techniques are memory compression and request discrimination, and were separately studied and validated in a previous work to be now combined in a joint effort. Memory compression is used to convert CPU power into extra memory capacity to overcome system underutilization scenarios caused by memory constraints. Request discrimination is used to identify web clients according to the value they have to the system. We study the combined use of these two techniques by describing a simple but still representative scenario and also by considering the dramatic impact they would have for a real workload obtained from a top national travel website. Our results indicate that an important improvement can be achieved by deciding not only how resources are allocated, but also how they are used. Moreover, we believe that this serves as an illustrative example of a new way of management: tailoring the resources to meet high level energy efficiency goals.

1. Introduction

Companies are now focusing more than ever on the need to improve energy efficiency. In addition to the cost of energy, a new challenge for them is the increasing social pressure to reduce their carbon footprint. Commercial electricity consumption is a major factor in rising atmospheric CO2 levels and data centers are a significant part of the problem. While energy costs are rising and data center equipment is stressing the power and cooling infrastructure, the main issue is not the current amount of data center emissions, but the fact that data center emissions are increasing faster than any other carbon emission. For this reason nowadays there is a big interest in "Green" data centers and supercomputer centers [10]. In this scenario, the research community is being challenged to rethink data center strategies, and add energy efficiency to a list of critical operating parameters that already includes service ability, reliability and performance.

Consolidation and virtualization can be combined to reduce the management complexity of large data centers as well as to increase the energy efficiency of such a system. But even in a scenario where the resources are consolidated and virtualized, utilizing all the capacity of the components that are switched on (and consuming power) is not always simple. To determine a set of applications to be collocated in a node to perfectly fit and exploit all the resources of the system is a hard problem to solve, especially when tenths or even hundreds of nodes and applications can be found in a data center. Furthermore, the fact that the demand associated with each system resource for a given application may not be related in any way to its demand for other resources (i.e. an application with a large memory footprint may not be very demanding in terms of CPU power) which creates a structural problem requiring constraints to be relaxed in order to overcome it.

In this paper we present how these two simple and well-known techniques can be combined to dramatically increase the energy efficiency of a virtualized and consolidated data center. Increased energy efficiency is obtained through the introduction of a new approach to the consolidation strategy by combining: memory compression and request discrimination. Combining these techniques enables an important reduction in the amount of active nodes required to process a web workload by dynamically classifying and shaping the workload, without degrading the offered service level. Furthermore, when the system eventually gets overloaded and no energy can be saved without loosing performance, we show how request discrimination can still improve the overall value obtained from the workload. The two techniques were separately studied and validated in a previous work to be now combined in a joint effort.

Memory compression is used to convert CPU power into extra memory capacity to overcome system underutilization scenarios caused by memory constraints. Request discrimination is used to characterize web clients by predicting their class and the value they will have on the system. Our experiments performed on a real workload, obtained from a top national travel service exemplify the dramatic improvement these thechniques offer in energy and performance efficiency.

The main contribution of this article is to demonstrate that the consolidation of dynamic workloads does not end with virtualization, but there is even more to consolidate when energy-efficiency is pursued. We will present two alternatives to rescue resources that consolidation does not currently capitalize on while using virtualization.

The rest of the paper is organized as follows. In Section 2 we study the related work found in the literature. Section 3 describes the basics in dynamic resource management that we assume as available in our work. Section 4 discusses the memory compression and request discrimination techniques applied in our work. Section 5 formally states the problem we address. Section 6 discusses our study and obtained results. Finally, some conclusions and future work are discussed in Section 7.

2. Related Work

Server consolidation has become very popular following the advances in virtualization technologies [17,9,2], it allows the control of how resources are allocated to the running applications. Dynamic allocation of server resources to applications has been extensively studied [4,6,12,14,23], however these proposals do not benefit from the techniques we are introducing to go beyond virtualization by tailoring resources. The problematic of consolidating multi-tier applications considered in [19] is complementary to our proposal . Also of great importance is the topic considered in [18] regarding the power-efficient management of enterprise workloads which exploits the heterogeneity of the platforms. Our proposal could be included in the analytical prediction layer proposed by the authors. Finally let us remark that our proposal could be combined with power-saving techniques at the lowest level such as dynamic voltage scaling and frequency scaling [9,15,26]. In a recent work [13], the authors use frequency scaling in a scheme that trades off web application performance and power usage while coordinating multiple autonomic managers. In this case the proposals of this article could be included in the utility function that they are using.

3. Managing a consolidated and virtualized environment

The techniques proposed in this paper are studied in the context of a virtualized data center where applications share hardware resources and a set of web applications and being runned by an automatic service management middleware, such as that described in [29]. At each moment in time, a number of application server instances are running in the system to host the applications deployed on the data center. The management middleware monitors the actual service level offered by each application and dynamically changes the configuration of the system to make the applications meet their goals. In particular, the system has to decide how many instances must be running for each application to meet its goal as well as in what nodes these instances are going to be placed: this is what is known as the placement problem. For the purpose of our work, we will assume that the data center uses virtualization technology [24,28] to control the resources allocated to each application by running each instance inside of a virtual machine container. In the scope of this paper we'll use a simple instance placement algorithm to illustrate the benefits of our techniques, but any other approach could be considered.

