Energiesparverfahren für Server Cluster

Studienarbeit im Fach Informatik

vorgelegt von

Stephan Sigwart

geboren am 23. April 1980 in Villingen

Institut für Informatik,
Lehrstuhl für verteilte Systeme und Betriebssysteme,
Friedrich Alexander Universität Erlangen-Nürnberg

Betreuer:	Dr.-Ing. Frank Bellosa
	Dipl.-Inf. Andreas Weißel
	Prof. Dr. Wolfgang Schröder-Preikschat

Beginn der Arbeit:	04. Juli 2003
Abgabedatum:	04. April 2004

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 04. April 2004

Power Management for Server Clusters

Student Thesis

by

Stephan Sigwart

born April 23rd, 1980, in Villingen

Department of Computer Science,
Distributed Systems and Operating Systems,
University of Erlangen-Nürnberg

Advisors:	Dr.-Ing. Frank Bellosa
	Dipl.-Inf. Andreas Weißel
	Prof. Dr. Wolfgang Schröder-Preikschat

Begin: July 4th, 2003
Submission: April 4th, 2004

Abstract

Power consumption is becoming a more and more important issue when there are many computers deployed in a server farm. Especially with up-to-date processors becoming faster and more complex, a significant fraction of the operation costs in a data center, which is hosting many servers, is actually spent on power consumption and cooling infrastructure. While there is already a lot of research done on how to balance incoming requests in such a cluster for best performance, there is nowadays a growing need to incorporate the aspects of power consumption.

There already exist some publications covering this research topic, presenting various methods and strategies as possible solutions. However, often different parameters and setups are used for the analysis forming the basis of those methods. Usage of different data as test-input, miscellaneous system platforms and varying assumptions about energy consumptions at idle- and peak-load are typical examples for such differences. Additionally, missing explanations about heuristics in the underlying algorithms do not allow an easy comparison between the quality of the presented results. So it is not possible to check whether or not those results are transferable to another system platform or workload situation. For this purpose, it would be necessary to check whether the heuristics and parameters used in the methods are tuned to fit the particular analyzed environment.

This work provides a synopsis about important aspects in cluster power management and points out characteristics by means of which different power management policies can be classified and analyzed. A survey of existing approaches is given, listing the particular underlying assumptions as well as a classification of the respective policy proposed to reduce the power consumption.

In addition, a simulator is presented which is capable of representing both different policies and scenarios by adjusting its parameters. This allows e.g. to simulate testing scenarios presented in various existing approaches with the same system platform parameters, so that an overall comparison can be done. Additionally, the heuristics of an algorithm may be investigated further. The simulator is using real traces from server logs which are identical to those applied in some existing publications. An implementation of several representative strategies is presented as an example and also used for analysis of their algorithms, parameters and heuristics.

Contents

Chapter 1

Introduction

1.1 Motivation

Operating a server cluster with a medium to large number of servers is an expensive matter. Not only with respect to purchase costs, but also with respect to the power and cooling infrastructure required to run such a facility. In the area of handhelds and other battery-buffered devices, saving energy has been an obviously interesting research topic from the very beginning. The motivation for that interest lies in the positive effects arising of such savings: lower power consumption means lower energy consumption, which means longer runtime for a battery-buffered device. In addition, electrical components consuming less power produce less heat, which allows usage of smaller cooling components, thus decreasing the overall size of the device.

Nowadays, this is evolving into a more and more critical issue in server clusters, too, as modern processors are becoming much more complex than those a few years ago. Simple heatsinks do not cover the needs any longer, more sophisticated cooling solutions are required. Especially in a server farm, where many servers are installed on a small room, meeting the requirements on the cooling and air-conditioning systems is quite a challenge. Such a system with many servers and sophisticated infrastructure supporting their operation is also consuming a lot of power.

One way to handle that issue lies in the development of strategies which are able to decrease the power and energy consumption of a cluster. This includes the positive effects known from the results of energy saving measures for battery-buffered devices: lower power consumption meaning less heat being produced meaning smaller cooling infrastructure being required.

1.2 Problems & Solutions

Such savings in power and energy consumption are possible, because most of the time a server cluster doesn't need all of its capacities. The unused capacities allow to switch the nodes of the cluster or single components of it into an operating mode consuming less power during this time.

The challenge lies in determining these spare capacities, as well as in maximizing this time span, so that as many parts of the system as possible may operate in a low power mode as long as possible.

Maximizing this time implies a tradeoff between minimizing energy consumption and maintaining the service quality of the cluster. When e.g. considering a sudden rise of the incoming workload and the latency that emerges from switching between different power modes, it is possible that the cluster is not able to handle the incoming workload at the postulated rate. As a result, the quality of service reflected in response time, execution time, throughput and so on decreases.

1.3 Power Management Policies

Existing approaches are following exactly this scheme, considering the resources of a server (cpu time, load on network interface and harddisc) as the key capacities and trying to put them into a low power mode in as many servers of the cluster as possible without degrading the overall performance.

Therefore, each strategy determines currently unneeded resources by using certain algorithms and then decides on how many and which components are to be suspended into a low power mode or on which components are to be reactivated. Those algorithms and decisions are based on various methods, often in combination with some heuristics (e.g. in form of parameters) which turned out to be good for at least the particular testing environment. As a consequence, it is often not clear why exactly those algorithms work, why or how those heuristics were chosen and whether or not the results could be reproduced when using different system setups or workloads with different characteristics.

1.4 Contributions of this Work

There exist many different approaches to reduce the power consumption of a server cluster, but there are only a few general methods as a basis which are used to attain this goal. These are elaborated in this work, as are other important characteristics by means of which different strategies can be classified and analyzed. A survey on existing research is given, describing their algorithms and analyzing them with reference to these characteristics. Underlying assumptions as well as restrictions are tabulated as an outline.

A simulator is presented which allows to implement strategies from those existing publications. The testing environment defined by the system parameters such as energy consumption or startup-delay of a node is freely parametrizable as well as heuristics used in the particular strategy. As test input, real traces from server logs are available which were also used in other publications, thus allowing comparison with their results. As an example, two representative algorithms are implemented and analyzed in terms of their heuristics and functioning.

Though not being topic of this work, the simulator has been made flexible enough to allow the implementation and testing of new policies. Due to the object-oriented modular design, it is also possible to simulate heterogeneous clusters with sets of nodes with different characteristics.

The remainder of this work is organized as follows. Chapter 2 gives a general view on the topic, listing the basic methods used in every strategy and other factors which might be taken for characterisation. In chapter 3, a survey of existing publications is given, describing their strategy for power management in server clusters. Chapter 4 presents a simulator which is capable of implementing the algorithms and models described in the previous chapter. Chapter 5 deals with a detailed view on three important, representative publications. Their methods are analyzed in detail and classified according to the factors introduced in chapter 2. The implementation of their algorithms in the simulator are presented as an example. Experiments with these implementations are done, using the original algorithms and some modifications. Conclusions as well as a final glance on the results and possible future work are given in chapter 6 and 7.

Chapter 2

General Aspects

Before going into detail about basic methods of energy saving strategies, a general view on the topic is presented, explaining the concepts and ideas behind.

2.1 Concepts

In a general view, a server cluster consists of n ($n \geq 2$) servers (nodes) which are linked together in order to provide a service with a better quality then a single server could. This improvement in quality might be e.g. a lower execution time, better responsiveness or a higher overall throughput. To hide the exact structure of the cluster to the client, a frontend called *dispatcher* works as a layer between the user and the cluster. Incoming requests for the provided service are received by the dispatcher and then delegated to one (or more) nodes in the cluster (w.l.o.g. following explanations will assume delegation to only one node, meaning that one request is worked off by only one node). This node then works off the request and sends the result to the client either directly or indirectly by passing it back to the dispatcher. Figure 2.1 gives a graphical view on that schematic design.

Vast research work has been done on the decision by the dispatcher about which node should execute a request to maximize the quality of service. The overall performance is improved by *balancing* the load over the cluster, so that each node has as much reserves as possible. When a decision is to be made by the dispatcher where an incoming workload should be executed, it chooses e.g. the node with the lowest load to gain a maximum of performance.

The size of the cluster (i.e. the number of nodes) is typically determined by the expected maximum workload the cluster has to work off within certain performance parameters. If there is for example a cluster of Apache web servers hosting a website which should be able to cover a peak load of 3000 incoming requests/second and each node is capable of handling 300 requests/second within acceptable performance parameters (like latency), $n = 10$ such nodes could form the cluster and provide the capacities to handle the postulated peak load. Depending on how definite and ex-

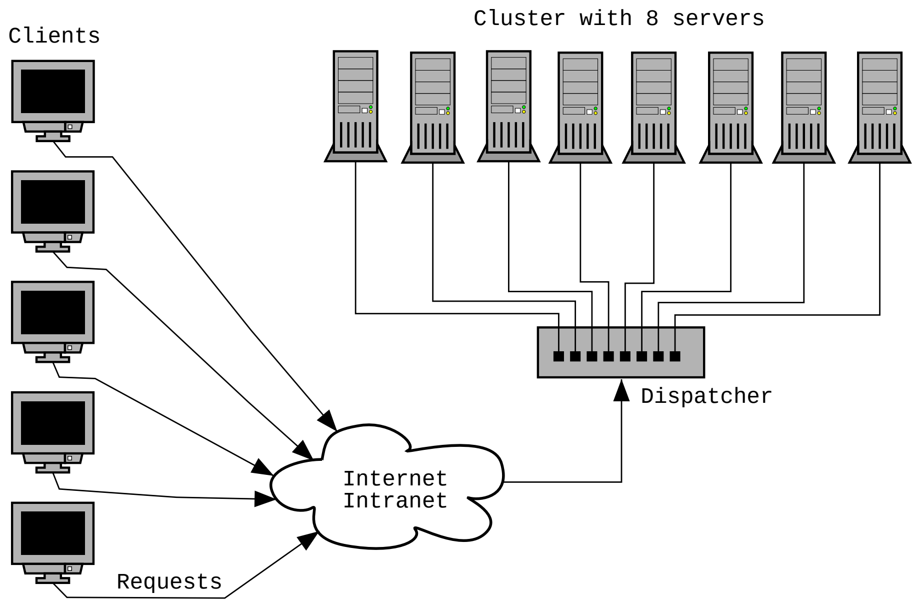

Figure 2.1: A server cluster with a dispatcher which receives incoming requests

act that peak load of 3000 is, one might want to have some reserves or to compensate a potential coordination overhead and use $n = 10 + x$ ($x \geq 0$) instead.

Anyway, in reality, that peak load of 3000 requests/second will be reached only for a limited time each day, and maybe not even every day of a week. So there are unused capacities that are unnecessarily consuming energy, and this is exploited by energy saving strategies. The potential gains are visualized in figure 2.2, visible in the difference between the capacities of a web server cluster consisting of $n = 10$ nodes and a typical graph of requests over one day.

An important remark here is that this assumption of resources being unused is generally not valid when it comes to computing cluster. Here, the computing time is very expensive due to the high purchase costs of the hardware. Thus a high utilization is sought and also reached in general. In contrast, a web server does have a load heavily depending on time of day. As existing publications take into account only web servers, this works view is also limited to this type of servers.

So, what is done by every energy saving strategy is to put some components of the cluster into an operation mode that is less effective but also consuming less power. Coming back to the example from above, this could e.g. be done by shutting down a complete node, which would result in a decrease of 300 Requests/second for the capacity of the cluster as long as the node is not working. Spoken metaphorically, the goal is to lower the graph of the cluster's capacity in figure 2.2 as far as possible without letting it drop below the graph of incoming requests. If that happens, there would be more incoming workload then the cluster could handle at the moment and hence the quality of service (e.g. latency, execution time) would decrease.

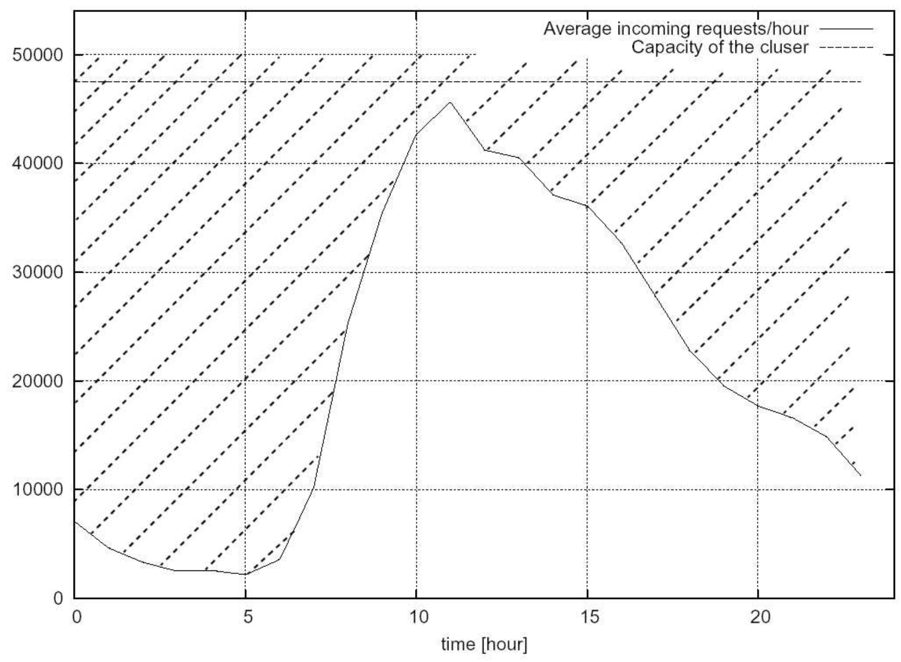

Figure 2.2: The capacities of a web server cluster and typical usage of one day, as taken from [16]. The hatched area marks the potential savings.

As an algorithm never knows *exactly* what workload the cluster will have to handle in the near future (except in a testing environment), it has to rely on guesses about it. These guesses are the basis for the decision about how many components might be put into a low power mode without risking to endanger the quality of service, i.e. letting the capacity graph drop below the workload graph. The better they are, the more efficient the algorithm can work and thus the more energy can be saved by it.

2.2 Classification of Algorithms

The strategies can be classified by a couple of characteristics, each one considering different aspects of the strategy, its algorithm and the data it uses for decisions. In the following, all of those characteristics will be listed and explained.

2.2.1 Quality of the Results

Obviously, the first characteristic crossing one's mind when comparing different strategies is the quality of the results it produces. The following characteristics of an algorithm determine its quality:

- how much energy does it save compared to a standard cluster running without energy saving strategies?
- how fast can it react and adapt to a sudden rise or decline in the rate of incoming workload?
- how universally valid is it, does it only work for very few environments or is it a generally good algorithm?
- how flexible is it, how many restrictions does it have concerning system parameters and behaviour, workload characteristics and such?
- on which assumptions does it rely, do they have to be fulfilled for the algorithm to work correctly?
- how often and how much does the quality of service drop because of the algorithm not being able to manage the cluster in a way that its capacity covers the needs all the time?
- how complex is it and how much overhead arises out of this complexity?

2.2.2 Principle Methods to Save Power

Another feature which allows to distinguish between strategies is the method it uses to put a single server into an operation mode that consumes less power. Basically, this transition from the current state to another consists in turning off some functionality or components in the system. That saves the power needed to provide that functionality and to keep the components running. It might be done for a single node only but also for more nodes, up to affecting the whole cluster.

On/Off Model

The most simple way to apply this principle is the *on/off model*, which means shutting down complete nodes whenever the decision is made that the remaining nodes are able to provide the quality of service. Incoming workload is not delegated to those eliminated nodes anymore and existing workload from them is appropriately migrated to the rest of the cluster if needed. Elsewise, the nodes finish their workload and then shut down.

In the same manner, whenever the decision is made to turn on one or more additional nodes, the existing workload is distributed adequately if necessary. One could also think of prefering the new nodes when distributing incoming requests as long as they have more spare capacities as the rest of the nodes in the cluster.

More fine-grained Models

Shutting down one or more complete nodes (turning them on) is a rather coarse-grained method to decrease (increase) the capacity of the system. A more fine-grained power management would consist in switching to additional operating states between ”fully operational” and ”off” which should consume less power but still provide some functionality/capacities. These additional states have to be supported explicitly by the system and might therefore not be available, whereas the simple on/off model can be implemented without any requirements.

This explicit support is e.g. provided by a system compliant to the *ACPI* (Advanced Configuration and Power Interface) standard [15] which allows OS-directed power management. The ACPI standard defines how to integrate power management into a system, offering a set of global states and transitions between them. The cpu as well as other devices are able to be set into a sleeping state differing in power consumption, wake-up latency and context conservation. This allows a more fine-grained power management which is able to adapt to a situation in a more flexible way.

Obviously, energy savings are possible due to the fact that a turned-off (or suspended) node doesn’t consume any energy (resp. less energy then a fully activated one, per definition of ”suspend” and the sleeping states of ACPI).

Special Case: Frequency Scaling and Voltage Scaling

Another special case allowing a fine-grained power management is *frequency and voltage scaling*.

Frequency scaling considers at the current cpu load and adjusts its operating frequency according to it, which means setting the lowest possible frequency that allows execution of the workload with adequate (i.e. acceptable) performance. The voltage V of a processor can be expressed as a linear function in its operating frequency f as $V = \alpha f$. So the power consumption is reduced by setting a lower frequency f.

As the energy consumption of a processor is usually proportional to V^2, power consumption can also be reduced by decreasing the operating voltage.

The overall power consumption is influenced by $V^2 f = cV^3$ (c being a constant factor), so the best savings in power consumption are attained by scaling frequency together with the operating voltage.

But in contrast to the on/off model, a certain base energy consumption will exist all the time, as the other components in a server have an energy consumption not depending on the frequency/voltage setting of the cpu. As a solution it is possible to combine voltage scaling with a more coarse-grained method like the on/off model, like e.g. described by E.N. Elnozahy et al. in [21] which will be further analyzed in chapter 5.

Processors supporting this technique are e.g. the Transmeta Crusoe [22], the Mobile Intel Pentium III with SpeedStep technology [8, 17] and processors supporting the AMD PowerNow technology [1].

2.2.3 Management

Whereas the previous classification is about *what* is done, one can also take a look at *how* this is done and how much information is needed.

Apparently, simply adjusting the voltage of a processor in one node according to its load does not need any information about the state of any other nodes in the cluster. So this is an *intra-node* power management mechanism, where no management or communication between the nodes has to be done.

Such an *inter-node* communication is necessary whenever an algorithm is based not only on the state of one node but of many (or all) nodes of the cluster. This is the case e.g. when it is required to collect the load of all nodes in order to decide whether shutting down a node is possible.

2.2.4 Decision Parameters

Another aspect revealing differences between the various strategies are the parameters by means of which the algorithm decides to take an action, like e.g. shutting down a node. Choosing different parameters for that decision directly affects the quality of its results, as already explained in section 2.1.

Examples for these parameters are:

- the rate of incoming requests/workload
- the excess demand, meaning the difference between the capacity of the cluster and the incoming workload
- the load of various components in the cluster, e.g. of the whole cluster; of the cpu, harddisc or network interface of the single nodes
- changes in the rate of requests or any other of the factors above

The last point implies another aspect under which the parameters can be used in different ways: the time from which parameters are being derived for making the decision. It is possible to take into account

- the current parameters,
- the history of the parameters, either a simple summary of past values or their first derivative, which would be the rate of change of the variables in a certain period of time, or

- the future values of the parameters, predicted by some arbitrary method or even known exactly.

