# A dynamic optimization model for power and performance management of virtualized clusters

Vinicius Petrucci[*], Orlando Loques
Universidade Federal Fluminense (UFF)
Niterói, Rio de Janeiro, Brasil
{vpetrucci,loques}@ic.uff.br

Daniel Mossé [†]
University of Pittsburgh
Pittsburgh-PA, USA
mosse@cs.pitt.edu

## ABSTRACT

An increasing number of large-scale server clusters are being deployed in data centers for supporting many different web-based application services in a seamless fashion. In this scenario, the rising energy costs for keeping up those web clusters is becoming an important concern for many business. In this paper we present an optimization solution for power and performance management in a platform running multiple independent web applications. Our approach assumes a virtualized server environment and includes an optimization model and strategy to dynamically manage the cluster power consumption, while meeting the application's workload demands.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; C.1.4 [**Parallel Architectures**]: Distributed architectures

## General Terms

Algorithms, Management, Performance, Experimentation

## Keywords

Power-aware computing, combinatorial optimization, server virtualization, web server clusters

## 1. INTRODUCTION

An increasing number of large server clusters are being deployed in data centers supporting many different web-based application services in a seamless, transparent fashion. These architectures are becoming common in utility/cloud computing platforms [8, 5], such as Amazon EC2 and Google AppEngine. These platforms may have great power demands, incurring in high energy costs and also indirectly contributing to increase CO2 generation and then to the environmental deterioration [17]. Thus, the energy consumed for keeping these server systems running became a very important concern, which in turn, requires major investigation of techniques to improve the energy efficiency of their computing infrastructure [3, 6].

In general, to allow hosting multiple independent applications, these platforms rely on virtualization techniques to enable the usage of different virtual machines (i.e., operating system plus software applications) on a single physical server. Virtualization provides a means for server consolidation and allows for on demand migration and dynamic allocation of these virtual machines, which run the applications, to physical servers [16, 27]. It is recognized that the dynamic consolidation of application workloads, through live migration of virtual machines, helps to increase server utilization, allowing to reduce the use of computer resources and the associated power demand [16, 23]. Furthermore, server consolidation can be combined with dynamic voltage and frequency scaling capabilities offered by current processors to get even better results.

In this paper, we propose an optimization solution for power and performance management in virtualized server clusters. We deal with the problem of selecting at runtime a power-efficient configuration and a corresponding mapping of the multiple applications running on top of virtual machines to physical servers. The optimization decision also includes selecting the best voltage/frequency combination for each physical server. To scale the solution over time, considering that the applications have individual time-varying workloads, our optimization strategy enables the virtualized server system to react to load variations and adapt its configuration accordingly.

The paper is organized as follows. The mathematical formulation and dynamic strategy of the underlying optimization problem are presented in Section 2. The system model and architecture used to apply our optimization proposal is described in Section 3. In Section 4, we experiment our optimization approach through simulations driven by actual workload traces. Section 5 summarizes related work and Section 6 concludes the paper.

## 2. OPTIMIZATION MODEL

The cluster optimization problem that we consider is to determine the most power efficient configuration (that is, which servers must be active and their respective CPU frequencies) that can handle a certain workload. The major goal of our approach is to reduce power consumption in the

[*]This work was partially supported by CNPq and Faperj.
[†]This material is based on work supported by NSF under grants ANI-0325353, CCF-0811295 and CCF-0803180.

cluster while meeting performance requirements. The underlying mathematical formulation for minimizing the power consumed in the virtualized cluster problem is given by a mixed integer programming (MIP) model.

In a virtualized environment, the applications are implemented as virtual machines (VM) which are assigned to physical servers. In our model, multiple different VMs can be mapped (consolidated) to a single server. We assume that applications in the cluster can be mapped to VMs and these to servers in two different ways, as follows. We assume at first (see Section 2.2) that an application runs on top of only one VM running on one physical server at a time. Alternatively, we present in Section 2.2.1 the optimization model that allows an application to be implemented using multiple VMs, which are mapped and distributed to different servers (see Figure 1). For example, a particular application workload, such as those including user sessions, can be "split" and executed in different servers. Note that this assumption may not hold for every application workload. We also introduce in our model a possible switching and migration penalty to avoid frequent and undesirable turning servers on/off and disruptive VM migration reconfigurations, described in Section 2.2.2.

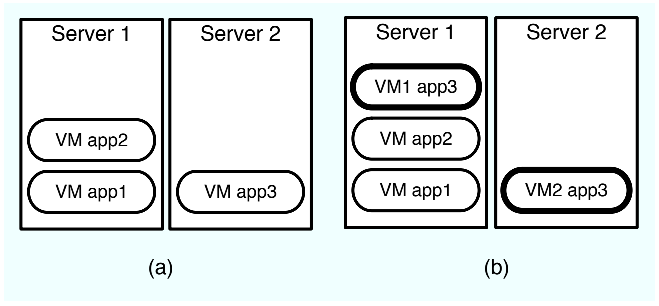

**Figure 1: Application workload mapping: (a) every application runs in only one VM instance on a given server, and (b) one application may run in more than one VM instance, whereas these VMs are balanced among multiple servers**

As some of the input variables for the optimization may change over time, such as the application workload vector, a new instance of the optimization problem is constructed and solved periodically. Currently, to achieve the necessary scalability within a bound processing time, for all inputs, we rely on the use of features, such as solution time limit and optimal gap tolerance, provided by state-of-art solver packages, such as ILOG CPLEX [10]. For example, the time limit feature sets the maximum time, in seconds, for a call to the optimizer and it returns the optimal or the best known solution at the end of the time limit. We are also working on heuristic techniques combined with our exact approach (based on our MIP formulation) for this particular problem.