The placement problem itself is out of the scope of our work. The techniques described in this paper can be helpful to any placement algorithm by relaxing one of the hardest constraint they have to deal with: the system capacity. Existing dynamic application placement proposals provide automation mechanisms by which resource allocation may be continuously adjusted to the changing workload. Previous work focuses on different goals, such as maximizing resource utilization [12] and allocating resources to applications following their service level goals [22, 6, 25], our proposal could be applied to improve the mentioned aproaches. In order to correctly define the placement scenario, we can be assume that the system is able to derive the relation between resource allocation and obtained service level for each application in the system, as is reported in [22].

4. Beyond consolidation and virtualization

In the following two sections we briefly discuss two simple but effective techniques to relax some of the constraints present in the placement problem. Both of them were studied in detail in some previous work and are now combined in a joint effort towards achieving energy-efficiency.

Memory compression is used to convert CPU power into additional system memory. The amount of extra memory produced using this technique can potentially go beyond consolidation through virtualization in two aspects: firstly, allowing the placement of an extra application that did not

fit in a node before, therefore reducing node underutilization; and secondly, increasing the performance of a placed application that, with a given amount of memory, can still run but at a fraction of the maximum achievable performance (i.e. producing a big volume of swapping activity).

Request discrimination is introduced to classify web requests according to the value they have to the system. In our work, the request classification pursues two different targets: to identify and reject those requests that consume system resources but have no value for an application (i.e. requests coming from web crawlers created by competitor businesses with content stealing purposes presented in in [20]); and by prioritiezing those requests that add more value to the system, in overload conditions.

Notice that both of the techniques described here can produce a similar effect in a system: reducing the number of nodes necessary to meet a certain service level criteria. This extra consolidation is achieved through memory compression by increasing the number of application instances that can be placed in a node, and through request discrimination by reducing the load on the system, thus allowing more options to collocate applications.

4.1 Reduction through discrimination

A fraction of the resources are wasted on work that yields no added value for the application or the company running it: consider an e-commerce site, and the amount of work performed for customers that will not buy. Furthermore, some work can be harmful to the system: e.g. requests coming from denial-of-service attacks, or the traffic generated by malicious bots. Work with no added value is a prime candidate for reduction, especially when the system is overloaded and accepting it causes an overall loss. On the other hand, any potentially harmful requests that can be detected should be banned as soon as possible, even if the system is far from overload.

Let us comment on the work in [21] and [20], which address these two problems; namely, detecting customers who generate no revenue in an e-commerce application, and detecting malicious bots with the purpose of banning them. The case study in these works is a national online travel agency that works as an electronic intermediary between customers and service providers (such as airline companies), with €40M sales volume during 2006 and an expected €100M during 2007. More precisely, in [20,21] and later experiments we have used web traffic logs from different periods of the year, ranging from one day to a week of traffic, with up to 3,7 million transactions. Each transaction is a particular request to the web site (such as requesting a page download, possibly including many database searches). Transactions are grouped into user sessions, with an average length of about 8 transactions per session for non-buying sessions, and about 18 transactions per session for sessions that end in a purchase. About 6.7% of transactions belong to sessions which will end in purchase. Figure 1 shows the pattern for the amount of traffic in a high-season week, where the daily and weekly patterns are clearly visible.

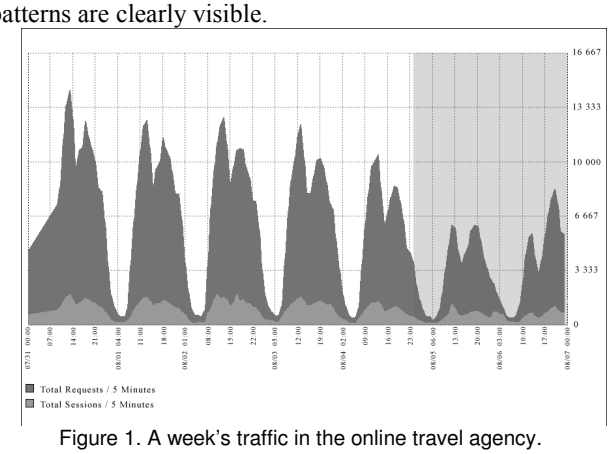

Figure 1. A week's traffic in the online travel agency.

The problem tackled in [20] is that of detecting stealing bots in e-commerce applications. Content stealing on the web is becoming a serious concern for information and e-commerce websites. In the practices known as web fetching or web scraping [11], a stealer bot simulates a human web user to extract desired content off the victim's website. Not only that, but in a B2B scenario, the victim incurs the costs of searching the provider's web for a supposed "customer" that will never buy, and looses the real customers who will instead buy via the stealer's web.

The work in [20] investigated whether it was possible to identify with reasonable certainty bots accessing a web site for automated banning so that the system could stop the corresponding session and free the allocated resources. In the mentioned online travel agent website, [20] concluded that around 15% to 20% of the traffic corresponds to bots other than simple crawlers; note that a feature of stealer bots is the large amount of search requests, hence this large traffic figure. Applying machine learning techniques, the authors were able to detect around 10%-12% of the total traffic as bots with a low % of "false alarms" and negligible overhead at runtime. This percentage of traffic could be banned in the real scenario, even when the system is not overloaded, since it is actually harmful to the company's interests to serve them. While the interest of the authors in [20] is leveraging revenue loss from the spurious transactions, it is easy to see this technique as a way to reduce the allocated resources: If we expect that we could ban 10%-12% of the incoming traffic as bots, we can reduce the resources assigned to the application by a similar percentage when deploying it.