2.2.5 Workload Characteristics

The last feature that allows to work out differences between a set of strategies are the workload characteristics for which the algorithm produces best results. Typically, an access log is taken (from a real server log or an artificially created one) and the requests/second rate derived from that log is sent to the cluster for the time interval the log stands for.

Different workloads may be distinguished by

- how generally valid they are, do they represent:
  - a normal, representative day
  - only a certain part of a day (only a very short time)
  - multiple days or even months like e.g. at an event like a World Cup
- features of the workload:
  - how big is the difference between maximum and minimum rate?
  - does the rate fluctuate a lot, i.e. are there many changes or is it rather stable?
  - how fast does the rate increase and decrease?
  - statistical variables: what is the mean value and variance?
  - how long/how fast does the time proceed that the log represents?

Depending on whether this workload is created artificially or taken from real access traces, the algorithm is valid for different scenarios.

2.2.6 Assumptions and System Parameters

Before one can test and judge a strategy, the particular testing environment has to be set up. For this purpose, a number of parameters have to be defined. The generated results of the strategy will usually differ when changing the environment.

Thus, a topic of interest is e.g. to find out about the strengths of a strategy in different scenarios, i.e. when does a strategy turn out to be a good choice, saving most energy. The main aim is to understand the algorithm and thus to find out on which parameters the respective benefits depend and how they influence the results for different settings.

The ultimate goal is to be able to choose the best strategy under given circumstances or to develop a new algorithm, combining the advantages of the existing ones without their handicaps.

The following variables are the extrinsic parameters[1] determining the outcomes. They are defined by K. Rajamani et al. in [19] as ”system-workload context”:

System Characteristics

- The smallest unit of the systems resources that can be added/removed, called *cluster unit* by K. Rajamani et al.:

  This means if turning on/shutting down a complete node is the most fine-grained way to increase/decrease the clusters capacity and thus its energy consumption, the cluster unit is a single server. If the nodes allow more power states than just ”on” and ”off” by e.g. being compliant to the ACPI standard or allowing voltage scaling of the processor, the cluster unit would be the transitions between the various operating states. In this case, turning a cluster unit on would mean to switch from a sleeping state to one with a higher activity level resp. increasing the processors frequency.

  A basic assumption is that turning a cluster unit on provides a certain additional resource capacity (compare next point) to the cluster for the cost of some power consumption.

- The *capacity* of a cluster unit:

  The maximum load that can be handled within acceptable performance parameters, measured e.g. in terms of the number of active client connections.

- The *Energy consumption* of the system, consisting of energy consumed by

  – a single cluster unit
  – the system with minimal configuration (*base energy consumption*)
  – the system during peak workload

- The *startup delay*:

  The time needed from the moment a cluster unit is turned on to the moment the system can use the cluster unit’s capacities. If a cluster unit were be a single server, this would be the time a server needs to boot and start the required services. The shorter the startup delay is, the faster an energy saving strategy can react to sudden spikes in the incoming workload. Again, voltage scaling is a rather fine-grained method, as adjusting a processors frequency takes almost no time and thus has virtually no startup delay.

- The *shutdown delay*:

[1]In contrast to the *intrinsic* parameters which are the ones in the algorithm itself determining its behaviour, the *extrinsic* parameters are defined by the used system platforms and input data

The time between removing a cluster unit from the system and actually shutting it down. This results in a time where the cluster unit is still consuming energy but does not add to the system's capacity anymore. Larger shutdown delays reduce the gains for small minima in the workload graph.

- A *correlation* factor:

  It describes to which degree the cluster units are depending on each other. A high correlation means that an energy saving strategy cannot be very flexible, as it has to consider the dependencies between the cluster units. This might e.g. be given by data being partitioned across the cluster which would require re-partitioning when a node has to be shut down. An inter-node communication (compare section 2.2.3) is required to handle a highly correlated system.

- *Migration*:

  The ability of a cluster unit to migrate or terminate the workload it is currently executing. If e.g. a node is not able to migrate its connections or terminate the corresponding requests, it cannot be shut down until all requests are worked off. This is especially important when executing a request takes a long time.

Of course it is assumed that cluster units can be turned on and off, elsewise an energy saving strategy cannot work at all. In addition, it is generally assumed that the nodes are homogeneous ones, as the increase in complexity when using heterogeneous nodes would be too large.

Workload characteristics

- *workload unit*:

  Defined analogously to the cluster unit, the minimal request that can be worked off by the system. The workload unit may differ in its characteristics such as the time and resources it takes to work it off, resulting in different impacts on the load of the system. It is important to know these impacts, if not in detail then at least by means of average values. A workload unit can e.g. be a single client connection.

- The variance of the workload defined as

  $$load\ ratio = \frac{average\ load}{peak\ load} \tag{2.1}$$

  A high load ratio near 1 stands for a quite constant workload graph without high spikes. Such a workload situation does not leave much room for optimization by means of energy consumption, as not many cluster units can be turned off over time. In contrast, a load ratio near 0 corresponds to a low average load in combination with a high peak load. For this scenario a lot of energy can potentially be saved.

- The rate of change in load in relation to the current load.

  If the rate of incoming workload is rising, the system has to work it off using the current capacities as long as new cluster units are fully operational (cmp. startup delay). As the current capacities depend on the current load, there is an upper bound of how much more workload the current configuration can work off with acceptable performance. For example, if the system is currently handling 3000 requests/second, an increase in incoming workload by 300 requests/second to 3300 requests/second does not affect the system much, as those 300 additional requests/second mean only a 10% higher load. In contrast, if the system is in a state capable of handling only 300 requests/second, such an increase by another 300 requests/second will have a great impact on the system's performance, as it means an increase of the load by 100%.

The effects of the interplay of these system-workload characteristics are further described in chapter 5 when different policies are evaluated.

Chapter 3

Related Work

There are currently only a few papers dealing with energy efficiency in server clusters, each one presenting a different approach. Chapter 2 has already given a general view on those strategies, presenting basic methods and characteristics. Now, a general overview on existing work will be given, describing their particular approach. Three important publications will be further analyzed and explained in chapter 5.

P. Bohrer et al. - The Case for Power Management in Web Servers [5]

P. Bohrer et al. did a detailed analysis of existing traces and log files. They examined them by means of different characteristics such as the average number of requests/second and peak requests/second. Thereafter, sophisticated power consumption measurements were done to determine the components of a system offering the most potential energy savings.

The cpu turned out to be the best candidate for this purpose. So, P. Bohrer et al. wrote an accurate simulator in order to examine the effects of voltage scaling in terms of saved energy. The energy consumption savings possible with voltage scaling were considered to be significant.

As a result, a new performance metric is introduced, no longer measuring the performance of a web cluster by means of connections/second but of connections/second/watt.

M. Elnozahy et al. - Energy Conservation Policies for Web Servers [11]

Three different policies are presented by M. Elnozahy et al. in [11], being based on dynamic voltage scaling (described e.g. in [24]) and a newly developed method called *request batching*. Both methods are chosen as the cpu is considered as the dominant energy consumer in a system, so the assumption is that most energy savings can be gained by increasing the efficiency of the cpu usage.

The operation principle of request batching is to group incoming requests together and execute them in batches. Between those batches the cpu is set into a low power mode, i.e. into *deep sleep*, a technology available in Intel Pentium 4 processors [9].

By analyzing these two methods and the combination of them, it turned out that request batching is most effective under light workload, whereas dynamic voltage scaling is most effective for more intense workload situations. The analyses were done with a simulator called *SALSA* and various real traces, such as e.g. one of the Olympics 98 web server access logs as well as a trace from a financial service.

R. Bianchini - Research Directions in Power and Energy Conservation for Clusters [4]

Bianchini gives a rather general view on the topic in [4], presenting a rather elaborated motivation on the research topic based on costs for cooling infrastructure, backup cooling and backup power-generation equipment as well as the effect on the environment. In this paper, the general problem is described by a very general and simple formula:

$$Energy = \sum_{n}^{nodes} (P_p^n \times T_p^n + \sum_{r}^{resources} ((\sum_{m}^{modes} P_{rm}^n \times T_{rm}^n) + M_r^n)) + P_s \times T_s \tag{3.1}$$

where $P_p^n \times T_p^n$ is the fix energy consumed by each node's power supply for the time T_p^n, $P_{rm}^n \times T_{rm}^n$ is the energy used by resource r of each node operating in state m for the duration of T_{rm}^n. M_r is the energy needed when switching between different operation modes and $P_s \times T_s$ is the energy consumed by the communication switch as long as it is working.

As a result, power and energy savings can be achieved by reducing the time the resources operate in modes consuming much power and increasing the time they operate in low power modes. Alternatively, savings are possible by reducing the power consumed by the power supplies, resources and communication switch. As a consequence, the postulated goal is - besides things out of reach for a cluster administrator like using more power-efficient hardware components (which would just mean to buy new hardware) - to schedule the given load in a way that allows an idle power mode for most resources.

The methods presented for reaching that goal are given in a very general and abstract way: Bianchini suggests load-concentration, but gives only pseudo-code for an algorithm, without any further specifications about parameters/thresholds and such. In principle, this algorithm aims at putting as many resources as possible into a low-power mode for the longest time possible.

Bianchini also points out that it is important to activate nodes in time to avoid performance degradations. In addition, he proposes to specialize the functionality of the nodes in a statical way. This should result in an explicit segregation of functionality that is frequently used and that which is rarely used, which would allow to set nodes with rarely used functionality into longer periods of low-power operation states.

As the values for power consumption are the same as those used by Bianchini et al. in [18], it is likely that the algorithm and its potential energy savings are the same or at least similar as the one being described in detail in [18]. So, a look at E. Pinheiro et al. - Dynamic Cluster Reconfiguration [18] in 5.2 will give a deeper insight into the actual algorithms.

J. Chase et al. - Balance of Power: Energy Management for Server Cluster [7]

Another quite elaborated motivation for the subject of energy saving strategies in server clusters is given by J. Chase et al. in [7]. They formulate the problem as a minimization of an energy-per-unit-of-service metric, called *joules per operation* (JOPs). Four different ways are presented how this can be achieved:

- by decreasing the postulated quality of service;
- by making the hardware more efficient, i.e. by increasing its *cycles per joule* value (*CPJ*);
- by making the software more efficient, i.e. by reducing its *cycles per operation* value (*COP*); or
- by reducing the *capacity costs* of unused components by putting them into an idle mode consuming less power.

These four aspects can be put into one simple formula in the following way:

$$Powerdemand = (OPS * COP)/CPJ + \sum_i p_s(i)$$

where components i power consumption in operation state s is $p_s(i)$ and OPS are the operations spent on the service (reducable by using the service less).

The goal of an energy saving strategy should be to make energy consumption proportional to the load by increasing the energy efficiency of the system during periods of low load without reducing the service quality.

Chase et al. point out that saving energy implies a tradeoff between energy consumption and latency, as saving energy by putting servers into a low-power state increases the overall latency of the system.

The presented method to save energy is to hold the average utilization of the cluster near a configurable threshold T. The number of active nodes is determined by comparing the average utilization with $TN/(N-1)$ (N being the current size of the active set of nodes). If the average utilization falls below this value, a node gets turned off. If it approaches T, a node is reactivated.

K. Rajamani et al. - On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters [19]

This publication identifies the key system and workload factors that heavily affect energy efficieny of request distribution schemes. Interactions between those factors and how they affect energy consumption is analyzed and explained in a brief way. A sophisticated industry-standard benchmark is used to study three different methods which are presented as energy saving strategies:

- Simple threshold
- Spare servers
- History based

A detailed explanation and analysis are to be found in chapter 5.

M. Elnozahy et al. - Energy-Efficient Server Clusters [21]

Elnozahy et al. present and analyze the effects of combinations of dynamic voltage scaling with an on/off model, called *vary on/vary off*. These two methods are now combined into a total of five different policies:

Independent voltage scaling (IVS) adjusts the frequency of each node separately, throttling each node down to the lowest frequency possible without performance degradations. Due to the load balancing, every node will operate roughly at the same frequency, although no inter-node management (compare section 2.2.3) is done.

Coordinated voltage scaling (CVS), on the other hand, does this additional management, by broadcasting the average operating frequency to each node periodically. Every node therefore restricts its frequency setting around that average value, resulting in a lower variance of the frequency distribution across the cluster. Lower variance should decrease the overall power consumption.

Vary-on vary-off (VOVO) implements the first method described in 2.2.2, concentrating the workload on the minimum number of nodes needed and shutting down the rest. Elnozahy et al. give no detailed explanation about how they decide on how many nodes are to be shut down.

IVS + VOVO is a combination of IVS and VOVO, which means shutting down as many nodes as possible and adjusting the frequency of the active ones.

CVS + VOVO is the combination of CVS and VOVO.

An in detail explanation and analysis are to be found in chapter 5.

J. Chase et al. - Managing Energy and Server Resources in Hosting Centers [6]

Another approach using *VOVO* (varyon/varyoff) as the method to reduce the power consumption is presented by Chase et al. The mechanism uses a quite sophisticated approach with an economic model as framework, where services ”bid” for resources as function of the delivered performance. A load monitor measures the performance and another module estimates the value of effects on service performance from resource distributions. Resource ”prices” are adjusted in a way balancing supply and demand, so that the resources are used most efficiently. A central executive section reallocates resources and adjusts the configuration of the system to handle varying load, resource availability and service value.

E. Pinheiro et al. - Dynamic Cluster Reconfiguration for Power and Performance [18]

E. Pinheiro et al. introduce a method that uses VOVO as well. With the control theory as basis they introduce and use a controller to adjust the value of *excess demand* by turning nodes on resp. shutting them down. Excess demand is defined as that part of the workload that could not be handled by the cluster within the given perfomance parameters. The usage of control theory provides a well-understood way to adjust the parameter ”capacity of the cluster” as near as possible to the graph of the incoming workload.

A detailed explanation and analysis are to be found in chapter 5.

Chapter 4

The Simulator

Reading related work soon reveals the problem of very different experimental setups being the basis for any measurements in the papers. An overall comparison of the algorithms can only be done when the same system platform (i.e. the same system parameters like e.g. cpu power, energy consumption and so on) as well as the same input data for testing purposes are used. Often, it is also not easy to understand the algorithms proposed for reducing power consumption because of various heuristics being used in them.

4.1 Basic Idea

The idea is to write a simulator where all those parameters can easily be changed independently from each other. It is then possible to implement different strategies but applying the same underlying system platform, so a comparison can be done. By now varying this uniform system platform, it can be evaluated if certain algorithms might be better in different environments. A second aspect of interest is taking one algorithm with a constant system platform and playing around with the heuristics in the algorithm. This gives a deeper insight into the algorithm, which might be especially useful when there are no detailed explanations given.

4.2 Design

The simulator basically consists of the following parts:

- a set of nodes representing the single servers in the cluster
- a load generator, that e.g. reads an access log file and passes that information to the cluster in form of an incoming request rate
- a dispatcher, which receives that load and distributes it to the nodes, controlling them according to some policy (e.g. turning them off)

A main goal is flexibility, i.e. it should e.g. be possible to replace the behaviour of the nodes easily in order to test the influence of different system platforms on the investigated algorithm.

Therefore, an object-oriented approach was chosen for implementation, so as to simplify separation of the different modules and therefore to increase the flexibility of the simulator. The different classes representing the above parts of the simulator will be described in section 4.3.

Basically, the simulator works with discrete steps in time called *cycles*. The log files used for generating workload (cmp. section 4.4.2) have a limit concerning temporal resolution, i.e. the most fine-grained information about incoming workload can be given in steps of one second. Thus, one cycle represents one second.

Figures 4.1 - 4.3 describe the principle functioning of the simulator: the simulator tells the dispatcher to begin one cycle (1). At the beginning of each new cycle the dispatcher requests new workload from the loadgen (2). It then gathers informations like e.g. average cpu utilization or current cpu frequency from each node (3) and uses it for determining the new configuration of the cluster according to the power management policy it implements. In order to achieve that new configuration, the dispatcher sends commands to the single nodes, telling them e.g. to power down or increase its cpu frequency (4). Thereafter, the workload gathered for this cycle has to be distributed to the nodes where it gets stored in a queue (5). At the end of the cycle, all active nodes are told to work for one cycle and execute their stored workload (6).

In order to evaluate the effiency of the power consumption reducing policy, the dispatcher is able to request the power consumption of each node for the last cycle. Accumulating this value for each node and printing the sum onto stdout allows to plot a power consumption graph with a script using gnuplot for example.

4.3 The Classes

4.3.1 Node

This abstract class represents a single computer in the cluster. Every implementation has to provide the following methods:

- `void doCycle()`

  Tells the node to work off the requests it has in its queue. Note: In that abstract class, no further details are specified about how those requests get *into* the queue. This is done in the particular specialization.

- `float getPowerConsumption()`

  Returns the power consumption of the node for the last cycle.