## 2.1 Notation

Before describing the optimization model, we introduce the following notation. Let $N$ be the set of physical servers in the cluster and $F_i$ the set of frequencies for each server $i \in N$. Let $M$ be the set of applications intended to run on the virtualized cluster. The parameter $cap_{ij}$ represents the maximum performance or capacity (e.g., requests/sec) of the server $i$ running at CPU frequency $j \in F_i$. The parameters $pb_{ij}$ and $pi_{ij}$ denote the busy and active-idle power cost, respectively, to run server $i$ at frequency $j$, where the $\alpha_{ij}$ variable denotes the utilization of server $i$ running at frequency $j$. For each application $k \in M$, we define the parameter $d_k$ to represent the workload demand of that application. In practice, because load variations are small in short time intervals, the application demand vector can be generated by monitoring the load (in requests per second) for each application in a front-end machine in a server cluster (see Section 3). The following decision variables are defined: $x_{ijk}$ is a binary variable that denotes whether server $i$ uses frequency $j$ to run application $k$ ($x_{ijk} = 1$), or not ($x_{ijk} = 0$); $y_{ij}$ is a binary variable that denotes whether server $i$ is active at frequency $j$ ($y_{ij} = 1$), or not ($y_{ij} = 0$).

## 2.2 Problem formulation

$$\textit{Minimize} \quad \sum_{i \in N} \sum_{j \in F_i} \alpha_{ij} \cdot pb_{ij} + (y_{ij} - \alpha_{ij}) \cdot pi_{ij} \tag{1}$$

$$\textit{Subject to} \sum_{k \in M} d_k \cdot x_{ijk} \leq cap_{ij} \cdot y_{ij} \quad \forall i \in N,\ \forall j \in F_i \tag{2}$$

$$\sum_{i \in N} \sum_{j \in F_i} x_{ijk} = 1 \qquad \forall k \in M \tag{3}$$

$$\sum_{j \in F_i} y_{ij} \leq 1 \qquad \forall i \in N \tag{4}$$

$$\alpha_{ij} \leq y_{ij} \qquad \forall i \in N,\ \forall j \in F_i \tag{5}$$

$$x_{ijk} \in \{0,1\},\ y_{ij} \in \{0,1\},\ \alpha_{ij} \in [0,1]$$

The objective function given by Equation (1) is to find a cluster configuration that minimizes the overall server cluster cost in terms of power consumption. The power consumption for a given server $i$ is the combination of the busy power $pb_{ij}$ and the idle (but active) power $pi_{ij}$ for the selected CPU frequency $j$.

The constraints (2) prevent a possible solution in which the demand of all applications $k \in M$ running on a server $i$ at frequency $j$ exceed the capacity of that server. The constraints (3) guarantee that a server $i$ at frequency $j$ is assigned to a given application $k$. The constraints (4) are defined so that only one frequency $j$ on a given server $i$ can be chosen. The constraints (5) are used to bind the decision variable $y_{ij}$ with the $\alpha_{ij}$ variable in the objective function. The solution is thus given by the decision variable $x_{ijk}$, where $i$ is a server reference, $j$ is the server's status (i.e., its operating frequency or inactive status, when $j = 0$), and $k$ represents the respective allocated application.

Our virtualized cluster configuration problem is a variant of the one dimensional variable sized bin packing problem [7], which is defined as follows. Suppose we are given a set $B$ of different bin types (servers), where each bin type $i \in B$ has capacity $W_i$ and a fixed cost $C_i$. The problem involves packing a set $J$ of items (applications), where each item $j \in J$ has a weight (demand) $w_j$, into a minimum-cost set of bins. In addition, the total weight of the allocated items into a bin cannot exceed its capacity. As a generalization of the classic bin-packing problem, this problem is known to be NP-hard [7].

An important difference between our problem and the original variable-sized bin packing is that in our case we have, for each bin type, the possibility to choose among different options (CPU speeds) in a given bin (server). However, as shown previously in constraints (3) the MIP model, only one CPU speed on a given server can be chosen.

### 2.2.1 *Application workload balancing*

The previously described model cannot be applied to application workloads that demand more capacity than the supported by a single server. To solve this issue, we propose an extension to the optimization model. In practice, the proposed extension means that each application may be associated with multiple VMs running on different servers. From the original optimization model, we replace constraints (2) and (3) with the constraints (6) and (7), respectively, which are defined as follows:

$$\sum_{i \in N} \sum_{j \in F_i} cap_{ij} \cdot x_{ijk} \geq d_k \quad \forall k \in M \tag{6}$$

$$\sum_{k \in M} x_{ijk} \leq y_{ij} \qquad \forall i \in N, \forall j \in F_i \tag{7}$$

We also modify the decision variable $x_{ijk} \in [0, 1]$ to represent the utilization factor of an application $k$ in a given server $i$ that runs at frequency $j$. Note that this modification adds a relaxation to the optimization problem, since the $x$ decision variable is now continuous. One positive side effect is that it makes the MIP problem a bit easier to be solved, while still maintaining NP-hardness.

To handle the incoming workloads, as stated by the constraints (6), for each application $k$, the total execution capacity available across all servers (to be selected) which runs that application must be greater than or equal to the given application demand $d_k$. The constraints (7) guarantee that variable $y_{ij}$ equals one if a server $i$ at frequency $j$ is assigned to handle any portion of a given workload for application $k$. That is, the server will be on if there is at least one workload application running on it.