In any case, a key point is finding the relation between load reduction and resource reduction. The experiments in some of our EU-funded projects [5], where we have researched the dynamic management of resources, let us

conclude that there is essentially a linear relation among load volume and CPU usage. That is, if we reduce the number of requests by 10% or 15%, the CPU requirement will be reduced by at least 10% or 15%. The reduction will probably be larger if the transactions we discard are especially heavy ones (which is the case for stealing bots). We cannot at this moment, make similar claims for other resources, such as memory, which we are still investigating. For this reason we center our work only on CPU even though we believe that we will be able to extend the conclusions to other resources soon.

We turn now to the case of traffic that could be reduced in case of overload. The workload of the site mentioned above and similar sites has clear daily, weekly, and seasonal patterns, but is also affected by engine rankings, promotions, and advertising campaigns. These external events make capacity planning and resource management a complex task: it is difficult to predict when these user peaks are going to occur, before they start happening. So for an online intermediary, these situations are frequent and not serving users is a loss in revenue. One way to meet peaks in such dynamic workloads is, of course, by over-provisioning. But over-provisioning means having resources that are idle most of the time, and consumes power. The approach taken in the AUGURES system [21] is to use the same machine learning module as in [20] to predict, as early as possible, whether a user session is going to end in a purchase or not, so that in case of overload non-promising sessions can be delayed or dropped. The net effect is that of simulating a certain amount of over-provisioning, because during high load peaks we will accept a part of the revenue (buying customers) that would otherwise be lost.

Figure 2 shows the percentage of buyers in the traffic admitted by AUGURES when the percentage of admitted traffic varies from 100% to 10%, in comparison with the basic strategy that accepts requests at random. The area between both plots represents admitted buyers that would otherwise be lost. For example, when the % of admitted traffic is 40% of the total traffic, the % of accepted buyers is double that which would be accepted by randomly accepting customers. Recall that, in average, if all the traffic is accepted, the percentage of buyer clients is observed to be around a 6.7%.

Note that, from a marketing point of view, dropping a user's session *does* have a cost in prestige or user dissatisfaction, so this technique should be used only when no more requests can be served. On the other hand, when power consumption is thrown into the equation, its cost may well balance the marketing cost of rejecting a few, clearly non-buying, sessions even in non-overload situations.

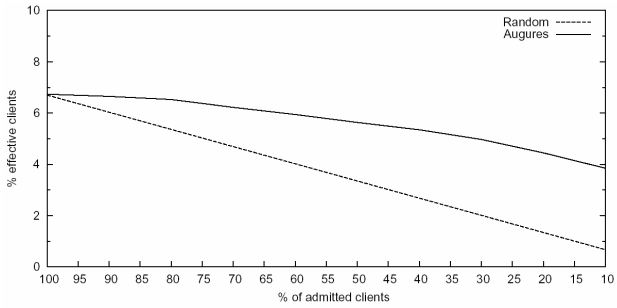

Figure 2. Number of buyers accepted in case of overload for AUGURES vs. random

4.2 Recycling though resource transformation

After virtualizing a system, some resources may still not be used by any application. The demand associated with each resource of the system for a given application may not be related in any way to the demand of other resources (i.e. an application with a large memory footprint may not be very demanding in terms of CPU power), which can potentially lead to an underutilization of some resources in the system.

To illustrate this situation, figure 3a shows a usual placement problem: some applications could be placed in a node in terms of CPU power (they would meet their performance goals), but the memory capacity of the system makes it impossible to place all the applications together. As a result, one extra node must be used to place one of the applications, and both of the nodes remain underutilized in terms of CPU.

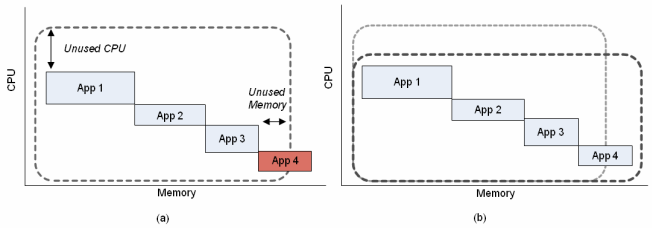

Figure 3. Resources can be changed from one form to another: (a) cpu to (b) memory.

Memory compression is a widely studied topic that can be really helpful for the placement problem. It allows the system to increase the density of the placement (number of applications placed on a node) to better exploit the resources of the system. This process can be understood as a resource transformation: CPU cycles are converted into extra memory. Figure 3b shows the same placement problem but considers hat some extra memory can be gained at the price of losing some CPU power.

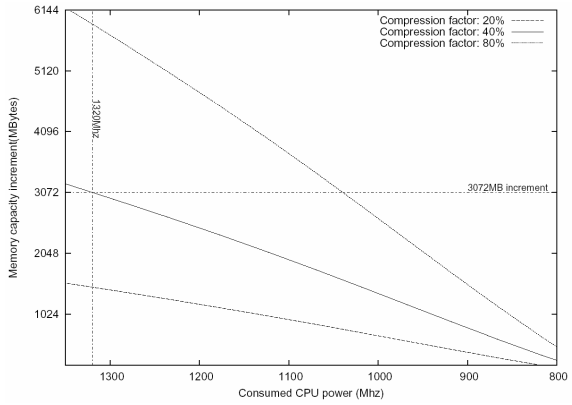

Figure 4. Trade-off between memory and computational power required to achieve it.