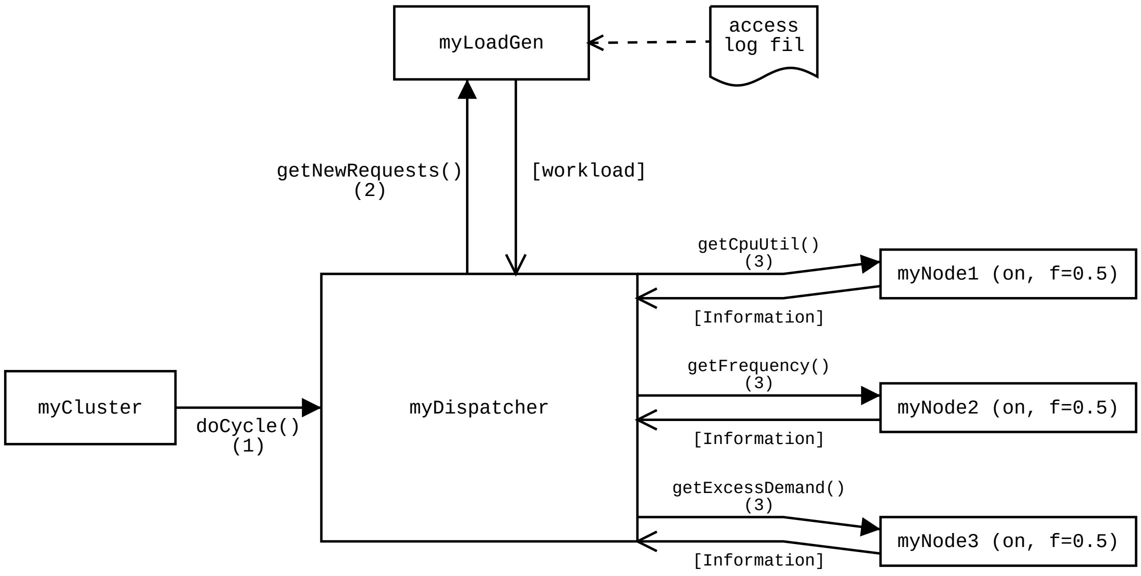

Figure 4.1: At the beginning of a cycle, the dispatcher receives workload from the loadgen and gathers informations from the nodes

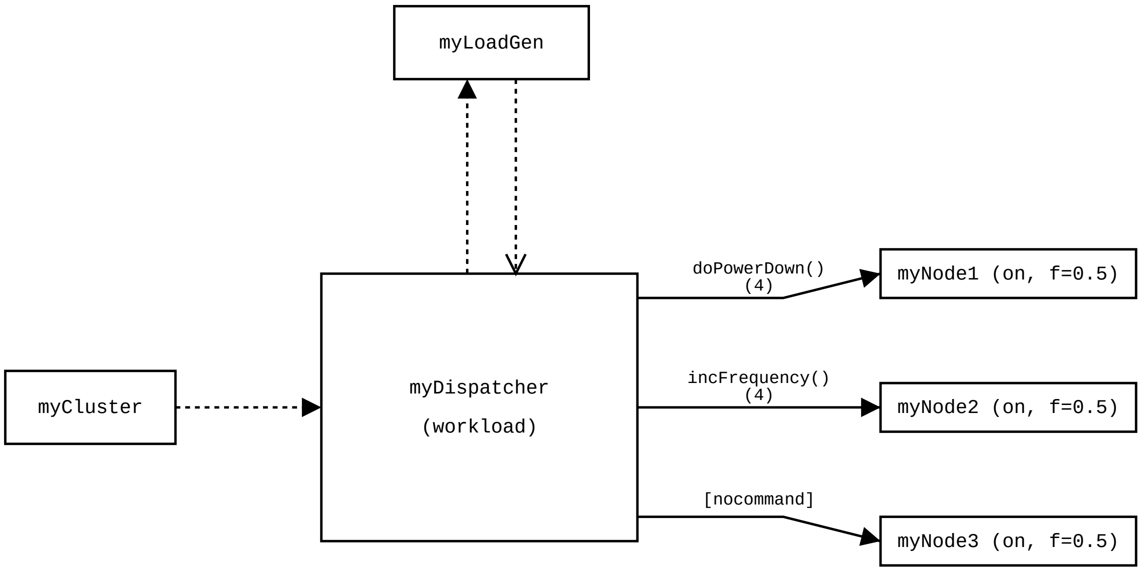

Figure 4.2: Then the dispatcher sends commands to the nodes to achieve a new configuration according to his policy

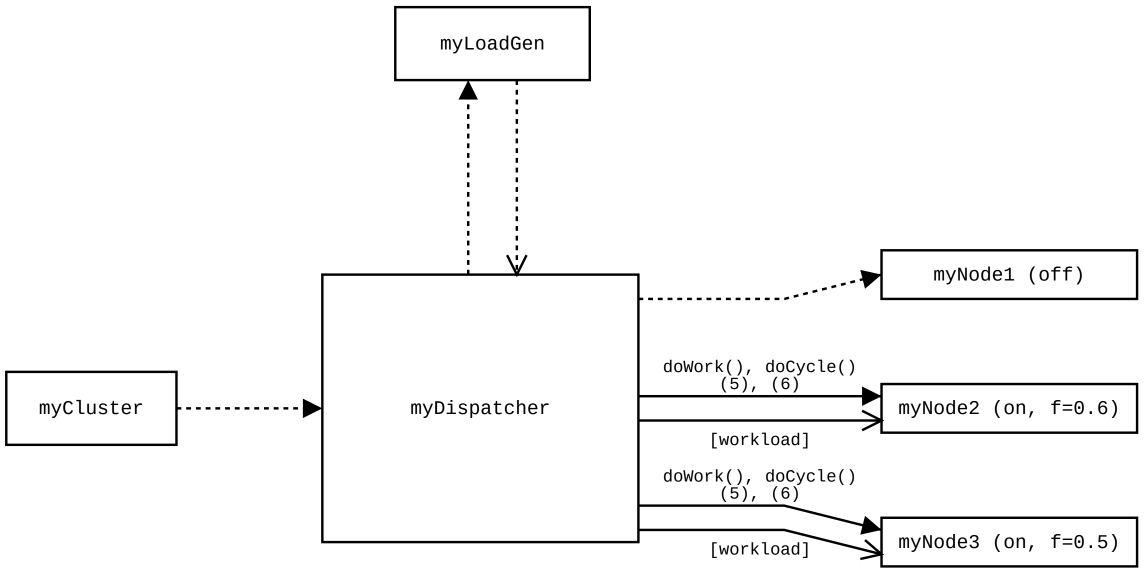

Figure 4.3: At the end of a cycle, the workload is passed to the running nodes and worked off by them

It is possible to use nodes with different characteristics such as capacity or the ability to scale the frequency together in a scenario. The *Dispatcher* (see below) then has to consider those differences in its policy. Thus it is possible to simulate heterogeneous clusters like e.g. described in [13].

4.3.2 LoadGen

The LoadGen reads in a log file and generates incoming requests for the cluster. It basically offers the following functionality:

- `int getNewRequests()`

  Returns the request rate for the next second out of the log file

The structure of the log file is explained in section 4.4.2.

4.3.3 Dispatcher

The Dispatcher is another abstract class, representing the part between the outside of the cluster from where requests are coming in and the single nodes in the cluster. Thus it handles a set of nodes according to some policy and receives incoming requests from the LoadGen. Depending on that incoming load, it manages the nodes (e.g. turning some of them on or off) and distributes the load onto them. Every dispatcher derived from this abstract base class has to implement the following method:

- `void doCycle()`

  Does one cycle, in which new requests are fetched from the LoadGen and passed to the nodes after some policy dependent decisions about the cluster configuration. It then tells the single nodes to do a cycle and work off the given load.

The main module is the *Dispatcher* which is fetching incoming *Requests* from the *LoadGen* and managing a set of *Nodes*. The dispatcher may make an arbitrary decision about which nodes should execute the workload and which nodes may be put into a low power state or turned off. For this purpose, he can retrieve informations from any node about its status, like e.g. the amount of unfinished work.

The algorithms presented by Pinheiro et al. [18] and Elnozahy et al. [21] were chosen to be inspected in detail (cmp. chapter 5) and were implemented for the simulator. This will also give a more detailed insight on how that program works.

4.4 Preparations

Before the simulator was written, some preparations had to be done.

4.4.1 Measurements

First, some measurements were made to get some realistic energy values as basis for a Node implementation. Therefore, an implementation of resource containers in the Linux kernel 2.5.58 by Martin Waitz [23] was used. It consists of a hierarchy of containers and is among other things able to account the energy consumption of the machine running it, in this case, an Intel Pentium 4 with 2GHz. Different test-runs were done with the program ”hammerhead” [14] on an Apache webserver [2] to determine the relation between incoming request rate (generated by hammerhead) and energy-consumption (exact measurements with the root resource container, representing the energy consumption of the whole system).

As result, a node representing an Intel Pentium 4 with 2Ghz can handle a maximum of 750 requests/second[1]. The complete relation between incoming request rate, served request rate and energy consumption can be seen in figures 4.4 and 4.5. This behaviour was applied in the implementation of the class representing a node.

[1]Note: as the peak load of the world cup 98 trace introduced in the next section is only 1800 requests/second, for all the experiments nodes with a capacity of 500 requests/second were taken to allow some more variations of the active set of nodes

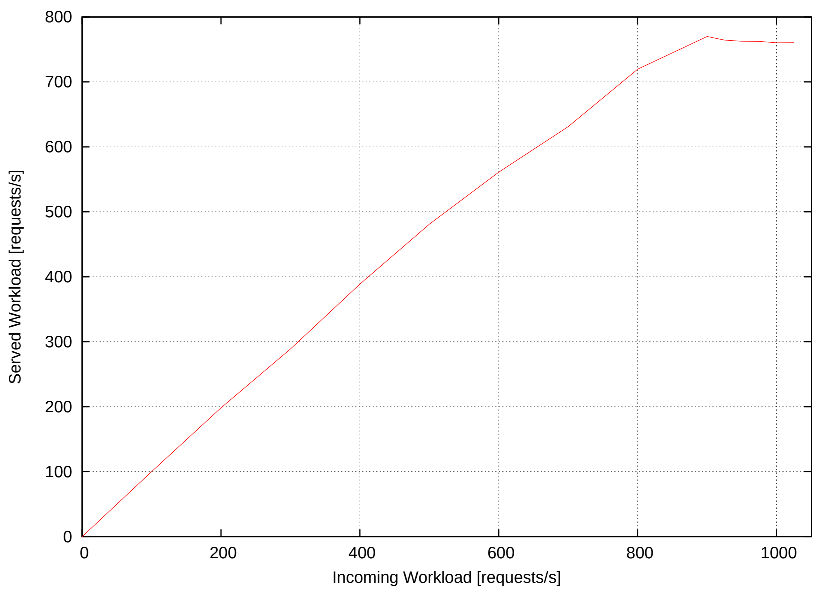

Figure 4.4: The number of requests a 2GHz Intel Pentium IV can serve for a given workload

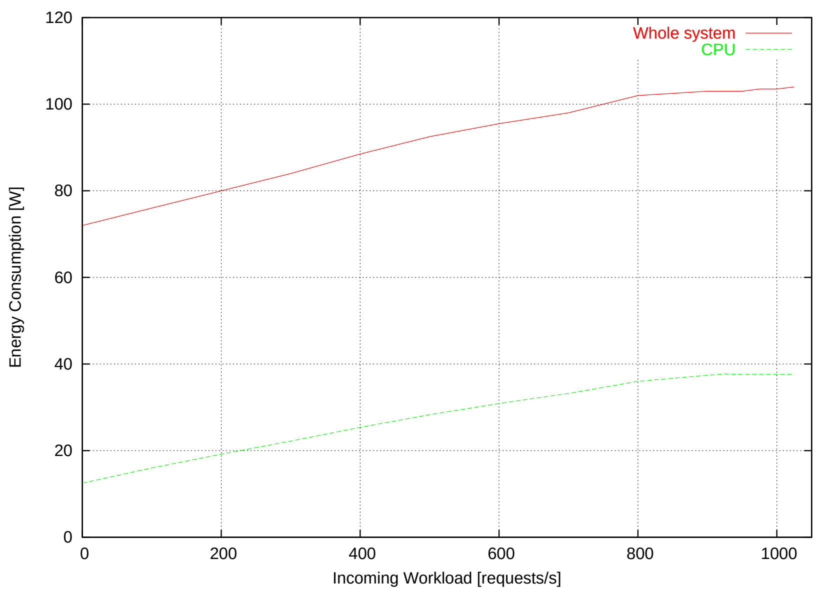

Figure 4.5: The energy consumption of a 2GHz Intel Pentium IV for a given workload

4.4.2 Input Data

The test input used by Pinheiro et al. [18] was the trace of accesses to the World Cup ’98 site from 12pm on 07/12 to 12pm 07/14, so those logs were taken for this work as well (available at [3]).

For reasons of performance, these logs were converted from a detailed format including various informations about every request into a new format only containing the request rate for every second.

The original log file format contained structs consisting of:

```
struct request {
  uint32_t timestamp;
  uint32_t clientID;
  uint32_t objectID;
  uint32_t size;
  uint8_t method;
  uint8_t status;
  uint8_t type;
  uint8_t server;
};
```

This structure contains a lot of additional information about each request like e.g. a client ID or the http protocol version that are not needed to determine the request rate. In addition, to get the request rate for one instant in time one has to count all requests with the respective timestamp. In order to optimize this, a program called ”`logconvert`” was written that converts the large original log files containing single requests into a smaller log file containing only the request rate for each point in time.

```
struct requestrate {
  uint32_t rate;
};
```

Thus it is possible to use any other access log file with the simulator by converting it into the simple format.

4.5 Implementation

The central part of course consists of specialized Dispatcher classes, which work according to the algorithms described in [18] and [21]. Before describing the implementations any further, the algorithms themselves will be explained shortly. A detailed description of the algorithms and the mathematical models behind is to be found in chapter 5.

4.5.1 Vary-on/Vary-off Dispatcher

The algorithm from [18] takes into account the demand for resources at each point in time as well as the past history of demands and the speed of change in demands. It uses a control theory approach with a PID controller to adjust the capacity of the cluster (i.e. the number of active nodes) to the needs of the incoming workload.

A feedback controller for each resource is the basis therefor. As the workload on the network interface was the bottleneck through all tests (according to [18] pg. 11), this work only takes the network load into account as resource, and so has only one controller, which is used for every cluster reconfiguration decision.

This functionality is packed into a new Controller class, which gets continuously updated about the excess demand, that is, the number of requests the cluster couldn't work off during the last cycle. The controller thereof calculates the values needed for the proportional, integral and differential components. If the output exceeds a certain limit, one or more nodes get turned on resp. shut down. These thresholds and calculations how many nodes are to be turned on or shut down are based on heuristics according to the author.

Pseudo code for this VOVO_CT[2] dispatcher:

```
get new workload
calculate current capacity of the cluster and thereof current excess demand
update the controller with this new value
every 10 seconds do
    if (|output of the controller| > threshold)
     and (last reconfiguration was at least 60 seconds before)
        trigger reconfiguration:
        if (output of the controller < 0)
            remove_nodes(heuristic value)
        else
            add_nodes(heuristic value)
```

As an output the dispatcher prints values like e.g. the number of nodes or the power consumption of the cluster on `stdout` for each discrete point in time. These values can be analyzed by e.g. plotting them with `gnuplot`.

Measurements using this implementation with some different values for the parameters are to be found in section 5.2.3.

[2]In the following, the abbreviation *VOVO_CT* will stand for the algorithm using VOVO with the control theoretic approach given by Pinheiro et al.

4.5.2 Vary-on/Vary-off Dispatcher with Additional Voltage-Scaling

The algorithms described in section 3 which are presented in [21] by M. Elnozahy et al. were chosen as a second example. The implementation of all five combinations is not possible as the authors only describe the exact algorithms for *Independent Voltage Scaling* (IVS), *Coordinated Voltage Scaling* (CVS) and the most complex version, *Coordinated Voltage Scaling with Vary-on/Vary-off* (CVS+VOVO). Any threshold or such for *Vary-on/Vary-off* (VOVO) and hence also for *Independent Voltage Scaling with Vary-on/Vary-off* (IVS+VOVO) are not given. In addition, IVS and CVS produce the same results, as the single requests of the incoming workload do not differ in execution time like they would do on a real cluster.

The values needed to implement IVS (and CVS) are the thresholds of the parameters used to decide when to turn on a node or shut down a node. The algorithm presented by Elnozahy considers the average cpu utilization for this purpose and increases the frequency setting of a node if it rises beyond 95% resp. decreases the setting if it drops below 80%.

Pseudo code for the IVS dispatcher:

```
get new workload
every 10 seconds do
    for each node do
        if (average cpu utilization < 80%)
            decrease frequency
        else if (average cpu utilization > 95%)
            increase frequency
```

Pseudo code for the CVS dispatcher:

```
get new workload
every 10 seconds do
    calculate average cpu frequency of all nodes
    broadcast average cpu freqyency to all nodes as limit
    for each node do
        if (average cpu utilization < 80%)
         and (cpu frequency-frequency step > limit-threshold)
            // i.e. target frequency is within frequency limit
            decrease frequency
        else if (average cpu utilization > 95%)
         and (cpu frequency+frequency step < limit+threshold)
```

```
            // i.e. target frequency is within frequency limit
            increase frequency
```

For the case of CVS in combination with VOVO a more sophisticated decision is necessary, as the capacity of the cluster can be increased by either turning on an additional node as well as by increasing the cpu frequency of the nodes already in use. As solution a limit is calculated considering the base power consumption of a node and the variable power component of the cpu (depending on the utilization). This boundary represents the frequency setting where it is better to turn on an additional node and decrease the frequency of the $n+1$ nodes which are already running rather then increasing the frequency of the n running nodes. This is also done for the case when the capacity of the cluster should be decreased and the dispatcher has to decide between shutting a node down in combination with a frequency increase of the remaining $n-1$ nodes or a frequency decrease of n nodes.

Pseudo code for the CVS+VOVO dispatcher:

```
get new workload
every 10 seconds do
    calculate average cpu frequency of all nodes
    broadcast average cpu freqyency to all nodes as limit
    for each node do
        if (average cpu utilization < 80%)
         and (cpu frequency-frequency step > limit-threshold)
            // i.e. target frequency is within frequency limit
            decrease frequency
        else if (average cpu utilization > 95%)
         and (cpu frequency+frequency step < limit+threshold)
            // i.e. target frequency is within frequency limit
            increase frequency
    calculate new average cpu frequency, f_varyoff and f_varyon
    if (average cpu frequency <= f_varyoff)
     and (last reconfiguration was at least 60 seconds before)
        remove one node
        for each remaining node do
            set_frequency(old_frequency * n/(n-1))
    else if (average cpu frequency >= f_varyon)
     and (last reconfiguration was at least 60 seconds before)
```

```
add one node
for each node do
    set_frequency(old_frequency * n/(n+1))
```

As an output the dispatcher prints values like e.g. the number of nodes or the power consumption of the cluster on `stdout` for each discrete point in time. These values can be analyzed by e.g. plotting them with `gnuplot`.

Measurements using this implementation with some different values for the parameters are to be found in section 5.3.3.

Chapter 5

Analysis

Before describing three representative policies for power management individually in detail, a general overview is given, categorizing them according to the characteristics presented in section 2.2. Thereafter, each strategy will be presented, explained and analyzed, including the respective assumptions, requirements and theoretical models behind.

5.1 Overview

All three power management policies have the following characteristics in common:

- They all require a cluster with homogenous nodes and a correlation factor of 0.
- The workload unit is one incoming request, the exact capacity of a node by means of workload units is not provided.
- Only scenarios with webservers are considered, meaning that no migration of currently executed workload is required when shutting a node down, as each execution lasts only a very short time.
- The exact traces of accesses used to evaluate the policies are not available, so a detailed characterization of the workload can not be done.

The other characteristics according to section 2.2 of the policies presented in [21, 18, 19] are described in table 5.1.

Feature	Energy-Efficient Server Clusters [21]	Dynamic Cluster Reconfiguration for Power and Performance [18]	On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters [19]
Quality of the results			
Energy savings	from 19.4% (IVS) to 49.7% (CVS+VOVO)	up to 45%	from 7.1% to 45.6%
Restrictions and requirements	Voltage scaling is required	none	knowledge about the workload is required for history based policy
Quality of service	does not drop acc. to paper	is allowed to drop within the limit set with the `degrad` parameter	does not drop acc. to paper
Principle method to save power			
	Voltage Scaling and Vary-On/Vary-Off	Vary-On/Vary-Off	Vary-On/Vary-Off
Management			
	intra-node and inter-node management	inter-node management only	inter-node management only
Decision parameters			
Parameter	Average cpu utilization and average cpu frequency	excess demand for a resource (e.g. cpu utilization, network load)	load on the system (cpu utilization)
Timeframe	current value and average over a set of recent intervals	current value, accumulation over a set of recent intervals and rate of change	current value and rate of change
Outlined system (cmp. section 2.2.6)			
Cluster unit	one discrete step in cpu frequency as fine-grained unit, one node as coarse-grained unit	one node	one node
Capacity of the cluster unit	capacity differences gained by changing frequency and capacity of one node	capacity of one node	capacity of one node
Energy consumption of one node (idle/peak load)	13.5W/36.2W	70W/94W	13.3W/14.8W
startup delay	30s	100s	30s
shutdown delay	30s	45s	<1s

Table 5.1: Characteristics of three representative policies; IVS = Independent Voltage Scaling, CVS+VOVO = Vary-On/Vary-Off in combination with Coordinated Voltage Scaling

5.2 E. Pinheiro et al. - Dynamic Cluster Reconfiguration for Power and Performance

In [18], Pinheiro et al. present an approach similar to the VOVO algorithm in [21], decreasing the power consumption in a cluster by shutting down as many nodes as possible (with an acceptable degradation in the throughput rate). But unlike Elnozahy et al., he gives some detailed heuristics about when to shut down/turn on one or more nodes and how to re-distribute the existing load (if necessary).