### 2.2.2 *Modeling switching and migration costs*

In a real server cluster environment, dynamic configurations in the cluster may be subjected to switching costs, for example, in terms of the power consumed while a machine is being turned on/off. To handle this issue, we incorporate a penalty value in our optimization model to represent the overhead of turning machines on/off. Specifically, we modify the objective function of the original optimization model, Equation (1), as follows:

$$\begin{aligned} Minimize \quad & \sum_{i \in N} \sum_{j \in F_i} \alpha_{ij} \cdot pb_{ij} + (y_{ij} - \alpha_{ij}) \cdot pi_{ij} \\ & \qquad + swt\_cost(U_{ij}, y_{ij}) \\ & \qquad + mig\_cost(A_{ijk}, x_{ijk}) \end{aligned} \tag{8}$$

The modified objective function given by Equation (8) has new terms to account for switching and migration costs. To calculate a transition cost, we have included in the model a new parameter input $U_{ij}$ to denote the current cluster usage in terms of which machines are turned on and off; that is, $U_{ij} = 1$ if machine $i$ is running at speed $j$. Similarly, the new parameter input $A_{ijk}$ denotes which application is currently associated with which server-frequency.

More precisely, we may define the switching cost function as follows: $swt\_cost(U_{ij}, y_{ij}) = y_{ij} \cdot (1 - U_{ij}) \cdot ON\_P$. The constant $ON\_P$ represents a power overhead penalty for turning a machine on (if it was off), which means an additional power consumed to boot a server machine. Currenly, we do not consider the penalty of changing frequencies. Actually, for a given server $i \in N$, if $U_{ij} = 1$ for at least one $j \in F_i$, we set $U_{ij} = 1$ for all $j \in F_i$ to avoid taking into account frequency switching costs. In addition, we assume that both server switching on/off and migration penalties can be estimated in a real server cluster. For example, the cost of VM migration could be measured a priori and stored in a table using the approach proposed in [27].

## 2.3 Optimization control policy

Dynamic optimization behavior is attained by periodically monitoring the proper inputs of the optimization model and solving a new optimal configuration given the updated values of the inputs. In other words, as some of the input variables may change over time, such as the application workload vector, a new instance of the optimization problem is constructed and solved at run-time. We assume that the workload does not change radically often and it is mostly stable during the specified control period.

Particularly, we are able to devise a control loop of the following form: (a) monitor and store the most recent values of the optimization input variables, (b) construct and solve a new optimization problem instance, yielding a new optimal configuration and (c) apply the changes in the system, transitioning the system to a new state given by the new optimized configuration. The details are given in the following algorithm.

```
do periodically:
  // 1. Input variables
  d = getDemandVector()
  curUsage = getCurrentUsage()
  curAlloc = getCurrentAlloc()

  // 2. Run optimization
  newUsage, newAlloc = bestConfig(d)

  // 3. Generate usage and alloc sets for changes
  chgUsage = sort(diff(newUsage, curUsage))
  chgAlloc = sort(diff(newAlloc, curAlloc))

  // 4. Power management operations
  for (i, j) in chgUsage:
      if j == 0:
          turnOff(i)
      else:
          if curUsage[i] == 0:
              turnOn(i)
          setFreq(i, j)

  // 5. Virtualization management operations
  for (k, i) in chgAlloc:
      if i == 0:
          stopVm(k, curAlloc[k])
      else:
          if curAlloc[k] == 0:
              startVm(k, i)
          else:
              migrateVm(k, curAlloc[k], i)
```

The control loop outlined above relies on the mathematical formulation described in Section 2.2 to solve the cluster configuration problem. The key idea of the optimization control policy is to periodically select and enforce the lowest power consumption configuration that maintains the cluster within the desired performance level, given the time-varying incoming workload of multiple applications.

The input variables for the control loop algorithm, described in the step 1, are: the monitored and updated application demand (load) vector, the current server configuration and application allocation. The **bestConfig** operator, at algorithm step 2, returns a cluster usage and allocation solution, where `newUsage` represents an usage configuration of servers and their respective status (i.e., its operating frequency or inactive), and `newAlloc` represents which application has to be associated with each server.

In fact, the configuration to be imposed is a difference between two sets: the new configuration and the current configuration solution. For example, suppose the current cluster usage is `curUsage={(1,0),(2,2),(3,0)}` and the new one is `newUsage={(1,0),(2,0),(3,4)}`. Thus, we need to perform a change in the system given by `chgUsage = newUsage` − `curUsage = {(2,0),(3,4)}`. That is, we need to turn off server 2, and turn on server 3 at the frequency 4. To handle this, we apply a **diff** operator in the usage and allocation solutions provided by the optimization operator (see the algorithm step 3).

The order in which the operations are executed may lead to a problematic behavior. Specifically, in the example above, if the new usage configuration shutdowns the current active server before the new server is ready to respond the requests, as the server booting time is not instantaneous, the cluster will be in an unavailable state. To solve this issue, we simply sort the new cluster usage representation so that the operation to shutdown servers is always performed at last, and the operations to increase frequency and turn on servers, respectively, are performed at first. This scheme can be similarly adopted to the new allocation representation, in which the operations to start and migrate virtual machines are performed at first. To achieve this, we make use of a **sort** operator in the configuration algorithm, as shown in algorithm step 3.

In the step 4, we employ dynamic configurations for power optimization which consists of (a) turning servers off in periods of low activity and turning them back on when the demand increases, and (b) exploiting the dynamic voltage and frequency scaling capability of current processor architectures to further reduce power consumption. Finally, to manage the application services (which are associated to virtual machines), we rely on configuration operators to start, stop, and migrate the virtual machines in the server cluster, such as those described in the algorithm step 5.

