RELATÓRIOS DE PESQUISA EM ENGENHARIA DE PRODUÇÃO V.11 N.8

# A column generation approach for power-aware optimization of virtualized heterogeneous server clusters

Hugo Harry Kramer
Departamento de Engenharia de Produção – Universidade Federal Fluminense
Rua Passo da Pátria, 156, São Domingos, 22210-240, Niterói, RJ
hugoharry@gmail.com

Vinicius Petrucci
Instituto de Computação – Universidade Federal Fluminense
Rua Passo da Pátria, 156, São Domingos, 22210-240, Niterói, RJ
vpetrucci@ic.uff.br

Anand Subramanian
Instituto de Computação – Universidade Federal Fluminense
Rua Passo da Pátria, 156, São Domingos, 22210-240, Niterói, RJ
anand@ic.uff.br

Eduardo Uchoa
Departamento de Engenharia de Produção – Universidade Federal Fluminense
Rua Passo da Pátria, 156, São Domingos, 22210-240, Niterói, RJ
uchoa@producao.uff.br

## Abstract

Increasingly, clusters of servers have been deployed in large data centers to support the development and implementation of many kinds of services, having distinct workload demands that vary over time, in a scalable and efficient computing environment. Emerging trends are utility/cloud computing platforms, where many network services, implemented and supported using server virtualization techniques, are hosted on a shared cluster infrastructure of physical servers. The energy consumed to maintain these large server clusters became a very important concern, which in turn, requires major investigation of optimization techniques to improve the energy efficiency of their computing infrastructure.

In this work, we present a novel optimization approach that simultaneously deals with (1) CPU power-saving techniques combined with server switching on/off mechanisms, (2) the case of server heterogeneity, (3) virtualized server environments, (4) an efficient optimization method based on column generation techniques. The key aspects of our approach are the basis on rigorous and robust optimization techniques, given by high quality solutions in short amount of processing time, and experimental results on the cluster configuration problem for large-scale heterogeneous

Versão Final Recebida em 21/02/2011 - Publicado em 06/06/2011

server clusters that can make use of virtualization techniques.

**Keywords:** Column generation, Energy management, Server cluster virtualization.

**Resumo**

De maneira crescente, *clusters* de servidores têm sido empregados no apoio ao desenvolvimento e implementação de uma grande variedade de serviços, com demandas por processamento distintas e variáveis ao longo do tempo, em ambientes computacionais escaláveis e eficientes. Plataformas conhecidas como computação em nuvem, emergem como uma tendência no apoio e desenvolvimento de serviços em rede que são hospedados e compartilhados em uma estrutura de servidores físicos através de técnicas de virtualização. A energia consumida para manter tais clusters de servidores se tornou uma importante questão de interesse que, por sua vez, requer investigação no que compete à aplicação de técnicas de otimização com o objetivo de aprimorar a eficiência energética de tais infraestruturas computacionais.

Neste trabalho, é apresentada uma nova abordagem baseada em otimização que lida simultaneamente com (1) técnicas de economia de energia em CPU combinadas com mecanismos de ligar/desligar servidores, (2) o caso da heterogeneidade dos servidores, (3) ambientes de servidores virtualizados e (4) um método eficiente de otimização baseado em técnicas de geração de colunas. Os aspectos principais da nossa abordagem se baseiam em técnicas robustas de otimização, ao obter soluções de qualidade em baixo tempo computacional, além de uma série de resultados experimentais no problema de configuração de *clusters* de larga escala com servidores heterogêneos que utilizam técnicas de virtualização.

**Palavras-Chave:** Geração de colunas, Gerenciamento de energia, Virtualização de *clusters* de servidores.

## 1 Introduction

An increasing number of clusters of servers have been deployed in large data centers to support the development and implementation of many kinds of services supporting different applications in a scalable and efficient computing environment, for example, focused on Web-based applications. A typical server cluster is a distributed system that consists of hundreds or thousands of machines linked by a fast network [9]. These cluster architectures are becoming common in utility/cloud computing platforms [11, 20], such as Amazon EC2 and Google AppEngine. In these platforms, the network services are mostly hosted on several shared physical servers and can have distinct workloads that vary over time.