5.2.1 Control Theory

A control-theoretic approach is applied to decide when to shut down/turn on nodes in the cluster, considering the incoming workload as well as the power and performance of different cluster configurations. For this the policy considers the demand for resources instead of the utilization of resources. In order to decide how many nodes should be shut down or turned on, a *controller* does the following calculation every 10 seconds:

$$o(t) = k_p e(t) + k_i \sum_{0}^{t} e(t) + k_d \Delta e(t)$$

The output $o(t)$ is depending on the *excess demand* with respect to the current configuration, considering the current excess demand ($e(t)$), accumulated excess demand over time ($\sum_0^t e(t)$) and the rate of change in excess demands ($\Delta e(t)$). A reconfiguration of the cluster is triggered if $o(t) > 0.55C$ (C being the capacity of a single node). The number of nodes that will be shut down or turned on is described by:

$$n = \frac{o(t) - 0.5C}{C}$$

It is possible to adjust the policy by setting two parameters. The first one, *degrad* is determining how much performance degradation is allowed. For example, $degrad = 120\%$ would mean calculating with $C' = 1.2 \times C$, increasing the capacity of a node at the cost of performance.

The second parameter is called *elapse* and allows a cluster reconfiguration only if elapse seconds have past since the last reconfiguration in order to avoid rapid changes.

5.2.2 Overview

The following assumptions are made:

- The cluster consists of homogenous nodes.

- The correlation factor (compare section 2.2.6) of the system is low, meaning that shutting down a node does not cripple the file system.
- A cpu running at reduced frequency takes longer to complete a given task (this increase in execution time is proportional to the frequency reduction).

The following system factors are described:

- Power consumption of an idle node is 70W and 94W if all resources are fully used.

Results:

- Savings in power consumption by 71% and in energy consumption by 45%.

5.2.3 Evaluation

As described in section 4.5.1, this policy was implemented in the simulator ($\rightarrow$ *VOVO_CT*). So it is possible to vary the parameters and test the policy with different workloads.

The access log file that was applied as test input by Pinheiro et al. is from the world cup 1998 and represents the accesses on the world cup site from day 48 12pm to day 50 12pm. It is available for download at [3]. Using it with the simulator presented in chapter 4 leads to the graph shown in figure 5.2. Compared to the graph presented in [18] (see figure 5.1), the requests/second rate is fluctuating very much. This is because Pinheiro et al. accelerated their trace by a factor of 20 and smoothed out short bursts of activity by using d' as request rate with $\alpha = 0.8$. [1]

$$d' = \alpha \times old\_demand + (1 - \alpha) \times current\_demand$$

Using that acceleration and smoothing in the simulator results in the graph shown in figure 5.3.

In figures 5.4 and 5.5, one can see the effect of smoothing the workload, even though a gaussian curve has no short bursts. Obviously, the smoothing produces better results both for the part where the workload is rising and where it is decreasing. If the workload is not smoothed before processed by the algorithm, the configuration of the cluster fluctuates a lot and the incoming workload often exceeds the capacity of the cluster.

Experiments where the world cup 1998 trace is used as input for the simulator are presented in section 5.5, when a comparison of all implemented algorithms is done.

Figures 5.6, 5.8 and 5.10 show how the *degrad* parameter affects the configuration of the cluster (capacity of a node is 500 requests/second). For the graph of the incoming requests/second rate,

[1]The reason for the difference in the general shape of the graph could not be worked out, one reason could be that Pinheiro et al. measured the load on the network interface but the access log used for the simulator contains only the number of requests per second, where 2 requests might differ in the load they inflict on a real system.

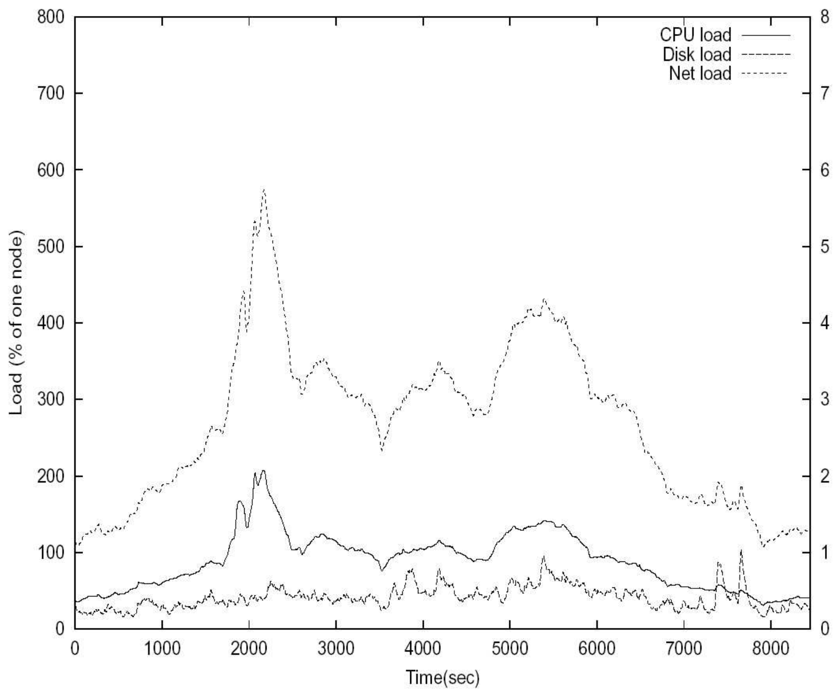

Figure 5.1: World cup 98 day 48-50 access graph used by Pinheiro et al.

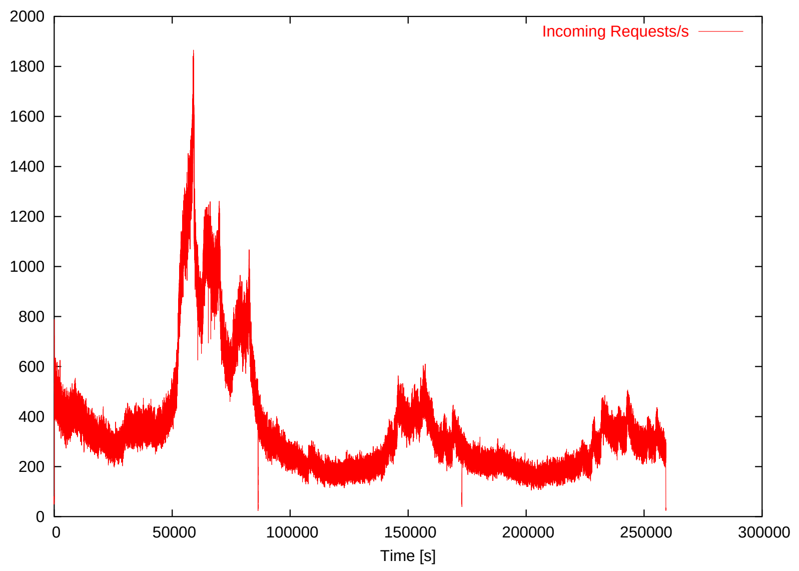

Figure 5.2: World cup 98 day 48-50 access graph produced by the simulator

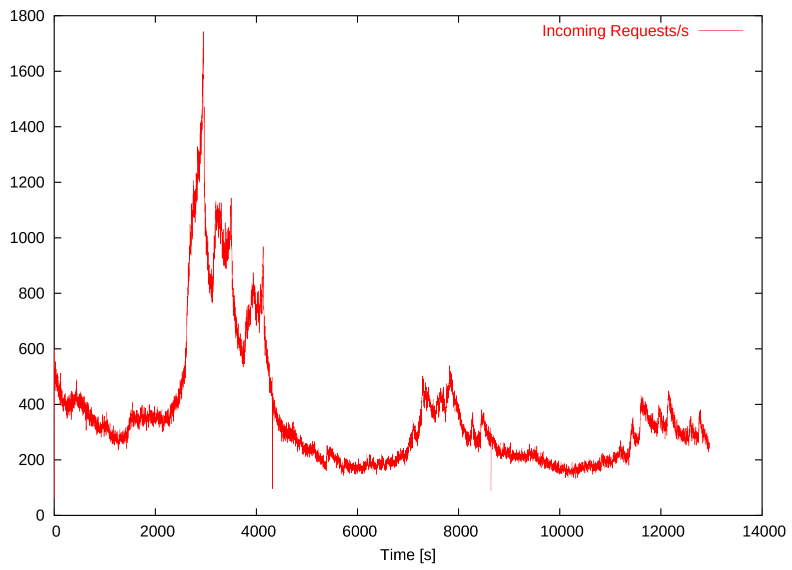

Figure 5.3: World cup 98 day 48-50 access graph produced by the simulator, using the same smoothing as Pinheiro et al.

a gaussian curve was taken so the effects can be seen more clearly. If a higher value is set for *degrad*, the graph of the cluster's capacity will more often drop below the graph of the incoming requests/second, resulting in performance degradations.

The energy consumption of the cluster for different values of *degrad* is shown in figures 5.7, 5.9 and 5.11.

These graphs were made using self-written scripts which compiled the simulator with different parameters and plotted the output of the resulting files with gnuplot [12]. The exact energy consumption values for the whole time of the experiment are as follows:

Degradation [%]	Energy Consumption [kJ]	Savings [%]
70	1566.94	40.53
85	1487.41	43.54
99	1393.82	47.10

It is clearly visible that the energy consumption drops when a higher degradation is allowed. But as seen in figure 5.10, the capacity of the cluster by far cannot match the needs of the incoming workload, e.g. at approx. second 750 when the algorithm decides to shut down node 2 again. Here, the capacity of the cluster is not big enough until approx. second 950 when 2 more nodes have