There may be some practical concerns in applying the control loop algorithm in a real cluster environment. For example, a sequential way of executing the configuration operations may be adopted. This would help to avoid inconsistency among the execution of multiple operations, for example, guaranteeing that the servers will be turned on before the migration operations are requested. Additionally, the time delay associated with effecting the configurations must be taken into account, such as turning a server on/off.

**Execution example.** As an example of optimization execution, we may assume that the demand vector (in request/sec) for three different applications is $d = [45, 120, 17]$. After solving the optimization problem, we have an abstract configuration solution given by a vector of tuples $(i, j, k)$, where $i$ is a server, $j$ is the respective CPU speed, and $k$ is the allocated application, defined by $conf = [(1,3,1),(1,3,2),(1,3,3)]$. This means that, application 1, 2 and 3 are hosted by server 1 at frequency 3, which is its maximum frequency, while the other servers are turned off.

At another execution snapshot, say $d = [45, 170, 17]$, the new configuration solution is defined by $conf = [(2,1,1),(1,3,2),(1,3,3)]$. This means that if the demand for application 2 has been increased from 120 to 170, we need to turn on a new server and migrate application 1 to this new server 2 that will run at frequency 1 (i.e., the minimum frequency). In case the demand for application 2 decreases, we may turn off server 2 to save power and migrate back application 1 to server 1. This particular example does not consider the overhead of switching servers on/off.

## 3. SYSTEM MODEL / ARCHITECTURE

In our server cluster, we need to maintain two correlated objectives. First, we aim to guarantee some quality-of-service requirements for the cluster (e.g., by controlling average cluster load or request response time). Second, to reduce costs, the set of currently active servers and their respective processor's speeds should be configured to minimize the power consumption.

The architecture (shown in Figure 2) consists of a cluster of replicated web servers. The cluster presents a single view to the clients through a special component termed *dispatcher server* (also called *front-end* or *load balancer*), which distributes incoming requests among the actual *servers* that process the requests (also known as *back-ends* or *workers*). Our optimization scheme can be implemented and deployed in the dispatcher server or in a dedicated server.

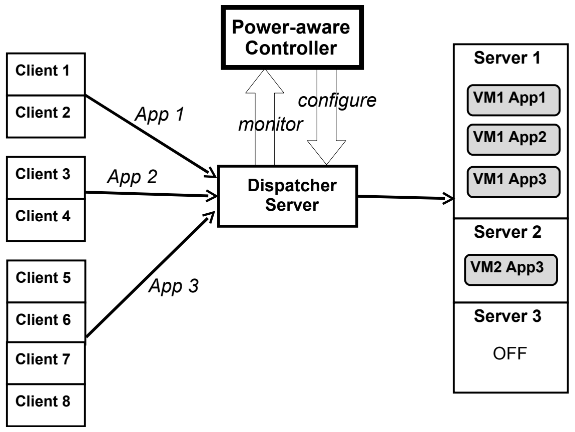

**Figure 2: System architecture**

In this work, we consider $m = 3$ virtualized application services to be hosted by a cluster with $n = 5$ physical servers. The system load is measured by the number of incoming requests per second in the cluster. Because we are interested in the macro behavior of the dynamic optimization, our cluster model is simplified in that we assume that there is no state information to be maintained for multiple requests.

Specifically, we measured the capacity of the servers, for each frequency, in terms of the maximum number of requests per second (req/s) that they can handle at 100% CPU utilization. To generate the benchmark workload, we used the *httperf* tool, where each HTTP request is a PHP script with a fixed average execution time. The power consumption of a server, for each available discrete frequency, varies linearly with its CPU utilization. Using the LabVIEW software environment, coupled with a USB based data acquisition device, we measured different active-idle and busy power values for each frequency to build our power model. This power estimation works well because servers have near-linear power response between active-idle and busy (100% utilization) [22]. That is, we are able to measure only active-idle and busy states to achieve very good estimates of power used at any utilization level for a given workload. Figure 3 shows an example of power and performance model for a machine measured in our cluster.

Server 5: ohm
CPU: AMD Athlon(tm) 64 X2 5000+

| Freq. (MHz) | $P_{busy}$ (W) | $P_{idle}$ (W) | Perf. (Req/s) |
|---|---|---|---|
| 1000 | 82.5 | 65.8 | 92.9 |
| 1800 | 99.2 | 68.5 | 165.9 |
| 2000 | 107.3 | 70.6 | 184.4 |
| 2200 | 116.6 | 72.3 | 201.0 |
| 2400 | 127.2 | 74.3 | 218.1 |
| 2600 | 140.1 | 76.9 | 235.3 |

**Figure 3: Power and performance model**

Depending on the system requirements, a relationship between cluster load and response time might be established, for example, assuming queue models or control theory [2, 28]. Leveraging our optimization strategy, we can tailor our controller, for example, to monitor the request response time and to adapt the cluster capacity accordingly. These approaches could be used to provide soft real-time guarantees in server clusters, in which the requests processing time have a specified deadline. We intend to incorporate the response time control in our approach implementation as future work.

## 3.1 Dynamic optimization support

Following this control loop paradigm, we have proposed and developed a framework [18], which works on top of an abstract configuration model of the system and provides general configuration mechanisms: (1) to specify and monitor run-time properties of an executing system, (2) evaluate the model for system's requirements violation and (3) perform adaptations on the system configuration to maintain the system behavior within acceptable bounds. The dynamic configurations are performed under guidance of scripts written in a high-level configuration language, designed separately from the target system.

In our approach, the monitoring and dynamic configuration mechanisms (used by our optimization policy in Section 2.3) can be implemented in terms of an application programming interface (API) given by the system support level (see Figure 4). For example, the Apache web server [24] supports an API to enable developers to extend a server with their own extension modules. The Xen hypervisor [1] also provides capabilities and mechanisms to monitor and manage virtual machines in a server cluster by means of an API.