Some of our recent work, described in [3], is focused on revisiting the memory compression topic by targeting advanced hardware architectures (current multiprocessors and multi-core technologies such as CELL [27] and Niagara [16]). This study concludes that memory compression can be carried out without observing a significant performance impact in many commercial applications (the study is performed over the SPECWeb2005 [30] application). The relation between the CPU power dedicated to compress memory and the memory gain obtained for three different levels of memory compressibility is represented in figure 4. Obviously, this relation is always defined by the level of memory compression achievable given a set of applications.

From the point of view of the applications, the overhead produced by memory compression techniques is negligible because although accessing compressed data is slower than accessing regular memory, it is still faster than accessing a standard SCSI disk. This means that the reduction in swapping by adding compressed memory as well as caching more data in the compressed memory can still result in a performance improvement for most applications.

Current work in this area covers offloading the compression process (i.e. using the SPU units of the CELL processor) and also adding compression support to the most extended virtualization products. Thus, the results presented in this paper are expected to even improve in the future.

5. Problem statement

5.1.1 The placement problem

In order to formally specify our proposal we will start by formulating the placement problem and then include our proposals in the placement algorithm.

We are given a set of servers S={1, . . . ,S} and a set of applications A={1, . . . ,A}. We use s and a to index into the sets of servers and applications, respectively. With each server S_s we associate a memory and CPU capacity (noted as Ω_s and Γ_s respectively).

With each application a_a, we associate its memory demand, γ_a that represents the amount of memory consumed by this application whenever it is started on a machine, which we assume to be load-independent as described in [22]. In general the CPU requirements of applications, ω_a, are given as being variable and are specified in different forms depending of the type of the application a_a.

In this context, the placement problem is to find matrix I which denotes a placement matrix of applications on servers, where $i_{m,n}$ is 1 if an instance of application m is running on server n.

The placement problem is known to be NP-hard [12,6] and heuristics must be used to solve it. While finding the best possible placement, the heuristic must observe a number of placement constraints, such as collocation and allocation restrictions. For any server, s, it must be enforced that $\sum_m i_{m,s}\ \gamma_m \leq \Gamma_s$ and $\sum_m i_{m,s}\ \omega_m \leq \Omega_s$ to guarantee that the nodes are not overallocated. Given a certain workload, changing the allocated CPU power to an application makes a significant difference in the service level offered by that application. But changing the amount of memory allocated to an application results in an even higher impact, because the application can be placed or not, depending on whether the amount of memory reserved to run it is enough or not to place it. This leads to a scenario where the placement problem can be represented as two different problems: placing applications following memory constraints and spreading CPU resources amongst the placed instances.

The objective of our work is not to focus on solving the placement problem as defined above but to introduce a new degree of freedom into it to allow the system find a new set of application placements that offer the same service level to each application but require different resource allocations. This objective is achieved by relaxing the allocation constraints, and by relaxing the hardest constraint in the system: the available physical resources in each node of the data center.

5.1.2 Relaxing some placement constraints

In order to incorporate the resource transformation property into the problem, first we should model the compression memory system. We assume that for each application we can determine if it could take advantage of this property. In this case we could define a function Φ that establishes the relation between CPU power dedicated to compress memory and the amount of gained memory (as shown in figure 4). Now the memory constraint becomes $\sum_m i_{m,s}\ \gamma_m \leq \Gamma_s + \Psi$ and the CPU constraint becomes $\sum_m i_{m,s}\ \omega_m \leq \Omega_s$-$\Phi$ (Ψ), where Ψ is the amount of CPU dedicated to memory compression.

Regarding the request discrimination, in this section we will assume a non-overloaded system (section 6.4 will focus on overloading conditions). We have shown in section 4.1 that under these circumstances a percentage of traffic could be banned since it is actually harmful for the company's interests to serve them. This means that we can assume that a certain application *m* can see its resource demand reduced by a factor δ_m. A summary of the differenct constraint scenarios is shown in Table 1.

In the experiments presented in the next section we consider the impact (in terms of energy-efficiency) that using the new constraints would have for a system, assuming that the overall performance is kept unaltered.

Scenario	Constrains
Virtualization	$\sum_m i_{m,s}\ \gamma_m \leq \Gamma_s$ $\sum_m i_{m,s}\ \omega_m \leq \Omega_s$
Transformation	$\sum_m i_{m,s}\ \gamma_m \leq \Gamma_s : \sum_m i_{m,s}\ \gamma_m \leq \Gamma_s + \Psi$ $\sum_m i_{m,s}\ \omega_m \leq \Omega_s : \sum_m i_{m,s}\ \omega_m \leq \Omega_s$-$\Phi$ (Ψ)
Discrimination	$\sum_m i_{m,s}\ \gamma_m \leq \Gamma_s$ $\sum_m i_{m,s}\ \omega_m \leq \Omega_s : \sum_m i_{m,s}\ (\omega_m - \delta_m) \leq \Omega_s$
Transformation + Discrimination	$\sum_m i_{m,s}\ \gamma_m \leq \Gamma_s : \sum_m i_{m,s}\ \gamma_m \leq \Gamma_s + \Psi$ $\sum_m i_{m,s}\ \omega_m \leq \Omega_s : \sum_m i_{m,s}\ (\omega_m - \delta_m) \leq \Omega_s$-$\Phi$ (Ψ)

Table 1. A summary of the constraints under different scenarios

6. Experiments

In this section we evaluate our proposal using two different experiments. First we demonstrate how memory compression can help save energy in a synthetic but illustrative scenario. Later, using a real workload provided by the travel website described in [20,21], we evaluate the impact and options opened up by the combination of memory compression and request discrimination when a system gets eventually overloaded. Before that, we describe the synthetic scenario and discuss how a state-of-the-art automatic management middleware (without energy-saving goals) would work for this scenario.