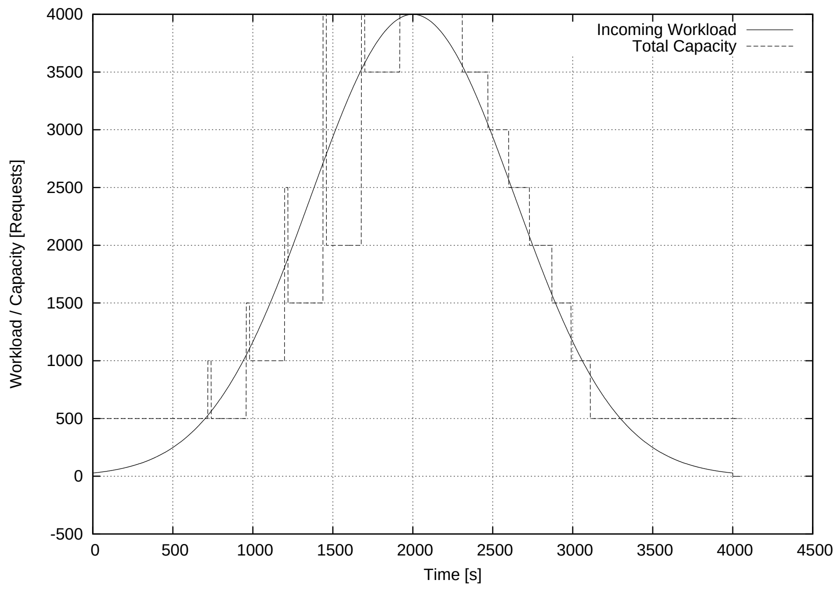

Figure 5.4: VOVO_CT on a gaussian curved workload without smoothing

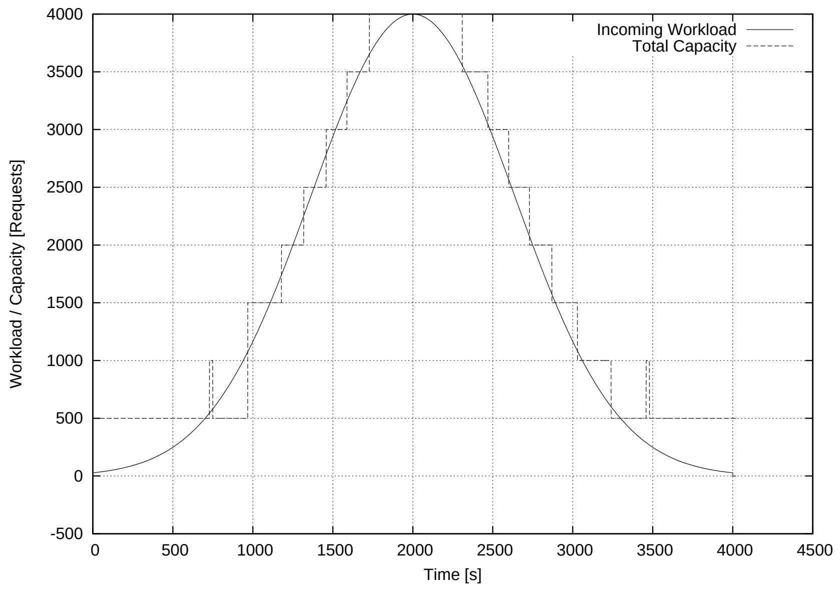

Figure 5.5: VOVO_CT on the same gaussian curved workload with smoothing

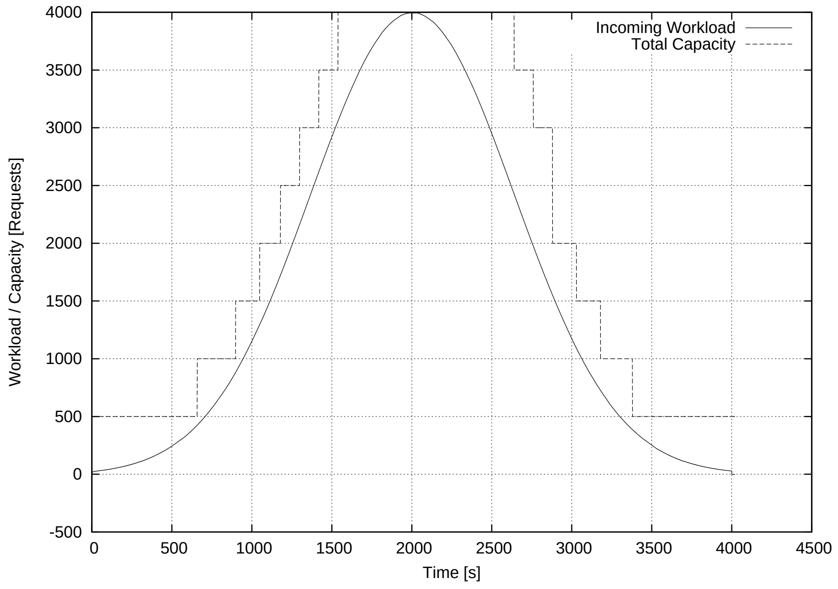

Figure 5.6: VOVO_CT: Effect of $degrad = 0.70$

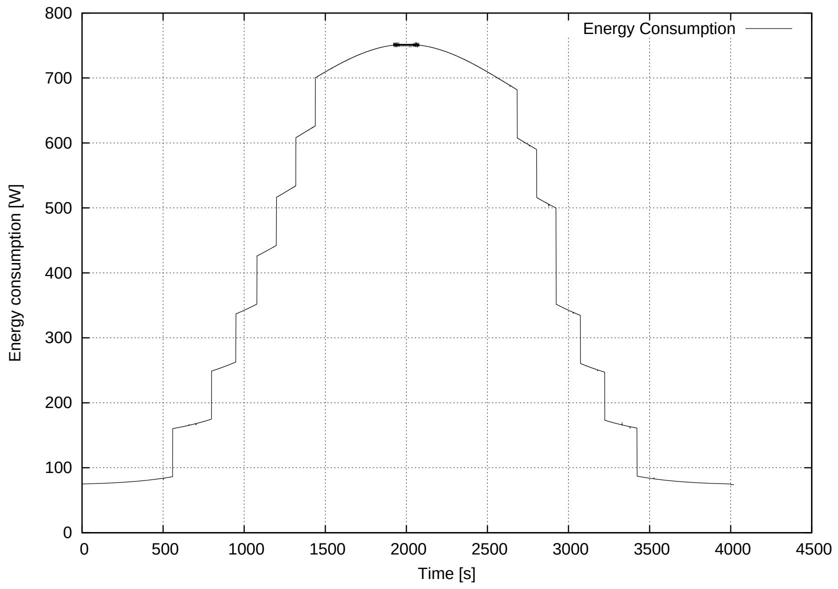

Figure 5.7: VOVO_CT: Energy consumption with $degrad = 0.70$

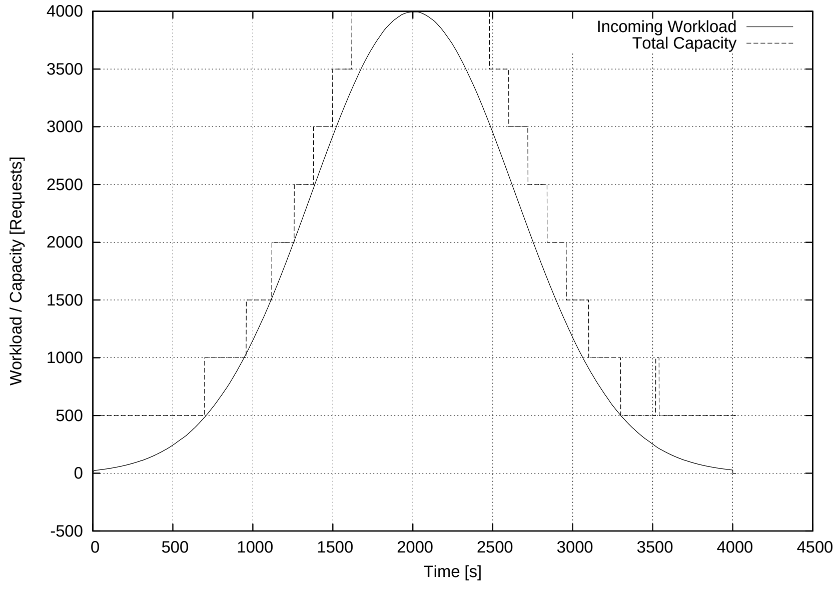

Figure 5.8: VOVO_CT: Effect of $degrad = 0.85$

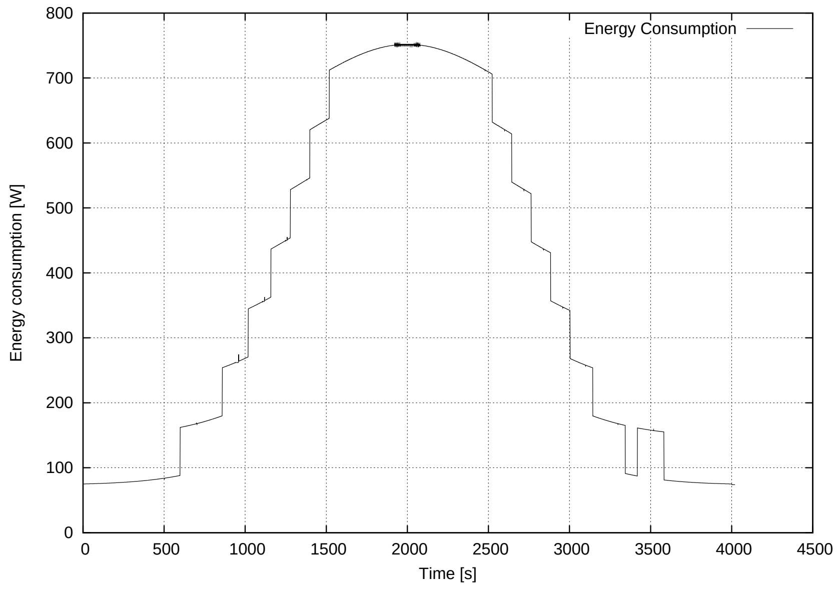

Figure 5.9: VOVO_CT: Energy consumption with $degrad = 0.85$

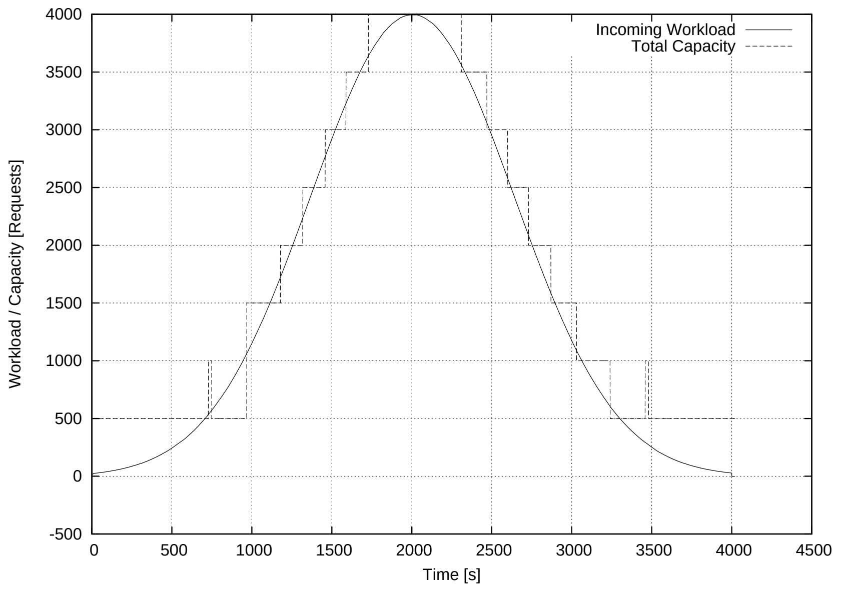

Figure 5.10: VOVO_CT: Effect of *degrad* $= 0.99$

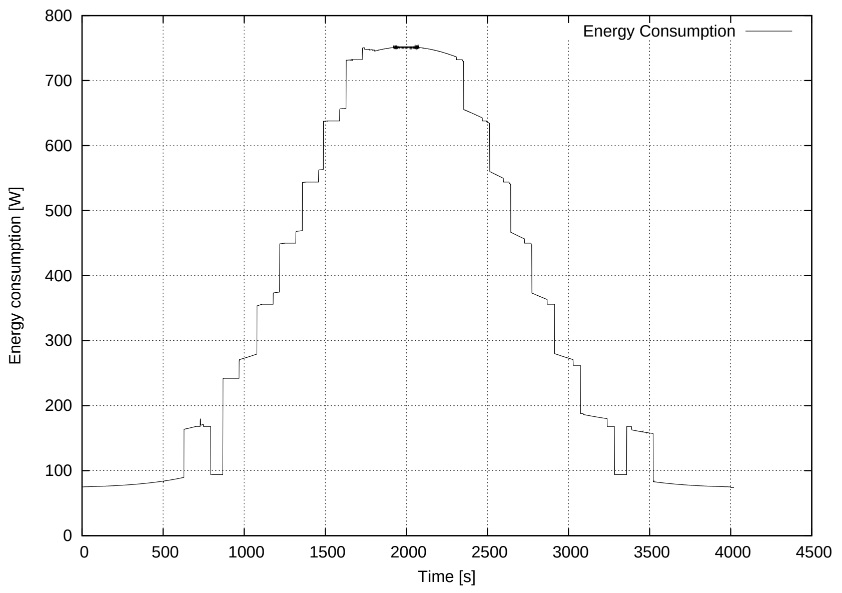

Figure 5.11: VOVO_CT: Energy consumption with *degrad* $= 0.99$

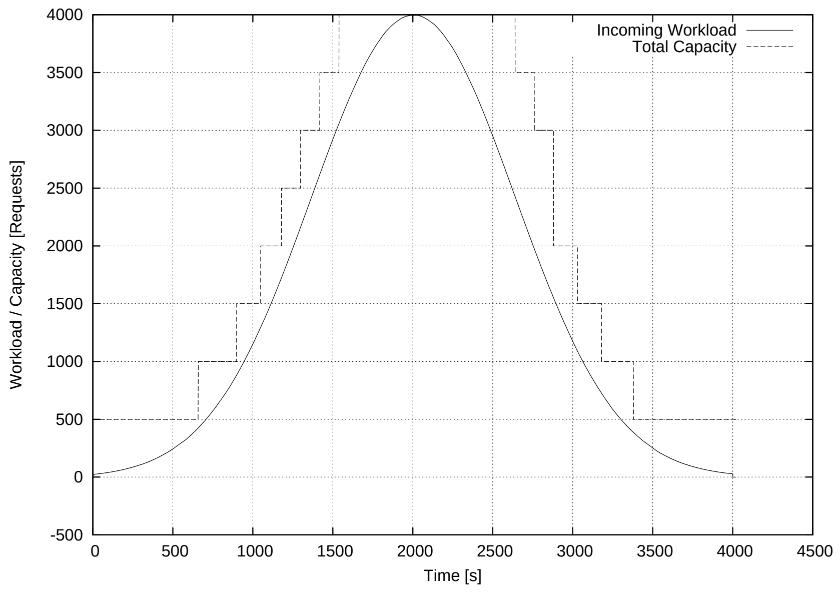

Figure 5.12: Behaviour of VOVO_CT with peak load after 2000s (startup delay=100s)

finished booting.

In three other experiments, the gaussian graphs for the incoming workload were accelerated in time from case to case, i.e. the peak load was reached after 2000 seconds (case 1, fig. 5.12) resp. after 1333 seconds (case 2, fig. 5.13) and after 666 seconds (case 3, fig. 5.14). So as the workload increased faster in case 2 than in case 1 and even faster in case 3, the algorithm had to adjust the cluster configuration in a different way for each case to achieve the required capacity of the cluster.

As the incoming workload is rising faster in case 2 than in case 1, the dispatcher in case 2 decides to turn on two nodes at once in approx. second 800 and provides enough capacities for the requests all the time.

But in case 3, the decision to turn on an extra node is made at a point where the workload is already increasing at a not too slow rate. As the dispatcher has to wait 120 seconds between 2 reconfigurations, it cannot add another node in time to cope with the more and more quickly increasing workload. In approx. second 310, it is possible to trigger the second reconfiguration and the dispatcher begins to boot two nodes which are fully operational in approx. second 410 (because the boot time of a node, i.e. the startup delay as defined in 2.2.6, is 100 seconds). As the load was unfortunately still accelerating in its increase, the $n = 4$ nodes running from approx. second 410 on

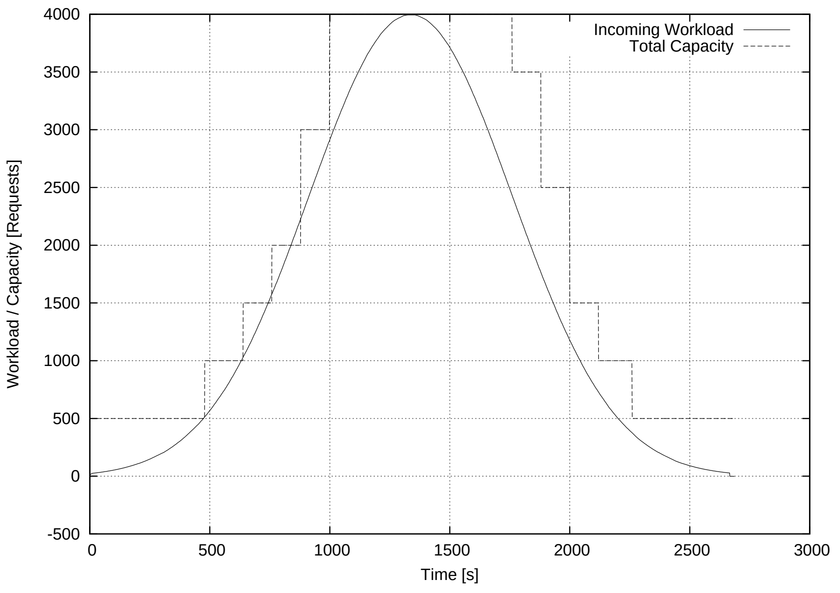

Figure 5.13: Behaviour of VOVO_CT with peak load after 1333s (startup delay=100s)

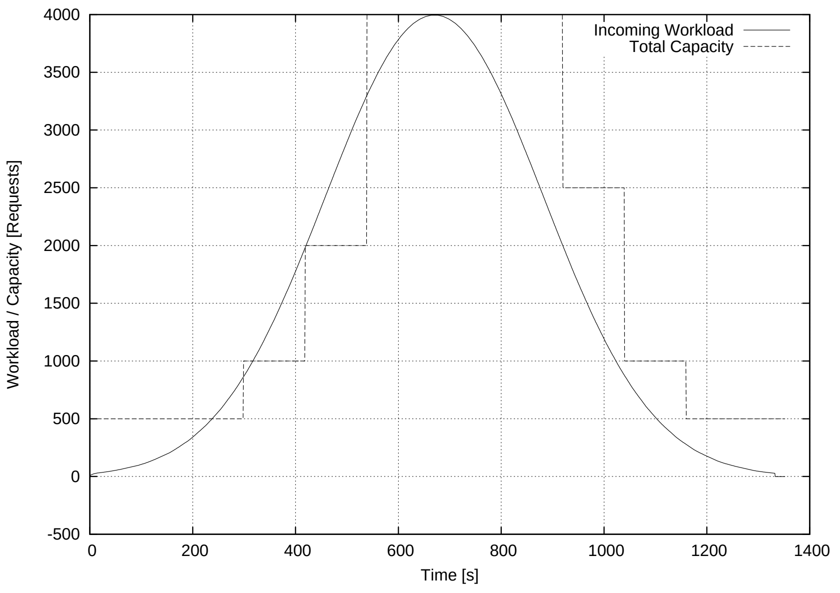

Figure 5.14: Behaviour of VOVO_CT with peak load after 666s (startup delay=100s)

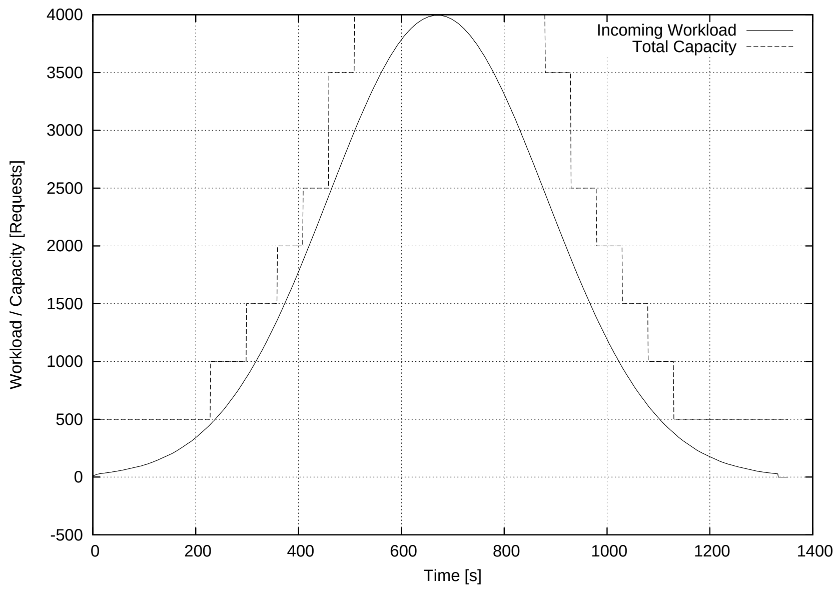

Figure 5.15: Behaviour of VOVO_CT with peak load after 666s (startup delay=30s)

are not enough but the dispatcher again has to wait another 20 seconds until approx. second 430 (since the last reconfiguration was done at approx. second 310) before more nodes can be brought online. So, when the dispatcher is again able to reconfigure the cluster to handle the incoming workload in approx. second 430, the decision is made to turn on the four remaining inactive nodes at once which finish booting in approx. second 530.

As a result it is clear that the algorithm is able to adapt to scenarios with faster rising workloads, but of course only within the limits of the system parameters. If the nodes did not need 100 seconds to boot but 30 seconds (an assumption by Elnozahy et al. in [21]), the algorithm is able to cope with a more rapid workload increase as it does not have to wait 120 but 50 seconds between each cluster reconfiguration (cmp. figure 5.15).

5.3 E.N. Elnozahy et al. - Energy-Efficient Server Clusters

Elnozahy et al. give a detailed approach in [21], evaluating five policies using dynamic voltage scaling, node vary-on/vary-off (VOVO) and combinations of these two methods. A brief description has already been given in chapter 3.

5.3.1 Mathematical Model

The mathematical model behind those policies using VOVO together with voltage scaling is given by the following formulas.

The dynamic component of cpu power consumption - the only part affected by system load - is:

$$\textit{Dynamic cpu Power} = A\ C\ V^2\ f$$

where A is an activity factor and C the total capacitance at the gate outputs.

Expressing voltage V as a linear function in frequency f like $V = \alpha f$ and combining constants results in this reduced equation:

$$\textit{Dynamic cpu Power} = c_1 f^3$$

As the rest of the system has constant power consumption, the total consumption is

$$\textit{System Power} = P(f) = c_0 + c_1 f^3$$

For a cluster using both *Dynamic voltage scaling* and *Vary-on/Vary-off*, it is possible to shut down a node to save its power consumption, and increase the frequency for the remaining ones to maintain the performance level. This results in a slightly increased power consumption for each of the remaining nodes. In the same manner, turning on a node at the costs for a new node might be beneficial, as all nodes may decrease their operating frequency.

This breakeven point is calculated by first extending the power consumption equation to n systems running at a certain frequency f_1 with $n \times P(f_1) = n \times (c_0 + c_1 f_1^3)$. Calculating the power consumption of n and $n-1$ (resp. $n+1$) nodes leads to a threshold for $f_{varyoff}$ (resp. f_{varyon}), where shutting down (resp. turning on) a node reduces overall power consumption:

$$f_{varyoff}(n) = \sqrt[3]{\frac{c_0}{c_1} \frac{(n-1)^2}{2n^2 - n}}$$

and

$$f_{varyon}(n) = \sqrt[3]{\frac{c_0}{c_1} \frac{(n+1)^2}{2n^2 + n}}$$

So, the optimal average freqency range for a n-nodes cluster is $f_{varyoff}(n) \leq$ cpu frequency $\leq f_{varyon}(n)$.

Obviously, there are upper and lower bounds for f_{varyon} and $f_{varyoff}$ that must be taken into account:

$$\frac{n}{n-1} f_{varyoff}(n) > f_{max}$$

This means that a cluster with n nodes operating at $f_{varyoff}(n)$ cannot set the frequency of $n-1$ nodes any higher as each node is already running its maximum frequency possible.

$$\frac{n}{n+1} f_{varyon}(n) < f_{min}$$

Analogously, a cluster with n nodes operating at $f_{varyon}(n)$ cannot set the frequency of $n+1$ nodes any lower as each node is already running its minimum frequency possible.

5.3.2 Overview

The following assumptions are made:

- The following load balancing is done: incoming workload is distributed to the nodes according to a weighting, using their average response time of recent requests.
- Cpu power is the largest and most variable component of system power consumption, other components have constant power consumption.
- a cpu running at reduced frequency takes longer to complete a given task (this increase in execution time is proportional to the frequency reduction).

The following system factors are described:

- The frequency might be set to a value between 600MHz and 1.2GHZ, the corresponding voltage between 1.15V and 1.4V.
- There is a constant basis power consumption of 8.5W, a complete idle power consumption of 13.5W (including 5W for constant processor power) and a fully loaded power consumption of 36.2W $\rightsquigarrow c_0 = 13.5$W and $c_1 = 22.7$W.
- The frequency will be set to the next discrete value that is smaller than the current one if the utilization of cpu is less then 80% and increased to the next higher discrete value when the utilization is higher then 95%.
- No further details are given on how to decide when to shut down (turn on) a node.

Results:

- Savings by IVS and CVS are relatively the same (about 20% to 29%)

- VOVO is workload dependent better or worse then IVS/CVS: a more stable workload results in worse savings for VOVO
- IVS+VOVO is 8%-12% better then VOVO
- CVS+VOVO gets the best results (12%-18% more than VOVO, 3%-10% more than IVS+VOVO, saves up to 50% of the energy expended with no power management).

5.3.3 Evaluation

IVS and CVS produce the same results on the simulator because the cluster does not receive single requests that differ in execution time but only a request/second rate. As that rate is distributed equally to the nodes by the dispatcher, the load on those nodes does not vary between them. Thus only IVS is evaluated in the following.

The other restraint is that Elnozahy et al. do not give clear information about the VOVO algorithm and the decision when to turn on resp. shut down a node. So VOVO and IVS+VOVO will not be presented here. But as CVS+VOVO is explained in detail this strategy will be presented and analyzed in different scenarios.

IVS

A dispatcher using only IVS is very adaptable to any incoming workload because a reconfiguration with voltage scaling has a nearly instant effect on the capacity of the cluster (cmp. figure 5.16), i.e. the startup delay is nearly 0. Thus its only limit concerning the degree of flexibility is given by the length of the time interval that is used to check the cluster configuration and possibly change it.

But the disadvantage of IVS is the constant basis power consumption c_0 that is present all the time even when the cpu is operating at the lowest possible frequency. If that basis power consumption of a system is high, IVS cannot save much power (cmp. figure 5.17: a gaussian graph with runtime 4000 seconds and peak load 4000 requests/second was applied on a cluster with nodes with high c_0 and low c_0 using the IVS dispatcher). On the other hand, if most of the power is consumed by the cpu (i.e. $c_1 \gg c_0$ resp. a large value for $\frac{c_1}{c_0}$), the performance of IVS increases pretty well compared to other, much more complex strategies (cmp. the evaluation of CVS+VOVO in section 5.3.3 where the energy savings of various strategies on that gaussian curved workload are listed for nodes with high c_0 and for nodes with low c_0).