The key idea of an API is that it specifies an abstract and well-defined interface to control the behavior of the system,

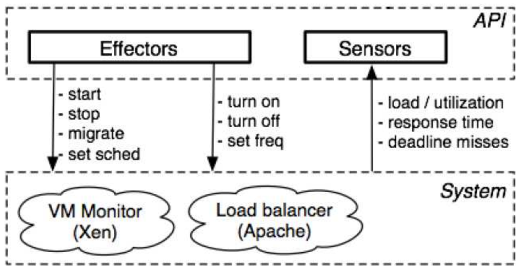

**Figure 4: API support for dynamic optimization**

which builds on lower-level mechanisms at the system level. A desired feature for an API is that it can be called from several programming languages and is available as a remote procedure call, such as the XML-RPC protocol.

Our optimization approach encapsulates most of the required functionality for dynamic configuration in terms of an API and provide generic configuration operators. This, in turn, enables the dynamic optimization logic to be described in a more appropriate way by using a number of high-level constructs (cf. Section 2.3). Examples of our API and operators include a call to a configuration operator, termed **bestConfig**, which encapsulates an optimization algorithm for solving the cluster configuration problem.

## 4. PROPOSAL EVALUATION

To evaluate our optimization proposal, we have carried out a set of simulation experiments using description of the cluster environment described in Section 3. The optimization problem formulation was implemented using the solver CPLEX 11 [10], which employs very efficient algorithms based on the branch-and-cut exact method [19] to search for optimal configuration solutions. The simulations were performed in an Intel Core 2 Quad 2.83 GHz with 8GB of RAM memory running Ubuntu Linux (kernel version 2.6.27). In the simulation results, we adopted a control loop of one second interval, where the optimization worst-case execution time was about 70 ms (Section 4.1.3), considering our study case of 3 applications and 5 physical servers (Section 3).

## 4.1 Simulation results

We generated three distinct workload traces using the 1998 World Cup Web logs to characterize the multiple applications in the cluster. The applications within the cluster can have a wide range of request demands, as shown in Figure 5. The workloads are spaced by 1 second, approximately 30 minutes in duration. We adapted the original workload data to fit our cluster capacity, which is measured in requests per second.

The maximum capacity of our cluster setup is 745 req/s, when all servers are turned on at full CPU speed. The combined workload curves of the three applications reach a peak of 734 req/s at time around 470s (see Figure 5), which represents 98% of the maximum capacity of our cluster.

### *4.1.1 Dynamic optimization execution*

In our simulation, we assume that App1, App2, and App3 services can be distributed and balanced among all servers in the cluster. Thus, we adopt the modified optimization formulation described in Section 2.2.1. For instance, the App1 service at the workload time around 700s demands a

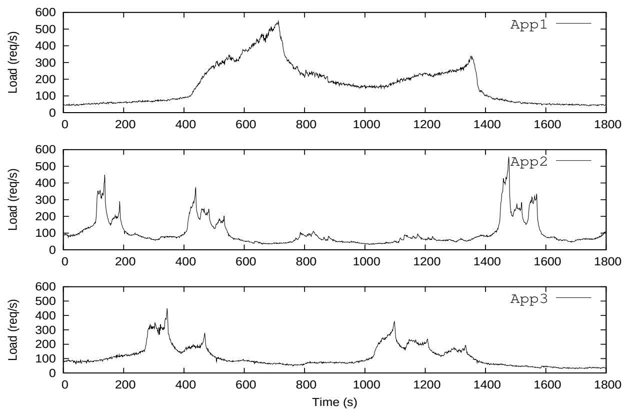

**Figure 5: Workload traces for three different applications using HTTP logs from WC98**

performance of more than 500 req/s. The maximum performance achieved by using a single machine in the cluster is 235 req/s, which corresponds to the 5th server running at maximum CPU frequency.

The simulation execution results are given in Figure 6. These results were plotted in 30 second intervals to help in visualizing the switching activities. The upper plot shows the configuration changes (frequency switching) for all servers in the cluster. If the operating frequency for a server is zero, it is turned off. In the bottom plot, we can observe the optimized allocation of the applications in the cluster along the simulation. For example, from time 0s to 57s, the allocation set $\{1, 2, 3\}$ is associated with the 5th server, meaning that all the three applications are consolidated on that server while the other servers can be turned off. At time around 470s, all servers are turned on to handle the highest peak for the combined workloads.

### 4.1.2 *Energy savings*

We mainly evaluated the effectiveness of our approach in terms of the percentage of energy consumption reduction in the cluster as compared to the Linux `on-demand` and `performance` CPU governors. The `performance` governor keeps all servers turned on at full speed to handle peak load and dynamic optimization is not conducted. The `ondemand` governor allows for managing the CPU frequency depending on system utilization, but does not include server on/off mechanisms.

We implemented by means of simulation the `ondemand` policy based on the algorithm described in [25]. The basic idea is as follows: If current utilization is more than an up threshold (80%), the policy increases the frequency to the maximum. Whenever a low utilization (less than 20%) is observed, the policy jumps directly to the lowest frequency that can keep the system utlization at 80%.

The allocation sets for the `performance` and `ondemand` governors are statically configured in this way: Server 1 hosts $\{App2\}$; Server 2 hosts $\{App3\}$; Server 3 hosts $\{App1\}$, Server 4 hosts $\{App1\}$; Server 5 hosts $\{App1, App2, App3\}$. The respective allocation shares are defined as follows: App1 uses 100% of Server 3, 100% of Server 4 and and 13.1% of Server 5; App2 uses 100% of Server 1 and 14.6% of Server 5; App3 uses 100% of Server 2 and 70.3% of Server 5. These values were obtained from the optimized configuration solution at the highest peak for the combined workloads. We used a simple round-robin method for application workload balancing.