6.1 Scenario description

We consider a scenario composed of 3 identical servers and 4 different web applications. Neither allocation restrictions nor collocation restrictions are defined, but placement is still subject to resource constraints, such as the node memory and CPU capacity.

We consider that each server has four 2.2GHz CPU and 4Gb of memory (based on an IBM JS21 blade). We assume that the virtualization overhead is 1Gb of memory and 1 CPU. This assumption is based on our previous experience [5]. Table 2 summarizes the specifications of each node.

No virtualization		**Virtualization overhead**	
CPU capacity	**Memory capacity**	**Effective CPU capacity**	**Effective mem. capacity**
4x 2.2Ghz	4096MB	3x 2.2Ghz (6.6 Ghz)	3072 MB

Table 2. Memory and CPU capacity of each node before and after considering the virtualization overhead

The characteristics of each application are described in table 3. Notice that application 1 can not be placed together with any other application because of the memory constraints. Applications 2, 3 and 4 can be collocated, but only two of them can be placed together in each node.

Applications	**Minimum Memory required**	**Maximum CPU required (spike)**
A1	2300 Mb	2200
A2	1300 Mb	2000
A3	1100 Mb	2000
A4	1000 Mb	1900

Table 3. Memory and CPU required by the Applications used in the experiments

Figure 5 shows the CPU demand required by each application over time to meet its service level goals. That is, the minimum amount of CPU power that must be allocated to each application if its service level goal wants to be met. Notice that there is no overloading at any point of the experiment (the aggregated CPU power of the four nodes can satisfy the requirement of all applications over the time), as can be seen in figure 6. Notice that this placement lead to a situation where the three nodes are clearly underutilized Labels A, B, C, D indicate 4 key points in the experiment that will be used later to analyze it in detail.

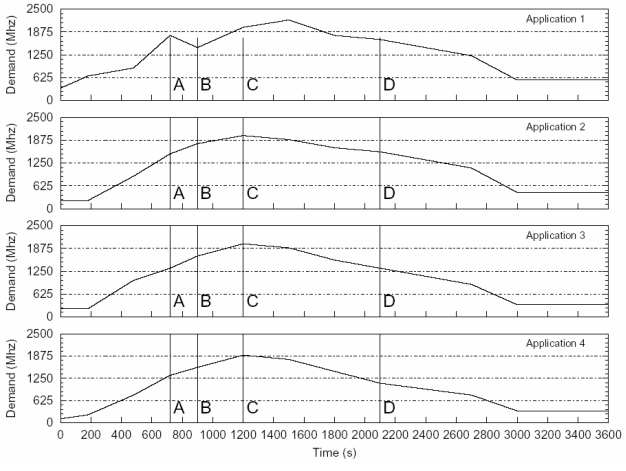

Figure 5. CPU demand of applications to meet their service level goals

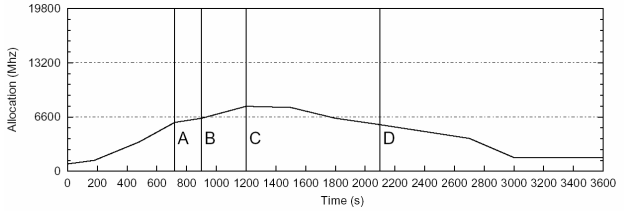

Figure 6. Aggregated CPU demand

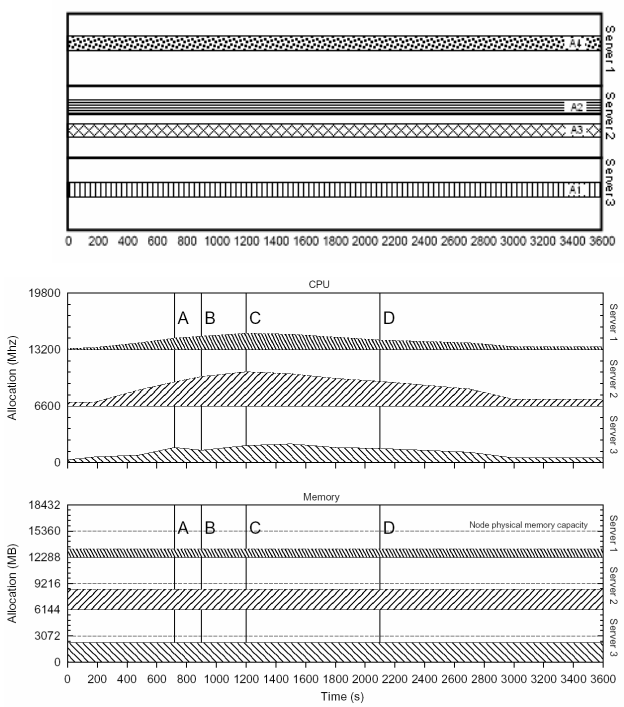

Figure 7. (a) Application placement: (b) memory used and (c) CPU used

6.2 Baseline placement

In this section we describe what a modern management middleware would do in the scenario described above. As we said before, application 1 can not be placed together with any other of the other applications because of its memory requirements. Given that the CPU demand of application 1 can be satisfied by one single node, we assume that this application would be placed in one node for the whole length of the experiment. The other applications must be placed in the two remaining nodes. Given that all three applications don't fit in one single node because of the memory constraint, two of them will have to be placed together while the other application will be alone in one node. Thus, the placement algorithm should decide at this point what two applications are going to be placed together. For this experiment we decide to pick application 2 and 3 to be deployed on node 2, and application 4 to be placed in node 1. Notice that other choices are possible but that the result would be analogous to that presented here. Figure 7 shows the application placement as well as the aggregated CPU and memory demand satisfied by each node.