The exact numbers of the energy consumption of both experiments are as follows:

c_0 [W]	c_1 [W]	Energy Consumption [kJ]	Energy savings [%]
74.0	20.0	2584.29	1.91
13.5	22.7	643.09	11.03

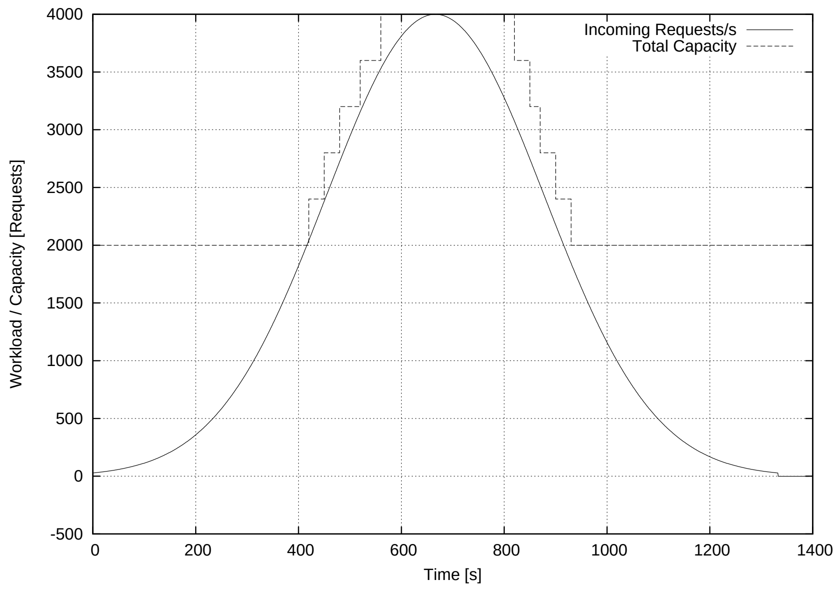

Figure 5.16: IVS is very adaptable even when the incoming workload rises very fast

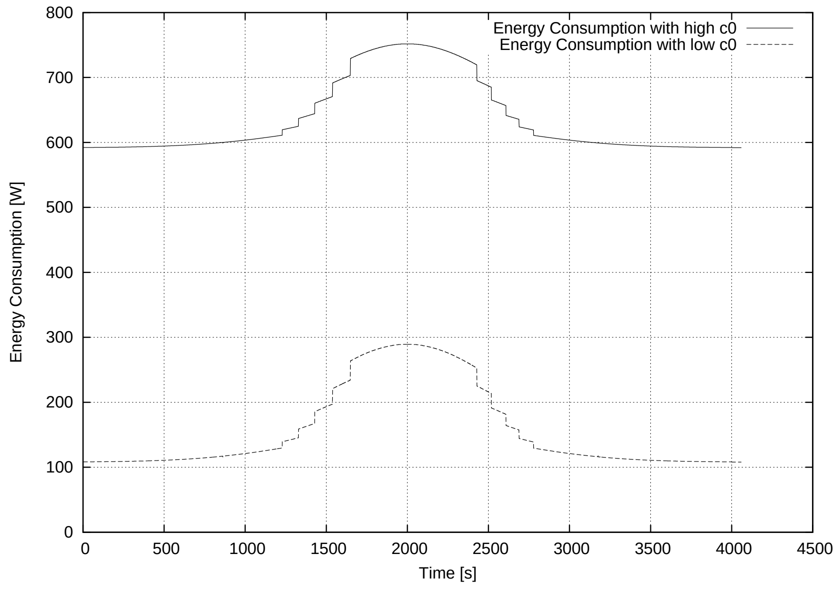

Figure 5.17: Energy consumption of IVS on a cluster with high c_0 and low c_0

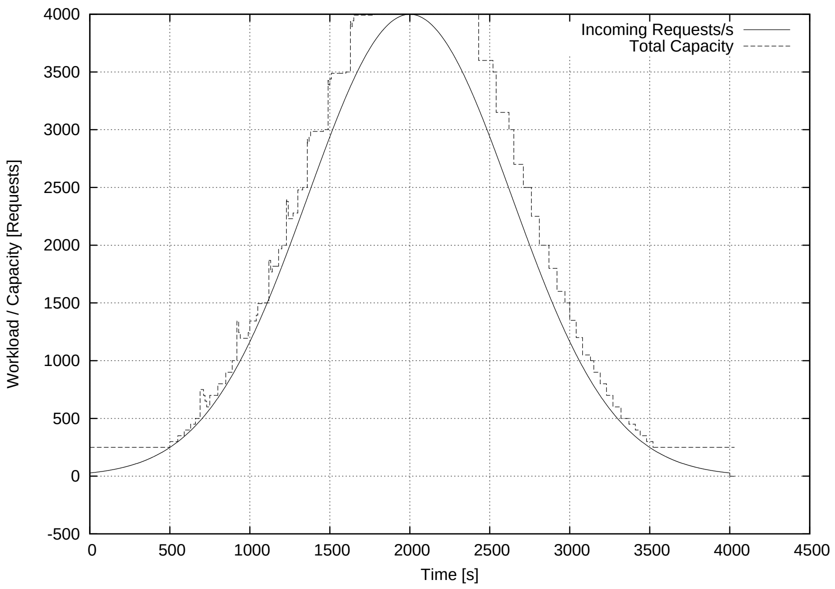

Figure 5.18: Behaviour of CVS+VOVO with peak load after 2666s (startup delay=30s)

CVS+VOVO

Testing CVS+VOVO in scenarios where the workloads are rising with different rates produces the graphs in figures 5.18, 5.19 and 5.20.

Obviously, the algorithm has no problems handling the increase of the workload in the case of the peak being reached after 2666 seconds or 1333 seconds. But in the scenario where the maximum of the workload is already applied after 666 seconds, the algorithm is not able to start enough nodes in the time where the workload rises rapidly, because it only adds one node at a time. In consequence of this restriction and the time that a node needs to boot, CVS+VOVO has a limit of increase in workload per time that it can handle. In every scenario where the incoming requests/second rise at a faster rate than that limit, the algorithm is not able to provide enough capacity for the cluster and has to suffer penalties in the quality of service.

The exact limit that the rate of increase in workload must not exceed between two points in time t_0 and t_1 (t_1 being t_0 plus twice the boot time of a node) is the capacity of a node:

$$\text{workload}(t_1) - \text{workload}(t_0) < \text{capacity of a node} \qquad t_1 = t_0 + \text{startup\_delay} \times 2$$

But in those cases where CVS+VOVO is fast enough (i.e. the increase in the workload is

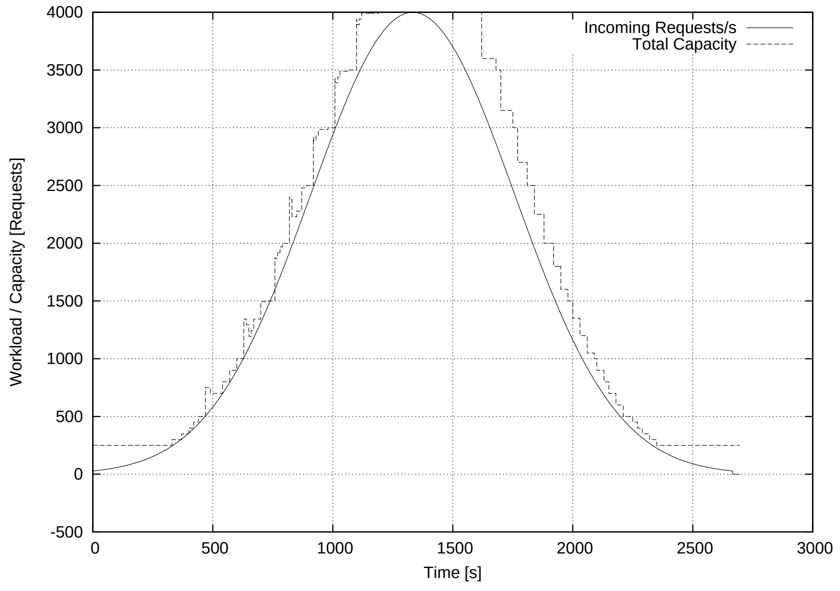

Figure 5.19: Behaviour of CVS+VOVO with peak load after 1333s (startup delay=30s)

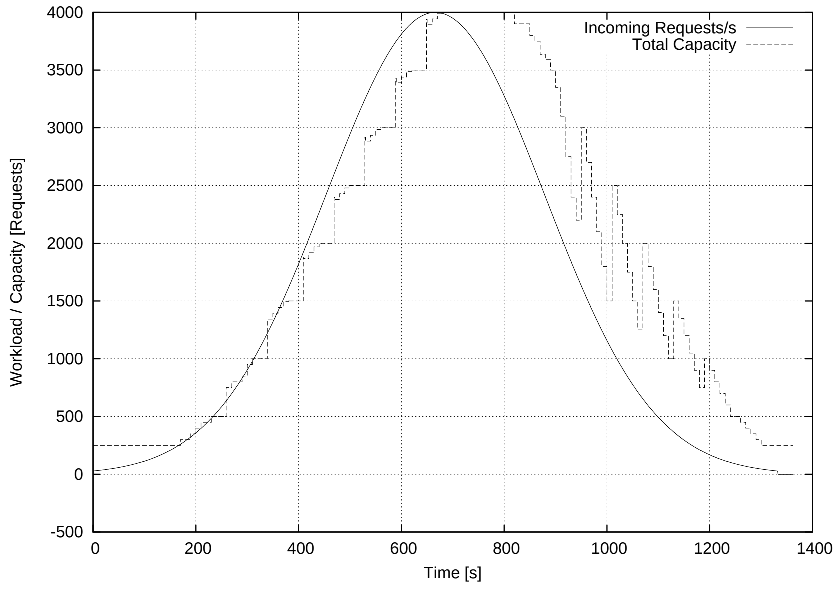

Figure 5.20: Behaviour of CVS+VOVO with peak load after 666s (startup delay=30s)

within the above limit) it can save more energy than IVS, VOVO_CT with $degrad = 0.70$ and even VOVO_CT with $degrad = 0.85$. For example when applying a gaussian curve as incoming workload over 4000 seconds (peak load reached after 2000 seconds) on a cluster (startup delay of a node, i.e. the time it needs to boot = 30s), the following energy consumptions were measured:

Algorithm [nodes with $c_0 = 70W$, $c_1 = 24W$]	Energy Consumption [kJ]	Total savings [%]
No energy saving strategy	2634.68	0.00
IVS	2584.29	1.91
VOVO_CT ($degrad = 0.70$)	1567.89	40.49
VOVO_CT ($degrad = 0.85$)	1481.93	43.75
CVS+VOVO	1424.15	45.95

Algorithm [nodes with $c_0 = 13W$, $c_1 = 22W$]	Energy Consumption [kJ]	Total savings [%]
No energy saving strategy	722.84	0.00
IVS	643.09	11.03
VOVO_CT ($degrad = 0.70$)	528.50	26.89
VOVO_CT ($degrad = 0.85$)	512.84	29.05
CVS+VOVO	475.69	34.19

Those tables show that not only IVS performs better when the $\frac{c_1}{c_0}$ ratio rises, but also that CVS+VOVO can save more energy in comparison to VOVO_CT when more energy in the system is consumed by the cpu.

Finally, figure 5.21 - 5.24 visualize how CVS+VOVO triggers different configurations when the energy consumption parameters of the nodes (i.e. c_0 and c_1) differ.

When the $\frac{c_1}{c_0}$ ratio rises (figure 5.23), CVS+VOVO does not wait that long until it turns on additional nodes as the costs for that action are not that high compared to the case where the $\frac{c_1}{c_0}$ ratio is low. In the same manner, it keeps nodes for a longer time in the active set because shutting down a node does not save that much energy compared to decreasing the frequency/voltage setting.

5.4 K. Rajamani et al. - On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters

The work of Rajamni et al. in [19] mainly consists of two parts. The first one gives a detailed composition of key system and workload factors which are heavily influencing energy efficiency. These factors have already been described in section 2.2.6 as system-workload context. Interactions

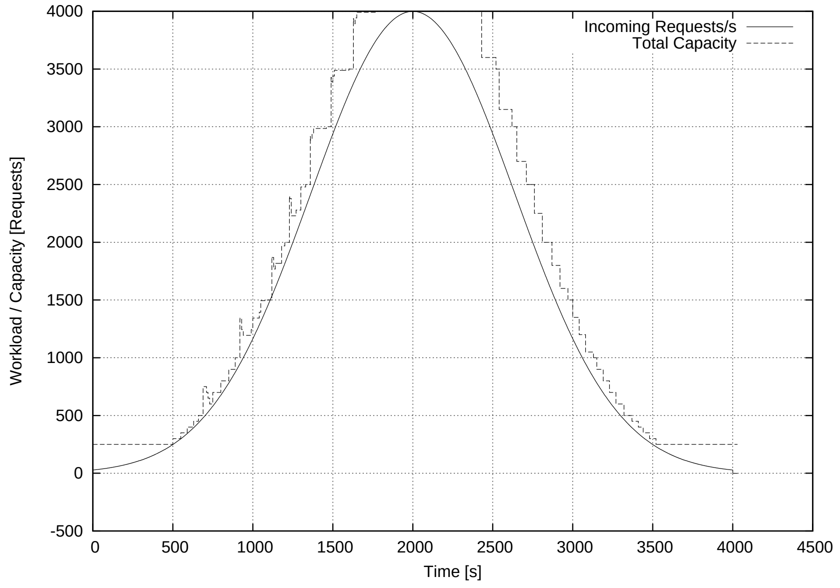

Figure 5.21: Capacity of a CVS+VOVO Cluster with $c_0 = 70W$ and $c_1 = 24W$

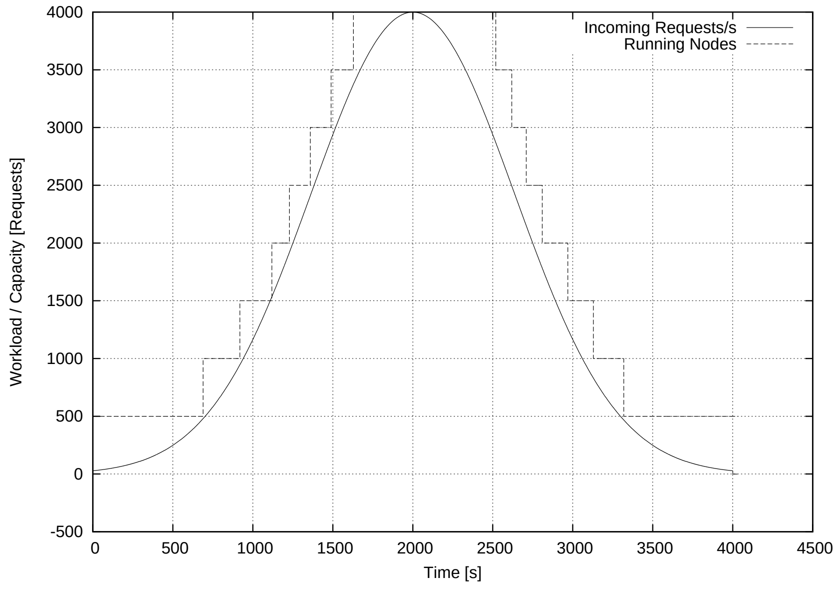

Figure 5.22: Running nodes in a CVS+VOVO Cluster with $c_0 = 70W$ and $c_1 = 24W$

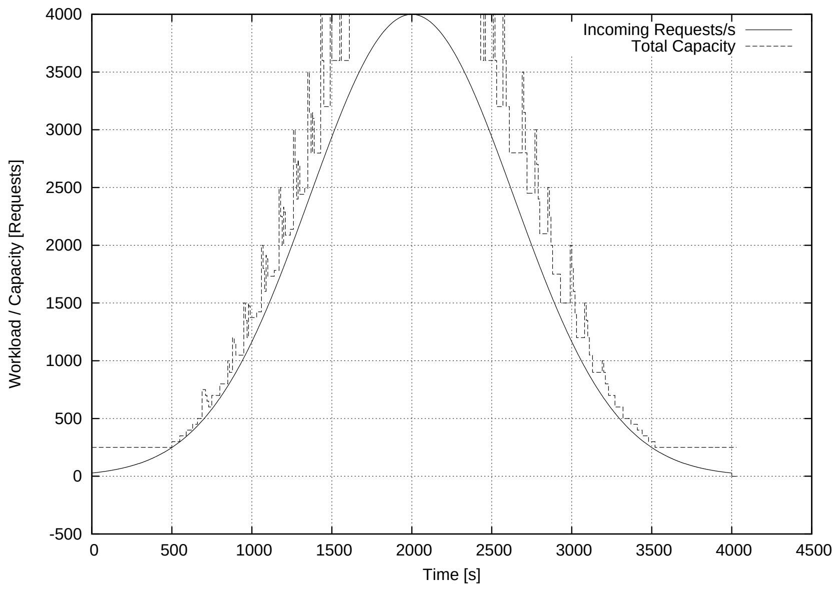

Figure 5.23: Capacity of a CVS+VOVO Cluster with $c_0 = 13W$ and $c_1 = 22W$

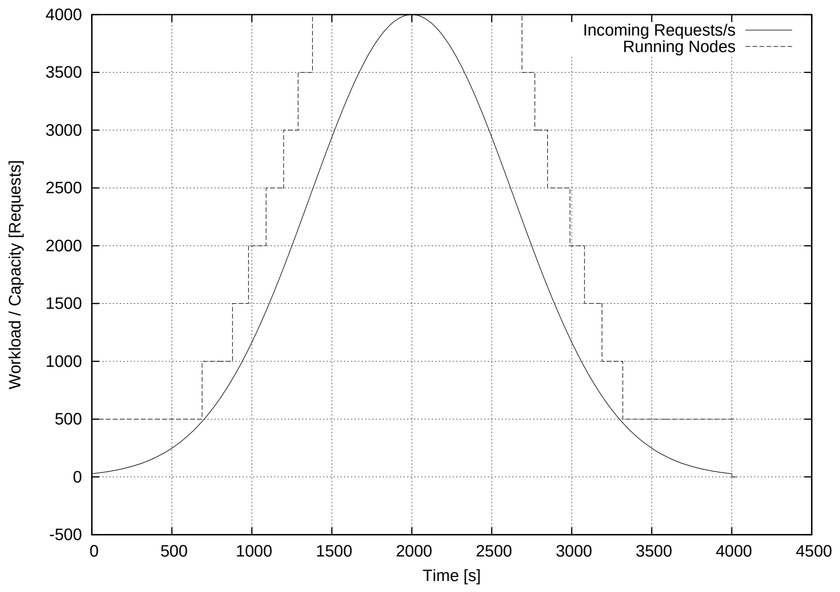

Figure 5.24: Running nodes in a CVS+VOVO Cluster with $c_0 = 13W$ and $c_1 = 22W$

between these factors and how they affect energy consumption are presented, assuming the load and its change is known for all instants of time:

$$L + SD \leq N \star C \tag{5.1}$$

Equation 5.1 describes how the startup delay D influences the number of necessary active servers N for any given load (with L the current load on the system, S its rate of change) when each server has a capacity of C (measured in terms of the number of active connections). The shutdown delay D_2 also affects energy consumption by consuming energy while shutting down servers proportionally to D_2.