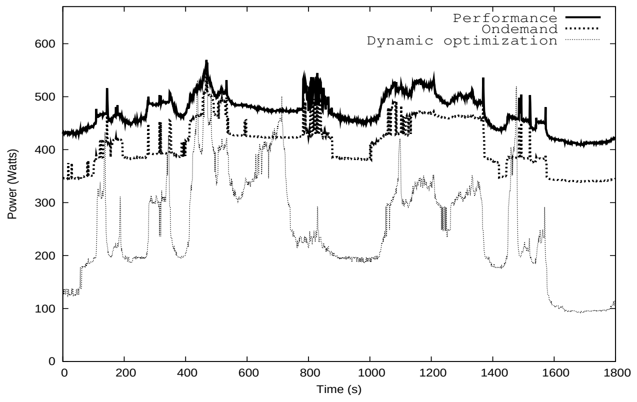

**Figure 7: Cluster power consumption**

The energy consumption was calculated by means of an approximation given by the sum of utilization of the active servers multiplied by the related busy and active-idle power consumption, accumulated along the simulation duration time. That is, assuming the notation introduced in Section 2.2, the cluster energy consumption can be expressed as $E = \sum_t \sum_{i,j} \alpha_{ij}^t \cdot pb_{ij} + (1 - \alpha_{ij}^t) \cdot pi_{ij}$ such as $i$ is an active server and $j$ is its operating frequency with the respective utilization at time $t$, imposed by the associated workloads, and $t \in \{1, \cdots, T\}$ is expressed in seconds, where $T$ is the duration of the simulation. Thus, $E$ is measured in Joule (or watt x second).

By using our approach, the energy consumption in the cluster is substantially reduced. In the `performance` simulation execution, a total energy consumed was 847,778.82J = 235.49Wh, whereas in the `ondemand` execution, the energy consumed was 735,630.05J = 204.34Wh. In the execution using our approach the energy usage was 452,050.15J = 125.57Wh. This means an energy consumption reduction of about 47% compared to `performance` policy and 38% compared to `ondemand`. The main argument for this greater energy savings is the fact that active-idle power consumption of current server machines (as shown in Figure 3) is substantially high. This in turn makes server on/off mechanisms (used by our optimization) very power-efficient.

### 4.1.3 *Scalability considerations*

To evaluate the scalability of our approach, we generated different pairs of server-application setup. For each pair, we ran the CPLEX to build and solve 180 instances. This means that each instance uses as its application demand vector the workload data at each time interval of 10 seconds using the traces shown in Figure 5.

The CPLEX solver was executed for every instance with a user-defined solution time limit of 180 seconds, which is related to the maximum allowed control period used in our dynamic optimization policy for managing the cluster environment. Table 1 shows the results of simulations with different number of servers and applications. From 5 to 30 servers, the optimal configuration solutions were found in all 180 runs within the solution time limit. From 50 to 100 servers, there is at least one instance where CPLEX could

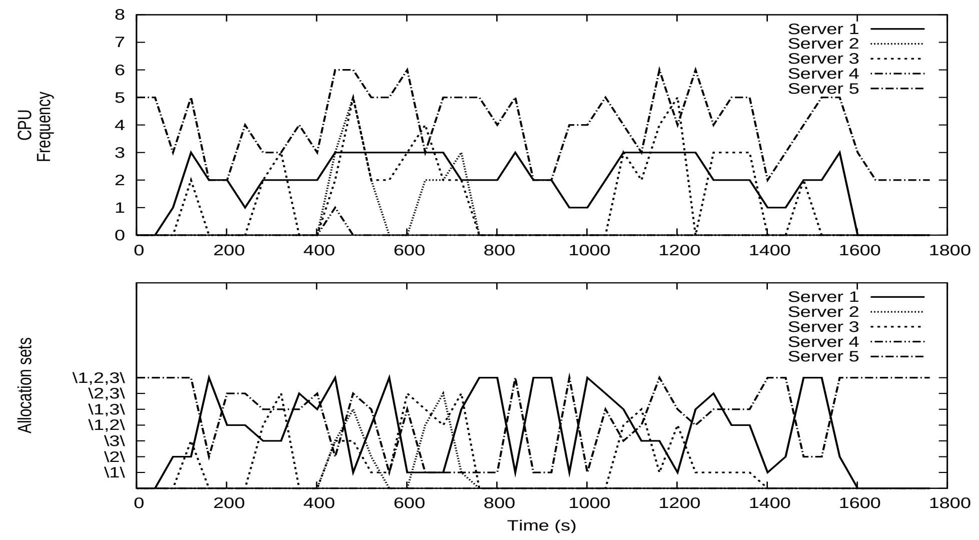

**Figure 6: Dynamic optimization execution**

not find the optimal solution for all instances within 180 seconds.

| Server-App | Avg. (s) | Stdev. (s) | Max. (s) |
|---|---|---|---|
| (5,3) | 0.022 | 0.018 | 0.070 |
| (10,6) | 0.054 | 0.035 | 0.250 |
| (15,9) | 0.062 | 0.038 | 0.240 |
| (30,18) | 0.392 | 0.913 | 8.610 |
| (50,30) | 13.630 | 29.959 | 180.010 |
| (80,48) | 58.941 | 53.570 | 180.020 |
| (100,60) | 80.135 | 52.394 | 180.030 |

**Table 1: Scalability simulation**

In order to speed up the process of obtaining a high quality solution, we adopted a simple heuristic by setting a gap tolerance of 5% with respect to the optimal solution. This is a user-defined value and intends to allow the solver to provide acceptable solutions in a short amount of time. Table 2 summarizes the simulation results when the minimum gap tolerance criteria was adopted. From 5 to 350 servers, the configuration solutions were found in all 180 runs within the gap tolerance (5%) and the solution time limit (180 seconds), with a maximum processing time about 75 seconds. For 500 servers, there were three instances where CPLEX could not find the solution within the time limit.