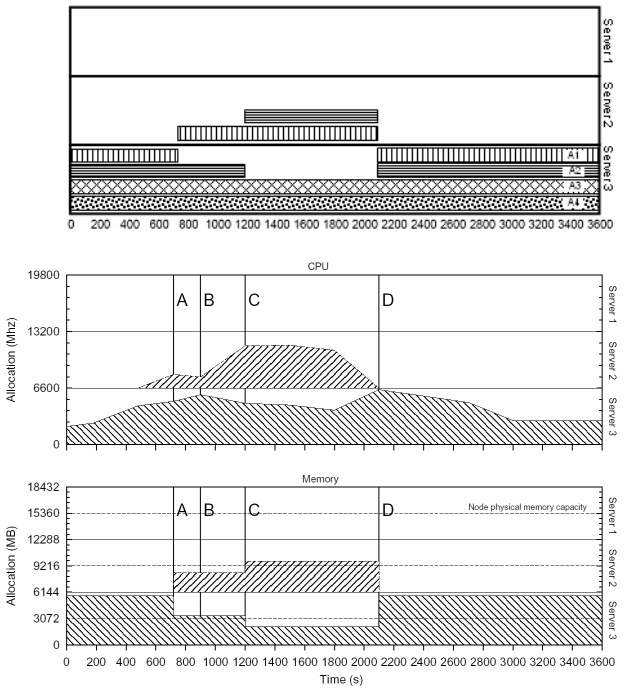

Figure 8. (a) Application placement using tailoring (b) memory used and (c) CPU used

6.3 Adding memory compression

At this point, we introduce the use of memory compression to increase the memory capacity of a node on demand. The memory, as discussed in Section 4.2, is produced at a cost in terms of CPU power. Notice that in the scenario described in section 6.2, memory constraints lead to a situation where the three nodes are clearly underutilized in terms of CPU power.

Looking at the data provided in Section 4.2 (which is based on real experiments conducted with realistic applications on top of an IBM JS21 blade server) one can observe how, depending on the compression rate achievable for a given set of applications placed in a node, a relation can be established between the CPU power required to compress memory and the increase in available memory observed. In the scope of this experiment, we assume an achievable compression factor of 47% (see Figure 3 for more details), and will use an increased memory capacity for each active node of 6GB at a cost of 1320MHz of CPU power.

With the new constraints, a new range of possible placements is opened up, including the option of having all four applications placed together on one single node.

For our experiment we picked that described in Figure 8. Initially, the four applications are placed in node 3, until point A is reached. Looking at Figure 9 we can see that before that point, all the CPU power required by the four applications can be satisfied, including the memory compression overhead of up to 1320MHz. Notice that 2 nodes can be switched off before time A when only applying a software memory compression technique, and also that in Figure 8 more memory can be allocated to each application than the physical memory capacity of a single node.

When time A is reached, the aggregated CPU demand exceeds the capacity of a single node. At that point, application 1 is migrated to a second node which is switched on for this purpose.

At time B, the CPU demand for application 1 is reduced while the demand for the other applications keeps raising. At time C the aggregated CPU demand put on server 3 by applications 2, 3 and 4 exceeds the capacity of the node. Hence one of the applications must be migrated to another server; in this case S2. Finally, at point D, aggregated CPU demand for all the applications can be satisfied again with one single node and thus all the applications are placed again in server S1.

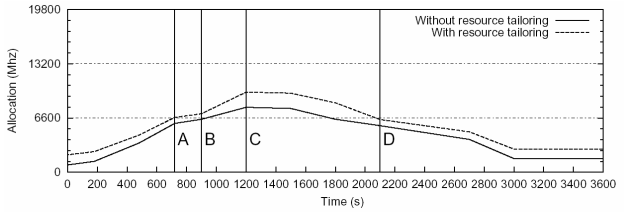

Figure 9. Total CPU used with and without tailoring

6.4 Overloaded scenario

Once a placement is decided, the resources allocated to each application are determined. After that, the application can be either overloaded or not, depending on the observed load. In the case an application is overloaded and not all the demand can be satisfied, a portion of the received requests must be dropped. Even in this hard scenario, request discrimination can be useful to increase the aggregated value of the system by considering the value associated to each request. In this section we provide an illustrative example of the benefits of this technique under overload conditions. In this section we provide an illustrative example of the benefits of this technique under overload conditions.

Looking at Figure 10 (derived from Figure 2 but expressed in terms of overload instead of admission rate) it can be observed that for the workload (previously described in section 4.1) the average percentage of requests that belong to a web session that will end in purchase (the percentage of buyers) is around 6.7% when no overload is present. As peak loads increase, our request discrimination technique is able to identify those clients with a higher potential revenue and avoid most profitable sessions to be dropped.