In the second part of the work, three methods are presented to determine the minimum number of servers needed to handle a given workload without performance degradation. The rest of the servers in the cluster are turned off to save energy.

5.4.1 Simple Threshold

The first and easiest method is using a simple threshold parameter T to acquire the number of servers necessary at each point of time N_t. It is determined by:

$$N_t = L_t/T \qquad L_t \text{ being the load at any instant}$$

As each server should not handle more connections then its capacity C, a lower bound on the number of servers needed to meet the quality of service constraints is given by

$$N_t \geq \widehat{L}_{t,t+D}/C \qquad \widehat{L}_{t,t+D} \text{ being the maximum load from time t to t+D}$$

Assuming to know the parameters of the system and the workload, this formula results in an exact value for the threshold T:

$$\widehat{L}_{t,t+D}/C \leq L_t/T \tag{5.2}$$

$$T = min(L_t/(\widehat{L}_{t,t+D}/C))$$

Using a higher value would result in performance degradation, a lower value in lower energy savings.

5.4.2 Spare Servers

The simple threshold algorithm is not able to handle spikes in the workload well, as the only way is to use such a low threshold that enough servers are running when a spike occurs. This allows improvement of the algorithm which is done by Rajamani et al. by introducing *spare servers*. Spare servers are always active and are dedicated to handle spikes. The resulting formula analogous to 5.2 is

$$\widehat{L}_{t,t+D}/C \leq S + L_t/T_s \qquad \text{with } S \text{ being the number of spare servers and } T_s \text{ the threshold.}$$

The appropriate number of spare servers should be chosen depending on the workload. If the average load is high and has high spikes, using spare servers saves more energy than the simple threshold algorithm could.

5.4.3 History-based Schemes

Another improvement presented by Rajamani et al. is to incorporate the general characteristics of the workload, especially the expected spikes. Three levels of knowledge are differentiated, assuming knowledge of

1. the maximum spike S_{max},
2. the maximum spike that occurs for each system configuration available (corresponding to the number of servers required for each load point) and
3. the increase in load at each point of time

5.4.4 Overview

Rajamani et al. present energy savings up to 45.6% and describe these savings as about 2% short of the maximum savings possible. The different algorithms which are specified in the paper were not implemented and evaluated in the simulator, so no further informations are available at this point.

5.5 Results

Experimenting with VOVO_CT, IVS and CVS+VOVO on various workloads which were formed like a gaussian curve gave the following results:

- A higher value for *degrad* in the VOVO_CT algorithm increases the energy savings but also the likelihood that the demanded quality of service cannot be fulfilled.
- VOVO_CT is a very flexible algorithm as it is able to turn on more than one node at once. Thus it performs well even in scenarios where the workload is rising very fast provided that the nodes can be booted fast enough.

- IVS is pretty flexible, too, due to the fact that changing the frequency setting of the cpu has an almost instant effect on the capacity of the node concerned.
- On the other hand, IVS has the disadvantage that in scenarios where the nodes have a high base power consumption (i.e. high c_0) it cannot save much energy because all nodes are running all the time. The energy savings rise with a higher $\frac{c_1}{c_0}$ ratio.
- CVS+VOVO is a more fine-grained method than VOVO and thus is generally able to save more energy.
- But CVS+VOVO has a limit on how fast the workload is allowed to rise in order to ensure a postulated quality of service because the dispatcher may turn on only one node at a time.

In order to draw a comparison between the implemented algorithms under a real workload, the results of experiments with the trace from the World Cup 1998 website [3] are now described. The behaviour of the policies concerning the cluster configuration and the respective energy consumptions are shown in figures 5.25 - 5.33. The corresponding energy savings are listed in the tables below.

The following system parameters were defined:

Parameter	Value
Number of nodes in the cluster	8
Capacity of a node	250 Requests/second
Capacity at minimum voltage/frequency setting	125 requests/second
Startup delay (boot time of a node)	30s
c_0	70W
c_1	24W

Using these parameters the energy savings in the table below were measured with the world cup 98 trace, as shown in figure 5.2.

Algorithm	Energy Consumption [MJ]	Total savings
No energy saving strategy	160.85	0.00
IVS	155.56	3.29
VOVO_CT (*degrad* = 0.70, no smoothing)	59.94	62.92
VOVO_CT (*degrad* = 0.85, no smoothing)	.	N/A
VOVO_CT (*degrad* = 0.70, with smoothing)	59.35	63.10
VOVO_CT (*degrad* = 0.85, with smoothing)	54.61	66.05
CVS+VOVO	47.59	70.41

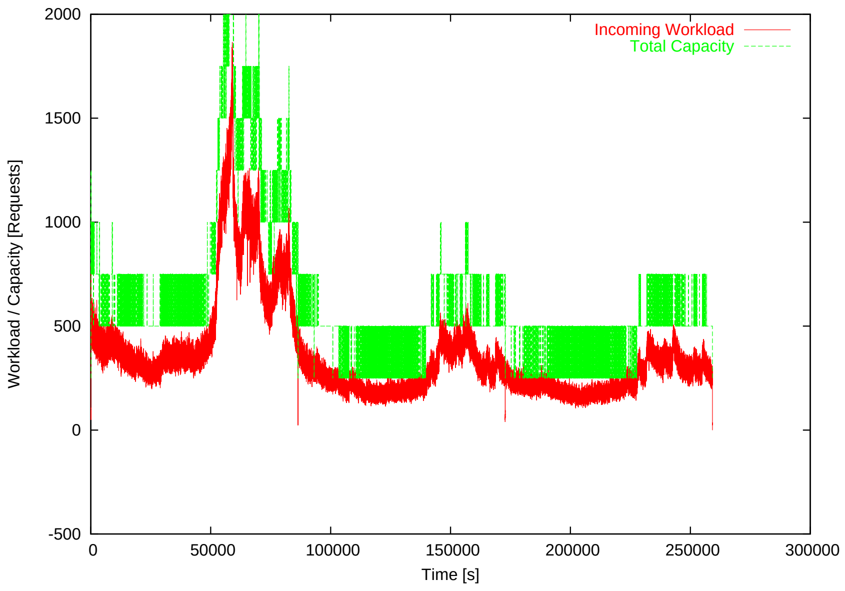

Figure 5.25: Behaviour of VOVO_CT with *degrad* $= 0.70$ (without smoothing)

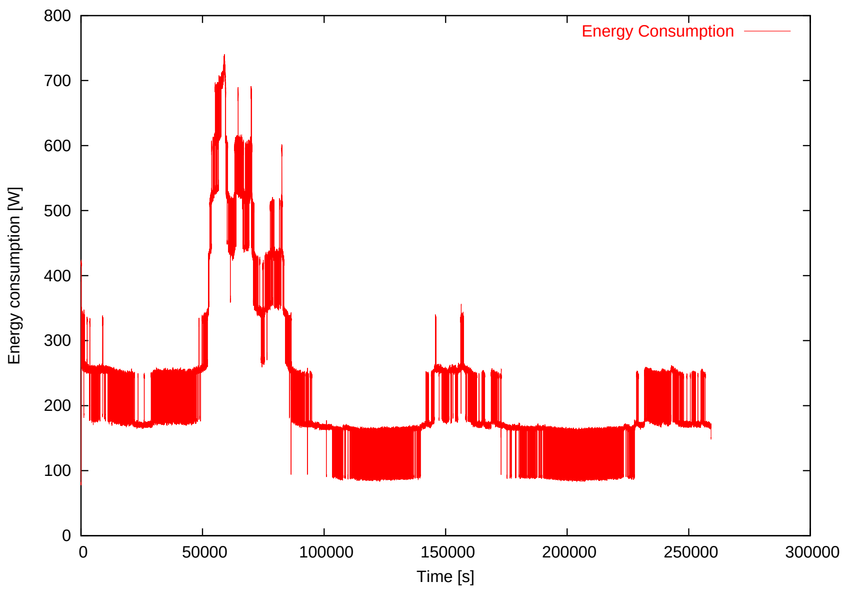

Figure 5.26: Energy usage of VOVO_CT corresponding to figure 5.25

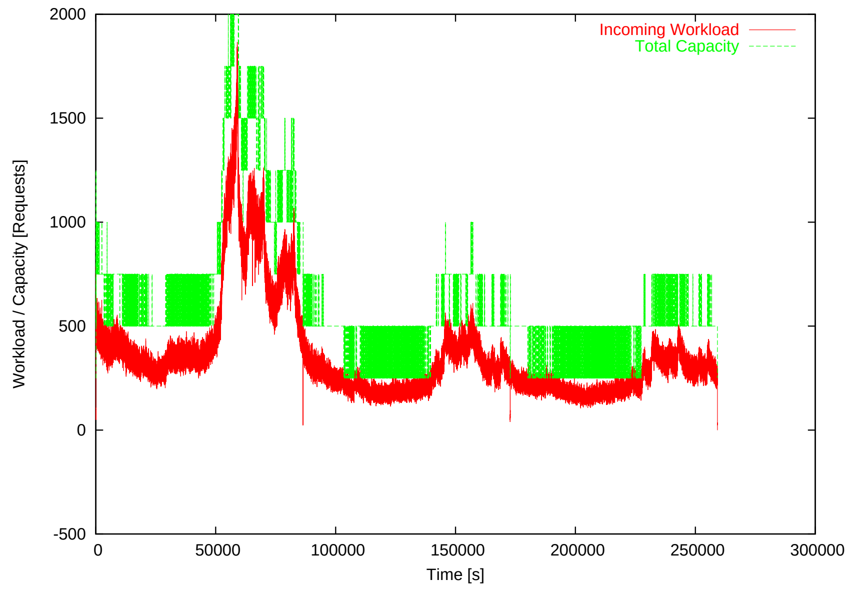

Figure 5.27: Behaviour of VOVO_CT with *degrad* = 0.70 (with smoothing)

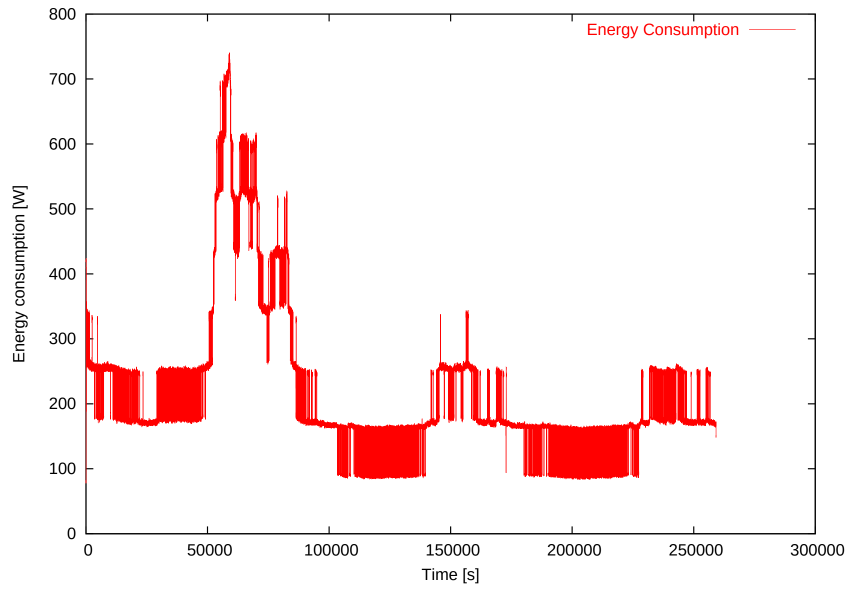

Figure 5.28: Energy usage of VOVO_CT corresponding to figure 5.27

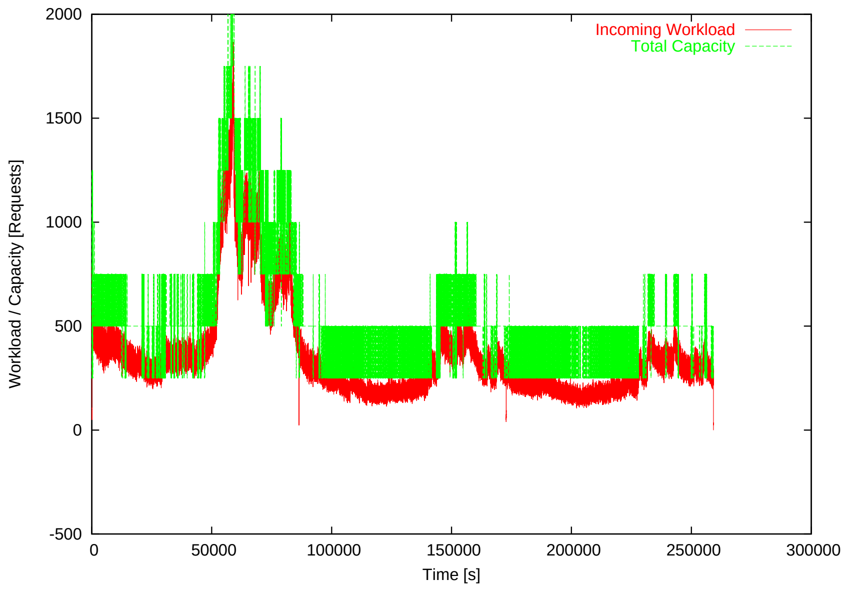

Figure 5.29: Behaviour of VOVO_CT with *degrad* $= 0.85$ (with smoothing)