This strategy considers the solution gap between the best integer feasible solution found so far and the lower bound (LB) provided by the solver, which is usually calculated by solving a pure linear version the original problem. In minimization problems, the LB can be seen as a reference value which ensures that the optimal solution is greater or equal than this quantity. Considering the small gap value used, the CPLEX was capable of finding highly acceptable solutions, i.e., close to the optimal lower bound.

Even though we generate a number of scenarios involving different pairs of server-application setup, it is not possible to assume that the CPLEX will have a similar behavior in all instances. The main difficulty is that the branch-and-cut method has a worst-case exponential time complexity

| Server-App | Avg. (s) | Stdev. (s) | Max. (s) |
|---|---|---|---|
| (5,3) | 0.006 | 0.008 | 0.040 |
| (10,6) | 0.023 | 0.022 | 0.100 |
| (15,9) | 0.031 | 0.030 | 0.130 |
| (30,18) | 0.062 | 0.067 | 0.540 |
| (50,30) | 0.139 | 0.281 | 2.390 |
| (80,48) | 0.267 | 0.235 | 3.000 |
| (100,60) | 0.481 | 0.409 | 3.080 |
| (200,120) | 2.893 | 1.993 | 11.550 |
| (350,210) | 16.488 | 12.979 | 75.440 |
| (500,300) | 48.409 | 41.472 | 181.030 |

**Table 2: Scalability simulation using the optimality gap criteria**

and depending on the combination of application workloads, this approach may lead to poor solutions in an acceptable runtime execution (solution time limit). Nevertheless, based on the simulations presented here, we have observed that the CPLEX performs well on the average case.

Given a typical optimization control period of few minutes, such as used in [16], the proposed optimization approach is suitable and scales well for clusters with up to 350 machines. This seems to be a reasonable size for a server cluster set, because, for instance, servers can be divided in smaller clusters or racks in a hierarchical fashion to address scalability issues.

### *4.1.4 Switching cost effects*

We describe two execution scenarios to analyze the switching cost effects in our optimization approach. To account for the switching cost, we define the penalty $ON\_P = 50W$ in terms of the power consumed for turning machines on, as proposed in Section 2.2.2. That is, we assume that a system switching on consume 50W more than when active-idle. We have specified a sufficiently high penalty to avoid switching overhead, but more accurate values for these penalties have to be investigated in further study. Table 3 shows a comparison between the two scenarios. Scenario $A$ does not account for switching costs, whereas scenario $B$ does.

| Time | Demand | Config. A | Config. B |
|---|---|---|---|
| 1 | $[32, 100, 15]$ | $[(5,2,1), (5,2,2), (5,2,3)]$ | $[(5,2,1), (5,2,2), (5,2,3)]$ |
| 2 | $[32, 120, 15]$ | $[(1,2,1), (1,2,2), (1,2,3)]$ | $[(5,3,1), (5,3,2), (5,3,3)]$ |
| 3 | $[32, 140, 15]$ | $[(5,4,1), (5,4,2), (5,4,3)]$ | $[(5,4,1), (5,4,2), (5,4,3)]$ |

**Table 3: Switching cost scenarios**

Recall that a configuration solution is defined as a vector of tuples $(i, j, k)$, where $i$ is a server at frequency $j$ and $k$ is an application allocated to that server. At time 1, the configuration solutions are identical for the two scenarios, which means that all applications are hosted by server 5 at frequency 2. However, at time 2, the new configuration for the scenario $A$ requires that server 1 be turned on (at frequency 2) and server 5 be turned off, which involve overhead costs. On the other hand, including switching costs as in scenario $B$ requires that server 5 (which is already turned on) increase its frequency to 3, which adds essentially no disruption to the system. At time 3, both the scenarios require that the server 5 increase its frequency to 4, transitioning to the same configuration. Note that similar disruptive on/off actions were employed by scenario $A$, whereas scenario $B$ has only relied on manipulation of frequencies. To simplify the calculation, we assumed that a server can be turned on in one time step.

We also evaluated the effectiveness of the switching cost modeling in terms of the number of switching activities reduction as compared to the optimization model that does not account for switching costs (used as baseline). Our simulation execution (using the workload description from Section 4.1) indicated that the adoption of a switching cost model required 90 activities of turnning servers on/off and 156 activities when employing the baseline model. This means a switching reduction of 42%. Moreover, in a real scenario, application services in the cluster may retain important persistent states, such as web sessions, which leads to additional cost in switching from one server to another. Also, some overhead related to hardware reliability can be accounted for. We believe these issues provide useful directions of future investigation.

## 5. RELATED WORK

There are differences between our optimization solution and the major works that precede it. Several optimization approaches, based on the bin packing problem, for configuring virtualized servers are described in the literature, such as [4, 13]. However, their models are not designed for power-aware optimization. In [28], the authors present a two-layer control architecture aimed at providing power-efficient real-time guarantees for virtualized computing environments. The work relies on a control theory based framework, but does not addresses live migration and on/off mechanisms in a multiple server context. A heuristic-based solution for the power-aware consolidation problem of virtualized clusters is presented in [23], but it does not guarantee to find solutions that are at least near to the optimal. In [16], a dynamic resource provisioning framework is developed based on lookahead control. A power-aware migration framework for virtualized HPC (High-performance computing) applications, which accounts for migration costs during virtual machine reconfigurations, is presented in [26, 27]. Similarly to our approach, it relies on virtualization techniques used for dynamic consolidation, although the application domains are different.