Figure 11 is derived from the data shown in Figure 1. It summarizes real traffic data coming from the travel website for two week days (48 hours) in 5 minute samples. Over this data we assume that the resources allocated to this application satisfy only around 60% percent of the maximum observed load. That is, when the load is higher than 9000 requests every 5 minutes, the system starts dropping requests.

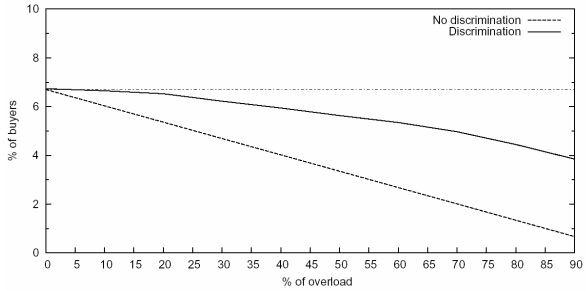

Figure 10. Effect of applying resource-tailoring when different % of the incoming traffic can be handled by the system.

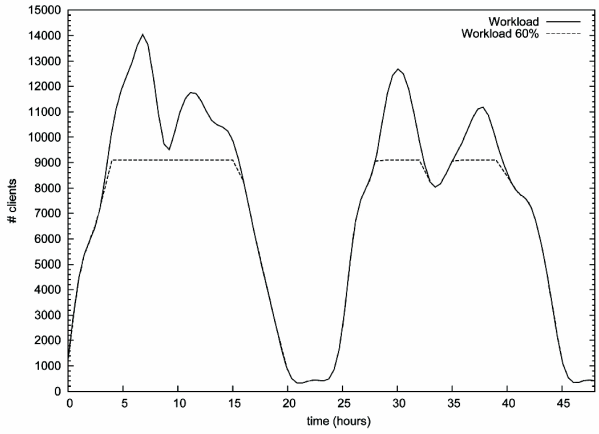

Figure 11. Real workload Vs accepted workload (assuming only 60% of the peak load can be processed)

Figure 12 shows the total number of requests corresponding to buying clients, represented by the solid line. This number is estimated to be 6.7% in average for this particular workload as it is discussed in Section 4.1. Using the data obtained from [20,21] and summarized in Figure 10, we can compute the fraction of requests corresponding to purchasing sessions that will be processed if no discrimination is applied (dashed line) and if the request discrimination technique is applied (dotted line).It can be

observed from the figure, that in periods of time when the system is not overloaded all the requests are accepted and thus all the requests corresponding to purchasing sessions are processed. On the other hand, when the system is overloaded the request discrimination technique (AUGURES) can improve the obtained result by dropping first the requests corresponding to non-purchasing sessions (those with a lower value to the system).

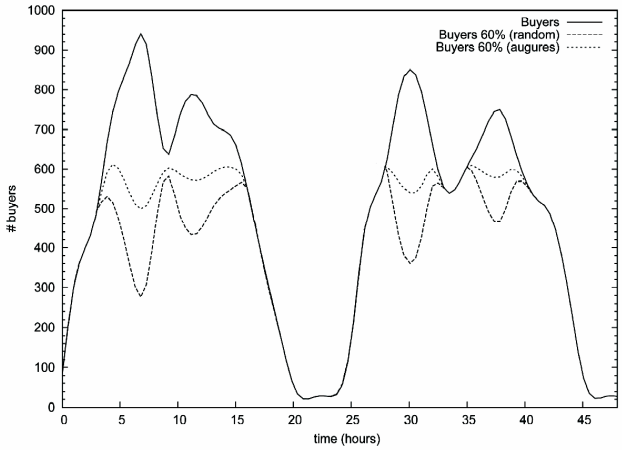

Figure 12. Total requests corresponding to buyer clients Vs accepted buyers with and without request discrimination

7. Conclusions

In this paper we demonstrate how consolidation with energy efficiency goals still has a long way to go beyond the use of virtualization. The use of virtual machine containers to run applications in consolidated data centers can save energy, while at the same time reducing system management complexity; however, under sole virtualization, system resources will still remain underutilized incurring in energy consumption. In this work, we identify new opportunities to improve the energy efficiency of systems, reducing the resources required, without negatively impacting the performance or user satisfaction. Obtained results show that a combined use of memory compression (to convert CPU power into extra memory) and request discrimination (to reduce the load put on the systems as well as to overcome overloading conditions) can boost the energy savings in a data center. Overall, we consider that this work is only an illustrative scenario of a new way of management: tailoring the resources to meet high level energy efficiency goals.

We are already working on the implementation of a prototype system that applies the techniques described in this paper. We would like to extend our work to consider other techniques that could be added in terms of availability such as self-healing techniques [1] and therefore take better advantage of the resources available. Our current work on the memory compression technique is focused on the use of hardware accelerators to offload the computational cost of the compression (i.e. using the SPE units in the CELL processor) and to include compression as a feature of the most widely used virtualization technologies.

Our interest as a group involves creating power-aware middleware to contribute to building energy-efficient data centers. Somehow, the next generation of computing systems must achieve significantly lower power needs, higher performance/watt ratio, and higher reliability than ever before.