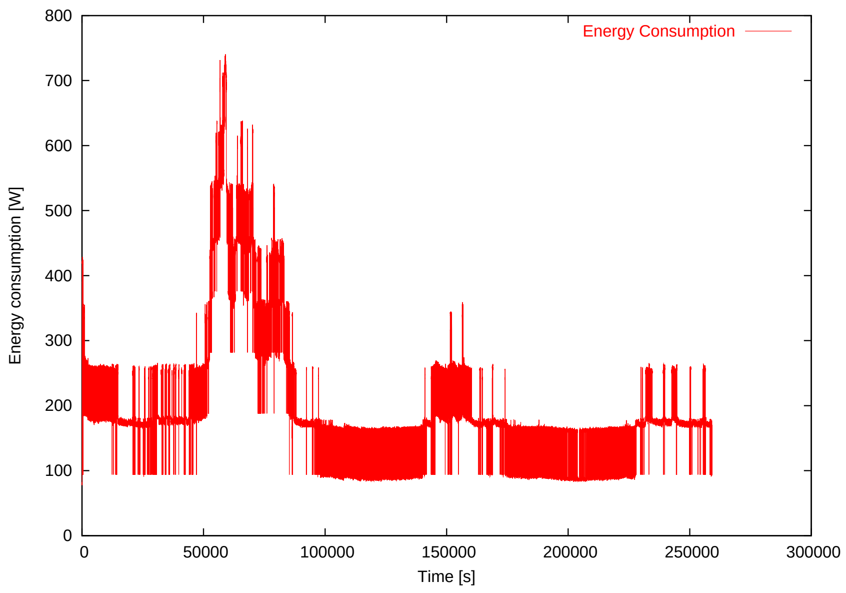

Figure 5.30: Energy usage of VOVO_CT corresponding to figure 5.29

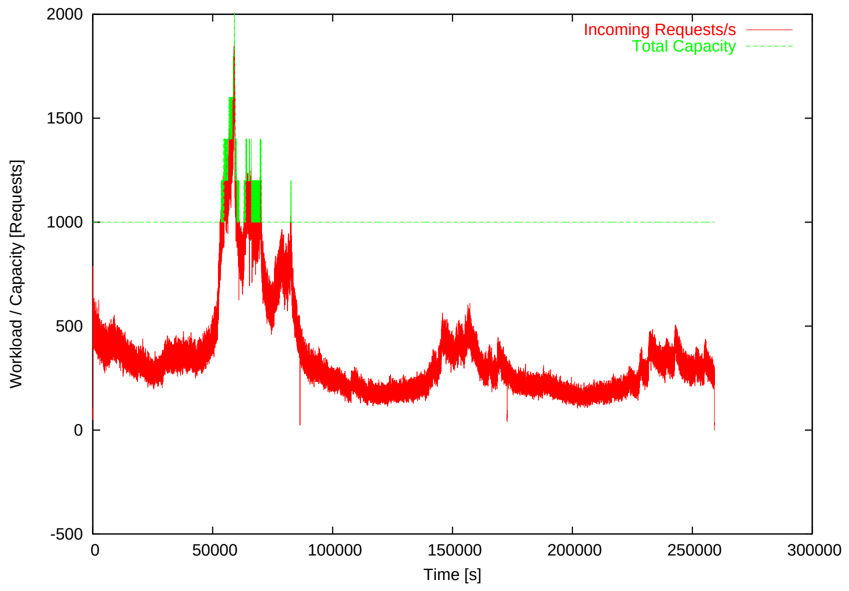

Figure 5.31: Behaviour of IVS

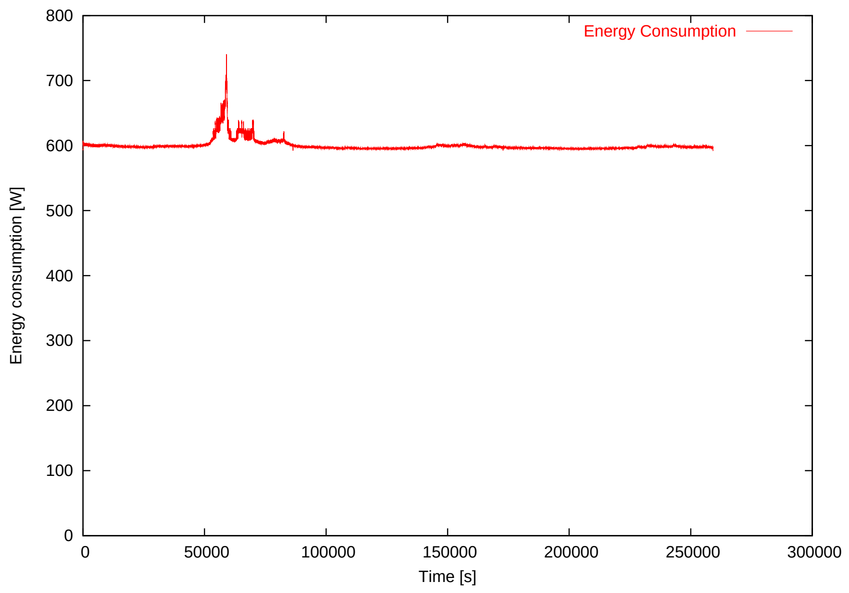

Figure 5.32: Energy usage of IVS corresponding to figure 5.31

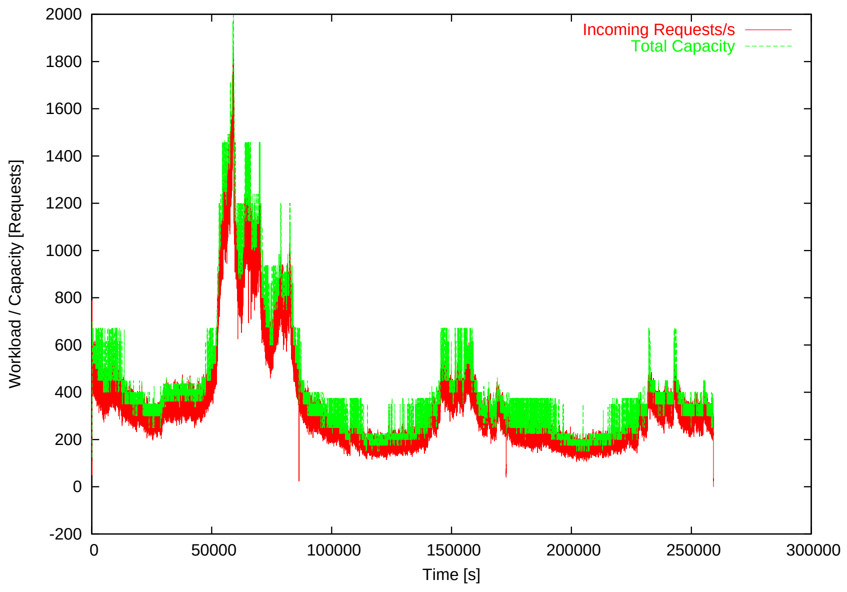

Figure 5.33: Behaviour of CVS+VOVO

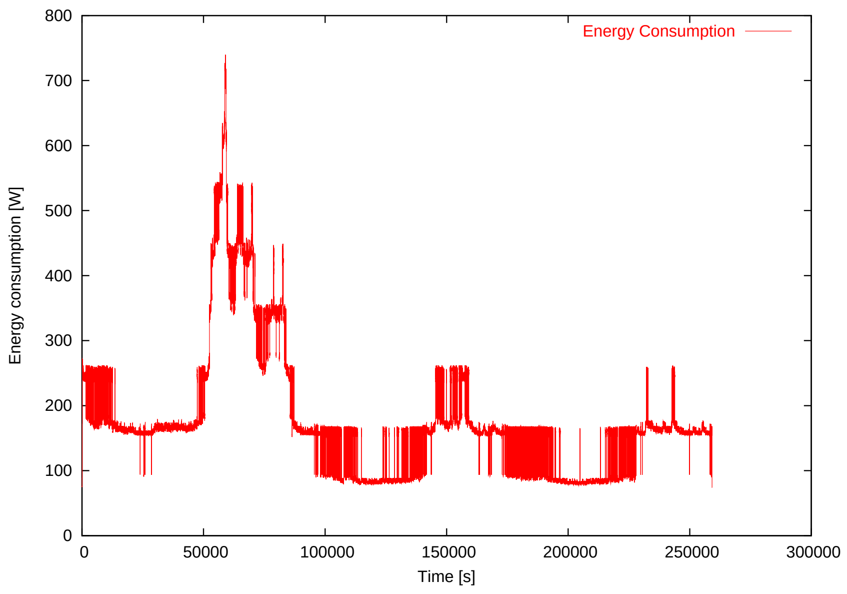

Figure 5.34: Energy usage of CVS+VOVO corresponding to figure 5.33

VOVO_CT with *degrad* $= 0.85$ and no smoothing performs very bad and is very often not able to handle the incoming workload without decreasing the quality of service drastically. Thus it is not listed in the table.

IVS is not able to save much energy, as the workload is almost all the time below the minimum capacity of the cluster (all of the 8 nodes running at lowest possible frequency setting with corresponding capacity of 125 requests/second) of $8 \times 125 = 1000$ requests/second.

CVS+VOVO performs slightly better than VOVO_CT as it applies not only the coarse grained vary-on/vary-off to save power but also the fine grained frequency/voltage scaling.

When using other values for the energy consumption of the nodes, the following values were measured:

Nodes with $c_0 = 13W$, $c_1 = 22W$ (as provided by Elnozahy et al. in [21]):

Algorithm	Energy Consumption [MJ]	Total savings
No energy saving strategy	36.39	0.00
IVS	30.54	16.08
VOVO_CT (*degrad* $= 0.70$, with smoothing)	17.93	50.73
VOVO_CT (*degrad* $= 0.85$, with smoothing)	17.06	53.12
CVS+VOVO	13.68	62.41

Due to the higher emphasis on the power consumption of the cpu c_1, both IVS and CVS+VOVO perform well in relation to other experiments where the base power consumption c_0 is dominating.

Nodes with $c_0 = 72W$, $c_1 = 32W$ (as measured on a Pentium 4, cmp. section 4.4.1):

Algorithm	Energy Consumption [MJ]	Total savings
No energy saving strategy	161.14	0.00
IVS	152.65	5.27
VOVO_CT (*degrad* $= 0.70$, with smoothing)	62.42	61.26
VOVO_CT (*degrad* $= 0.85$, with smoothing)	57.81	64.12
CVS+VOVO	49.67	69.18

Here the results are similar to those made before with $c_0 = 70W, c_1 = 24W$.

Nodes with $c_0 = 13.5W$, $c_1 = 1.5W$ (as specified by K. Rajamani et al. in [20]):

Algorithm	Energy Consumption [MJ]	Total savings
No energy saving strategy	28.55	0.00
IVS	28.16	1.37
VOVO_CT (*degrad* $= 0.70$, with smoothing)	10.03	64.87
VOVO_CT (*degrad* $= 0.85$, with smoothing)	9.16	67.92
CVS+VOVO	8.11	71.59

Again, CVS+VOVO is better than VOVO_CT but only a little bit due to the small c_1.

Nodes with $c_0 = 0.1W$, $c_1 = 20.0W$ (as theoretical model):

Algorithm	Energy Consumption [kJ]	Total savings
No energy saving strategy	7600.44	0.00
IVS	2282.30	69.97
VOVO_CT (*degrad* $= 0.70$, with smoothing)	7520.97	1.05
VOVO_CT (*degrad* $= 0.85$, with smoothing)	7518.17	1.08
CVS+VOVO	4358.23	42.66

In this theoretical model of a node where nearly all the energy is consumed by the cpu, the algorithms which use frequency/voltage scaling can score because only frequency/voltage scaling can save energy. Here it is the best to keep all nodes running (at nearly no costs) and have a very low average frequency/voltage setting for the cpu, as that saves most of the energy. This is not possible for CVS+VOVO, as VOVO shuts down some nodes (though less than in all other scenarios) and therefor increases the frequency/setting of the remaining nodes, which leads to an increase in power consumption.

But unfortunately, there does not exist a node where c_0 is nearly 0. As in c_0 the base power consumption of the cpu is also included (per definition), this device would be perfect scalable.

Annotations

When setting the startup delay from 30 seconds to 100 seconds or using the accelerated workload as input, the results remain the same: IVS does not save very much energy at all and CVS+VOVO is always better than VOVO_CT because of its greater flexibility. The advantage of VOVO_CT that it is able to react adequately on a rapid change in the incoming workload (by turning on or shutting down multiple nodes) cannot be exploited in this scenario in order to make a significant difference.

VOVO_CT would also perform better if the capacity of a node were smaller and a greater number of nodes were available as compensation therefor, as this would turn VOVO in a more fine grained method towards the level of frequency/voltage scaling.

The values given by Elnozahy et al. for the energy savings of IVS could not be exactly reconstructed in this scenario, but the reason therefor lies in the different system parameters for the nodes (i.e., their capacity and quantity) and in the characteristic features of the workload as explained above.

The savings for VOVO_CT and CVS+VOVO are a bit higher than those stated in the original publications, but this is due to the fact that the access trace of the world cup 1998 has its peak load of about 1800 requests/second only for a very short time and a relatively low workload of about 400 requests/second for the rest of the time. Therefore, strategies with VOVO can save a lot of energy by running the cluster with a few nodes only for most of the time compared to a static cluster with many nodes (which are necessary to cope with the high peak load).

Therefore, experimenting with other workloads with different characteristics, e.g. a higher average workload but a relatively lower peak load, will give values nearer to those given by Pinheiro, Elnozahy et al. in their publications.

Chapter 6

Future Work

This chapter gives a survey of possible extensions to the presented simulator and further experiments.

6.1 More Experiments Using the Simulator

With the simulator introduced in this work, it is possible to continue analyzing the implemented algorithms with other workloads which have different characteristics than those used in chapter 5, like e.g. suggested in section 5.5.

In addition, other parameters could be changed and the policies could be tested thereupon, e.g. experiments could be done using nodes with smaller capacities so that those strategies using VOVO should perform better. A point of interest would be to find out when VOVO is as good as IVS depending on the capacity of a node and the average workload.

One could also create another model for the behaviour of a node with respect to the energy consumption in relation to the load, which is more precise or reflects other circumstances/devices.

And of course, other strategies which are described in other publications such as [19] could be implemented, tested and evaluated.

6.2 Extending the Simulator

Another way to get new insights is to extend the functionality of the simulator. This could be done by e.g. implementing heterogeneous nodes as described in [13].

A second alternative could be to simulate differences of the incoming requests and the load they generate by adding a random factor in the `doWork` function in the class `Node`, which is responsible for working off incoming workload. Thereby it would be possible to test strategies that need such a variance between the nodes in a cluster, like e.g. CVS.

An aspect mentioned qualitatively but not measured quantitatively yet is the quality of service provided by the cluster. An extension in that direction could be to implement the possibility to measure that quality of service in terms such as maintained response times when a node has excess workload.

Not yet included either are other scenarios different from web servers like e.g. computing clusters which have other characteristics and requirements.

6.3 New Strategies

And finally, existing strategies could be extended by adding advantages known from other policies. For example, the factor of a time-of-day dependent, history based experience (like in [19]) could be added to CVS+VOVO in the way that at a certain time of day, when there is a rapid increase in the workload expected according to experiences from previous days, the policy does not only add one node if it is necessary to increase the cluster's capacity but two.

This would maybe add the advantage of VOVO_CT, namely to be able to handle workloads that are rising very fast, to CVS+VOVO and make CVS+VOVO more flexible. Of course, experiments would be necessary to verify that assumption.

Another possible topic which has not yet been investigated is trying to exploit the knowledge of exact energy consumption values, which are available by the work of Martin Waitz [23] and can be distributed over a network with the help of the work of Thorsten Ehlers [10].

Chapter 7

Conclusion

This work has elaborated important aspects and characteristics by means of which energy saving strategies can be evaluated and has presented a simulator which can be used to analyze such policies in various scenarios. The performance of an implemented policy can be tested with different parameters in order to examine the impact of the parameters. It has been made possible to use both artificially created workloads as well as workloads generated from real log files as input data.

The algorithms presented in three publications were analyzed by means of these general characteristics and some of them were exemplarily implemented in the simulator in order to evaluate them. These strategies using the concept of vary-on/vary-off, frequency/voltange scaling and combinations of them have been analyzed both in artificial environments in order to test certain aspects like e.g. the ability to react on rapid changes in the workload as well as with the help of real traces of access log files. Advantages and problems for each policy have been discovered and their reasons have been mentioned.

It turned out that a combination of frequency/voltage scaling with vary-on/vary-off achieves the best results for all workloads used in the experiments, especially when a significant fraction of the energy consumed by a node has to be spent on its cpu. It has been pointed out that for other workloads where there is no such big difference between the average load and the peak load, policies relying only on vary-on/vary-off should perform better and should be able to demonstrate their advantages, like e.g. the capability to turn on more than one node at a time.

With the simulator, it has been possible to reproduce and comprehend the results which were presented by the authors of the strategies. By experimenting with the same system parameters, comparisons of the policies have been made and differences in the results have been explained.

Finally, some possibilities for further work with the simulator have been suggested and make way for more research work and analyses.

Bibliography

[1] AMD. *AMD PowerNow!TM Technology*, 2000.

[2] The Apache HTTP Server Project. `http://httpd.apache.org/`.

[3] The Internet Traffic Archive. Worldcup98. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`.

[4] R. Bianchini. Research directions in power and energy conservation for clusters. Technical Report DCS-TR-466, Department of Computer Science, Rutgers University, November 2001.

[5] Pat Bohrer, Mootaz Elnozahy, Mike Kistler, Charles Lefurgy, Chandler McDowell, and Ramakrishnan Rajamony. The case for power management in web servers; p. bohrer, et al. In Robert Graybill and Rami Melhem, editors, *Power Aware Computing*. Kluwer Academic Publishers, 2002.

[6] Jeff Chase, Darrell Anderson, Prachi Thakur, and Amin Vahdat. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles SOSP'01,* October 2001.

[7] Jeff Chase and Ron Doyle. Balance of power: Energy management for server clusters. In *Proceedings of the Eighth Workshop on Hot Topic in Operating Systems HotOS'2001,* May 2001.

[8] Intel Corporation. Mobile Intel Pentium III Processors, Intel SpeedStep Technology. `http://www.intel.com/support/processors/mobile/pentiumiii/ss.htm`.

[9] Intel Corporation. Mobile Intel Pentium IV Processors, Intel Deep Sleep Technology. `http://www.intel.com/home/notebook/Pentium4-m/faq.htm#technical2`.

[10] Thorsten Ehlers. Transparent energy accounting in distributed systems. student thesis, March 2004. Department of Computer Science 4, University of Erlangen-Nürnberg.

[11] Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems USITS'03*, March 2003.

[12] Gnuplot interactive function plotting program. `http://gnuplot.info/`.

[13] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. Self-configuring heterogeneous server clusters. 2003.

[14] Hammerhead web server stress testing tool. `http://hammerhead.sourceforge.net/`.

[15] Microsoft Intel and Toshiba. Advanced configuration & power interface. `http://www.acpi.info/`.

[16] University of Erlangen-Nürnberg. Zugriffsstatistik fuer die domain www.uni-erlangen.de. `http://www.rrze.uni-erlangen.de/webadm/stats/www.uni-erlangen.de/`.

[17] Rainer Ollinge. Speedstep - technologie. `http://support.fujitsu-siemens.de/KnowHow/DE/Grundlagen/Prozessor/SpeedStep.htm`.

[18] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In Luca Benini, Mahmut Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2002.

[19] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. 2003.

[20] Karthick Rajamani and Charles Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'03)*, March 2003.

[21] Ram Rajamony, Mootaz Elnozahy, and Mike Kistler. Energy-efficient server clusters. In *Proceedings of the Second Workshop on Power Aware Computing Systems*, February 2002.

[22] Transmeta. *Crusoe TM Processor Model TM5800*, 2003.

[23] Martin Waitz. Accounting and control of power consumption in energy-aware operating systems. Master's thesis, Department of Computer Science 4, January 2003. SA-I4-2002-14.

[24] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the First Symposium on Operating System Design and Implementation OSDI'94*, November 1994.

Energiesparverfahren für Server Cluster

In der heutigen Zeit ist der Energieverbrauch von Rechnersystemen ein immer wichtigerer Aspekt, speziell wenn es sich um ganze Rechnerverbünde handelt. Da aktuelle Prozessoren im Vergleich zu ihren einfacheren Vorgängern immer schneller und komplexer werden, wird in modernen Rechenzentren in denen viele Server in Betrieb sind, ein nicht unbeträchtlicher Anteil der Betriebskosten für die Energieversorgung und Kühlung der Systeme aufgewendet. Die Forschung hat sich bisher hauptsächlich darauf konzentriert, wie man eintreffende Anfragen in solch einem Rechnerverbund für einen möglichst performanten Betrieb am effizientesten verteilt. Heutzutage gewinnt allerdings die Frage nach dem Energieverbrauch zunehmend an Bedeutung.

Es existieren schon einige Publikationen, die sich mit dieser Aufgabenstellung auseinandersetzen und verschiedene Methoden und Strategien als mögliche Lösung vorschlagen. Allerdings werden oft unterschiedliche Parameter und Hardwaresysteme als Grundlage für diese Methoden vorgestellt. Typische Beispiele hierfür sind Unterschiede in den verwendeten Eingabedaten, verschiedenste Systemplattformen oder unterschiedliche Annahmen über den Energieverbrauch im Ruhe- und Maximal-Last-Zustand. Außerdem erschweren fehlenden Erläuterungen über zugrunde liegende Algorithmen den direkten Vergleich der Qualität und Effizienz der verschiedenen Methoden. Man kann also ebenfalls nicht prüfen, ob diese Ergebnisse auf andere Plattformen oder Szenarien übertragbar sind, oder ob die Heuristiken und Parameter für einige wenige Fälle speziell optimiert wurden.

Diese Arbeit stellt wichtige Aspekte für Energiesparverfahren bei Server Clustern vor und zeigt ferner Charakteristiken auf, anhand derer Energiesparverfahren klassifiziert und analysiert werden können. Ein Überblick über bestehende Ansätze wird gegeben, wobei die jeweiligen zugrunde liegenden Annahmen genannt und die verwendete Methodik klassifiziert werden.

Außerdem wird ein parametrisierbarer Simulator präsentiert, der sowohl unterschiedliche Strategien als auch verschiedene Szenarien verarbeiten und auswerten kann. Damit können zum Beispiel Strategien verschiedener Publikationen unter identischen Bedingungen getestet und evaluiert werden, so dass ein Gesamtvergleich möglich wird. Zusätzlich können Heuristiken näher untersucht werden. Der Simulator verwendet echte Server log files die auch in einigen existierenden Publikationen für Untersuchungen eingesetzt wurden. Die Implementierung einiger repräsentativer Strategien wird als Beispiel vorgestellt und zur Analyse der jeweiligen Algorithmen, Parameter und

Heuristiken verwendet.

Als Beispiel wurden die Verfahren, die in drei Publikationen vorgestellt wurden analysiert und für den Simulator implementiert. Dieses Strategien deaktivieren ungenutzte Server (vary-on/vary-off), passen die Takfrequenz sowie die Spannung der Prozessoren an oder verwenden Kombinationen dieser Verfahren. Sie wurden sowohl mit künstlich erzeugten Zugriffsdaten getestet, um bestimmte Eigenschaften wie z.B. ihre Fähigkeit, auf schnelle Anstiege der eintreffenden Anfragen zu reagieren näher zu untersuchen, als auch mit der Hilfe von Zugriffsdaten echter log files. Die einzelnen Vor- und Nachteile jeder Strategie wurden aufgezeigt und ihre Gründe erläutert.

Es stellte sich heraus, dass die Kombination von Taktfrequenzanpassung in Verbindung mit vary-on/vary-off die besten Ergebnisse in allen Experimenten erzielte, vor allem wenn ein Grossteil des Energieverbrauchs eines Servers auf den Prozessor zurückzuführen ist. Es wurde aufgezeigt, dass vary-on/vary-off für andere Eingabedaten, die keinen so großen Unterschied zwischen durchschnittlicher und maximaler Last aufweisen, bessere Ergebnisse erzielen sollten. Dabei könnten z.B. auch die Vorteile dieser Strategien, wie die Fähigkeit mehrere Server auf einmal einschalten zu können, besser zum Ausdruck gelangen.

Für die implementierten Algorithmen konnten die Ergebnisse der Autoren nachvollzogen und mit Hilfe von Testreihen mit konstanten Systemparametern verglichen werden. Unterschiede zu den beschriebenen, tatsächlich gemessenen Werten wurden aufgezeigt und erklärt.

Abschließend wurden einige Möglichkeiten für weitere Experimente mit dem Simulator und Ansätze für Verbesserungen vorgeschlagen.