Contrasting with [23, 16, 27], our approach takes advantage of dynamic voltage/frequency scaling (DVFS) mechanisms to optimize the server's operating frequencies in order to reduce the overall energy consumption. An approach based on DVFS is presented in [9] for power optimization and end-to-end delay control in multi-tier web servers. Recent approaches, such as presented in [20, 2, 14, 15, 12, 21] also rely on DVFS techniques and include server on/off mechanisms for power optimization. However, these approaches are not designed (and not applicable) for virtualized server clusters. That is, they do not consider multiple application workloads in a shared cluster infrastructure.

## 6. CONCLUSION

In this paper, we presented an approach for power optimization in virtualized server clusters, including an optimization MIP model and dynamic configuration strategy. In the optimization model, we addressed application workload balancing and the often ignored switching penalty aimed to avoid frequent and undesirable turning servers on/off. Our simulations show that our strategy can achieve power savings of 47% compared to an uncontrolled system (used as baseline).

By using a simple but effective optimality gap criteria in the optimization solver, our approach scales well for clusters with up to 350 servers. Currently, we are working to integrate our MIP model with heuristic approaches to provide suboptimal but very high-quality solutions in a short period of time. However, in very large data centers, with thousands of machines, smaller sets (hundreds) of machines of can be allocated (dynamically) to support dedicated clusters for specific applications, which could be managed autonomously (as pointed out by [5]). Thus, we believe that our optimization approach can be successfully applied in this context.

As for future work, we aim to evaluate our approach in a real virtualized computing environment using Apache web servers [24] and virtual machine technology, such as the Xen hypervisor [1]. We also intend to incorporate in our experiment two features: (a) a measure to account for the overhead incurred during live migration of virtual machines, and (b) prediction techniques to improve the quality of dynamic optimization decisions, such as those described in [16, 11].

## 7. REFERENCES

[1] Paul Barham et al. Xen and the art of virtualization. In *SOSP'03*, pages 164–177. ACM, 2003.

[2] Luciano Bertini et al. Statistical QoS guarantee and energy-efficiency in web server clusters. In *ECRTS'07*, pages 83–92, 2007.

[3] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems. *Computer*, 37(11):68–74, 2004.

[4] Martin Bichler, Thomas Setzer, and Benjamin Speitkamp. Capacity Planning for Virtualized Servers. *Workshop on Information Technologies and Systems (WITS), Milwaukee, Wisconsin, USA*, 2006.

[5] Kenneth Church, Albert Greenberg, and James Hamilton. On delivering embarrassingly distributed

cloud services. In *HotNets*, 2008.

[6] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[7] Mohamed Haouari and Mehdi Serairi. Heuristics for the variable sized bin-packing problem. *Comput. Oper. Res.*, 36(10):2877–2884, 2009.

[8] Brian Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, 2008.

[9] Tibor Horvath et al. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers*, 56(4):444–458, 2007.

[10] ILOG, Inc. CPLEX, 2009. http://www.ilog.com/products/cplex/.

[11] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *ICAC '09*, pages 117–126, New York, NY, USA, 2009. ACM.

[12] Nagarajan Kandasamy et al. Self-optimization in computer systems via on-line control: Application to power management. In *ICAC '04*, pages 54–61, 2004.

[13] G. Khanna et al. Application performance management in virtualized server environments. *10th IEEE/IFIP Network Operations and Management Symposium*, pages 373–381, 0-0 2006.

[14] B. Khargharia et al. Autonomic power & performance management for large-scale data centers. *IEEE International Symposium on Parallel and Distributed Processing*, March 2007.

[15] B. Khargharia et al. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, 2008.

[16] Dara Kusic et al. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[17] McKinsey & Company. Revolutionizing data center efficiency. http://uptimeinstitute.org, 2008.

[18] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. A framework for dynamic adaptation of power-aware server clusters. In *SAC '09: Proceedings of the 24th ACM Symposium on Applied Computing*. ACM, 2009.

[19] Ted Ralphs. Branch Cut and Price Resource Web. http://www.branchandcut.org/, 2009.

[20] Cosmin Rusu et al. Energy-efficient real-time heterogeneous server clusters. In *RTAS'06*, pages 418–428, 2006.

[21] Vivek Sharma et al. Power-aware QoS management in web servers. In *RTSS'03*, pages 63–72, 2003.

[22] SPEC. SPECpower benchmark. http://www.spec.org/power_ssj2008/, 2008.

[23] Shekhar Srikantaiah et al. Energy aware consolidation for cloud computing. In *USENIX Workshop on Power Aware Computing and Systems*, 2008.

[24] The Apache Software Foundation. Apache HTTP server version 2.2. http://httpd.apache.org/docs/2.2/, 2008.

[25] A. Starikovskiy V. Palladi. The ondemand governor: past, present and future. *Proceedings of Linux Symposium*, 2:223–238, 2001.

[26] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of HPC applications. In *ICS '08*, pages 175–184, New York, NY, USA, 2008. ACM.

[27] Akshat Verma et al. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware'08*, pages 243–264, 2008.

[28] Yefu Wang et al. Power-efficient response time guarantees for virtualized enterprise servers. In *RTSS'08*, pages 303–312, 2008.