8. Acknowledgments

The authors would like to thank Manish Parashar, Jeff Kephart and Ricardo Bianchini for their comments on this topic. Also the authors would like to thank Iñigo Goiri for the resource usage measurement required in Section 4. This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contracts TIN2004-07739-C02-01 and TIN2005-08832-C03-03

9. References

[1] J. Alonso, L. Silva, A. Andrzejak, P. Silva and J. Torres “High-Availability Grid Services through the use of Virtualized Clustering“. The 8th IEEE/ACM International Conference on Grid Computing (GRID 2007). September 19-21, 2007, Austin, Texas, USA.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization” in Symposium on Operating Systems Principles (SOSP), Bolton Landing, NY, 2003

[3] V.Beltran, J. Torres and E. Ayguade “Improving Disk Bandwidth-Bound Applications Through Main Memory Compression” MEDEA Workshop MEmory performance: DEaling with Applications, systems and architecture. Brasov, Romania. Held in conjunction with PACT 2007 Conference Sept. 15-19 2007

[4] N. Bobroff, A. Kochut, and K. Beatty, “Dynamic placement of virtual machines for managing SLA violations,” in Integrated Network Management, Munich, Germany, May 2007.

[5] BREIN Project. http://www.eu-brein.com/

[6] D. Carrera, M. Steinder, I. Whalley, J. Torres and E. Ayguadé. Utility-based Placement of Dynamic Web Applications with Fairness Goals. Submitted to IEEE/IFIP Network Operations and Management Symposium (NOMS 2008).

[7] J. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in ACM Symposium on Operating Systems Principles, 2001.

[8] Elnozahy, M. Kistler, and R. Rajamony, "Energy conservation policies for server clusters," in 4th USENIX Symposium on Internet Technologies and Systems, Seattle, WA, Mar. 2003.

[9] R. Figueiredo, P. Dinda, and J. Fortes, "A case for grid computing on virtual machines" in International Conference on Distributed Computing, Providence, RI, May 2003.

[10] Green Grid Consortium, 2006 http://www.thegreengrid.org/

[11] Hepp, M., D. Bachlechner, and K. Siorpaes. Harvesting Wiki Consensus - Using Wikipedia Entries as Ontology Elements. Proceedings of the ESWC2006, Budva, Montenegro, 2006.

[12] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, A. Tantawi, "Dynamic placement for clustered web applications" In WWW Conference, Edinburgh, Scotland (2006)

[13] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, and F. R. an C. Lefurgy, "Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs," in IEEE Fourth International Conference on Autonomic Computing, Jun. 2007.

[14] T. Kimbrel, M. Steinder, M. Sviridenko, A. Tantawi, "Dynamic application placement under service and memory constraints". In International Workshop on Efficient and Experimental Algorithms, Santorini Island, Greece (2005)

[15] B. Khargharia, S. Hariri, and M. S. Youssif, "Autonomic power and performance management for computing systems," in IEEE International Conference on Autonomic Computing, June 2006.

[16] P. Kongetira, K. Aingaran, and K. Olukotun. "Niagara: A 32-way multithreaded sparc processor". IEEE Micro. 2005 (Vol. 25, No. 2).

[17] S. Nanda and T. Chiueh, "A survey of virtualization technologies" Stony Brook University, Tech. Rep. TR-179, Feb. 2005.

[18] R. Nathuji, C. Isci, E. Gorbatov. "Exploiting Platform Heterogeneity for Power Efficient Data Centers". In IEEE Fourth International Conference on Autonomic Computing, June. 2007.

[19] P. Padala, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem, and K. Shin. "Adaptive control of virtualized resources in utility computing environments". In Proc. European Conference on Computer Systems (EuroSys'07), March 2007.

[20] N. Poggi, J.L. Berral, T. Moreno, R. Gavaldà and J. Torres. "Automatic Detection and Banning of Content Stealing Bots for E-commerce". In Workshop on Machine Learning in Adversarial Environments for Computer Security (NIPS 2007).British Columbia, Canada. Dec. 2007

[21] N.Poggi, T. Moreno, J. Berral, R. Gavaldà, J. Torres. "Web Customer Modeling for Automated Session Prioritization on High Traffic Sites". In 11th International Conference on User Modeling. Corfu, Greece, June, 2007.

[22] M. Steinder, I. Whalley, D. Carrera, I. Gaweda and D. Chess. "Server virtualization in autonomic management of heterogeneous workloads". In 10th IFIP/IEEE International Symposium on Integrated Management (IM 2007), May 2007.

[23] C.-H. Tsai, K. G. shin, J. Reumann, and S. Singhal, "Online web cluster capacity estimation and its application to energy conservation," IEEE Transactional on Parallel and distributed Systems, vol. 18, no. 7, 2007.

[24] VMware EMC. http://www.vmware.com

[25] X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, Q. Wang. "Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center". in IEEE Fourth International Conference on Autonomic Computing, June 2007.

[26] M. Wang, N. Kandasamy, A. Guea, and M. Kam, "Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters," IEEE 3th International Conference on Autonomic Computing, June 2006.

[27] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. "The potential of the cell processor for scientific computing". In CF '06: Proceedings of the 3rd conference on Computing frontiers, New York, NY, USA, 2006. ACM Press.

[28] Xen. http://www.xensource.com

[29] "WebSphere eXtended Deployment," http://www-306.ibm.com/software/webservers/appserv/extend/

[30] Standard Performance Evaluation Corporation. SPECweb2005. http://www.spec.org/web2005/