These cluster platforms usually have great processing and performance demands, incurring in high energy costs and indirectly contributing to increase $CO_2$ generation and then to the environmental deterioration [27]. The energy consumed to maintain these large data centers became a very important concern, which in turn, requires major investigation of optimization techniques to improve the energy efficiency of their computing infrastructure [6, 16, 34].

To support the execution of multiple independent network services, modern server cluster platforms (also known as cloud computing) rely on virtualization techniques to enable the usage of different virtual machines (VMs) — operating system plus software applications — on a single physical server. Server virtualization has been widely adopted in data centers around the world for improving resource usage efficiency; particularly helping to make these computing environments more energy-efficient. Several virtual machine monitors or hypervisors, which act as a layer between the virtual machine and the actual hardware, have been developed to support server virtualization, such as Xen [13] and VMware [19].

The adoption of virtualization technologies for power-aware optimization in server clusters turns out to be a challenging research topic. Specifically, the ability to dynamically distribute server workloads in a virtualized server environment allows for turning off physical machines during periods of low activity, and bringing them back up when the demand increases. Moreover, server on/off mechanisms can be combined with CPU DVFS (*Dynamic Voltage and Frequency Scaling*), which is a technique that consists of varying the frequency and voltage of the CPU at runtime according to processing needs, in order to provide even better power optimizations. Examples of DVFS capabilities implemented by current microprocessors are Intel's "Enhanced Speedstep Technology", and AMD's "PowerNOW!".

Recent studies show that servers in data centers are loaded between 10 and 50 percent of peak, with a CPU utilization that rarely surpasses 40 percent [3]. Thus, the consolidation of the different workloads of services in a cluster, using server virtualization techniques, can reduce energy consumption and increase the utilization of physical servers [26, 37, 40]. In addition, techniques for processors energy-saving, typically considered major consumers of power in a server, have been widely used in the literature [5, 22, 35]. However, to our knowledge, none of research studies currently integrate and evaluate these two main techniques for energy-saving in an optimization-based approach for virtualized heterogeneous server environments.

The case of heterogeneity in server cluster was addressed by [21]. However, their approach considers a non virtualized cluster and solves a different optimization problem. Some optimization approaches, based on the bin packing problem, for configuring virtualized servers are described in the literature, such as [7, 24]. However, their models are not designed for power-aware optimization. [40] present a two-layer control architecture aimed at providing power-efficient real-time guarantees for virtualized computing environments. Their work relies on a sound control theory based framework, but does not address optimizations for server on/off mechanisms and virtual machine allocation in a server cluster context. A heuristic-based solution for the power-aware consolidation problem of virtualized clusters, without adopting DVFS, is presented in [37], but it does not show any analysis and guarantees to find solutions that are at least near to the optimal.

A dynamic resource provisioning framework is developed in [26] based on lookahead control. Their approach does not consider DVFS, but the proposed optimization controller addresses attractive issues, such as switching machines costs (i.e., overhead of turning servers on and off) and predictive configuration model. A power-aware migration framework for virtualized HPC (High-performance computing) applications, which accounts for migration costs during virtual machine reconfigurations, is presented in [39]. Similarly to our approach, it relies on virtualization techniques used for dynamic consolidation, although the application domains are different and DVFS technique is not employed.

An approach based on DVFS is presented in [22] for power optimization and end-to-end delay control in multi-tier web servers. Recent approaches, such as presented in [5, 15, 23, 25, 35, 36] also rely on DVFS techniques and include server on/off mechanisms, based on the seminal papers of [30] and [10], for power-aware optimizations. However, these approaches are not designed (and not applicable) for virtualized server clusters. That is, they do not consider the optimization of multiple service workloads in a shared cluster infrastructure. The problem of optimally allocating a power budget among servers in a cluster in order to minimize mean response time is described in [18]. In contrast to our optimization goal, which minimizes the cluster power consumption while meeting performance requirements, their problem poses a different optimization goal and is not applied to a virtualized cluster. However, their models of server power and performance to implement power management techniques are similar to ours.

It should be noted that solving the power optimization problem to optimality may be economically significant since a difference of 10% of power cost for running a large-scale data center can be worth a lot of money per year for a business. An optimization approach at least should provide some guarantees on how far the solution can be from the optimal. Most of related works found in the literature do not provide such analysis and assurances.

To the best of our knowledge, this is a novel approach that simultaneously deals with (1) the CPU DVFS technique combined with server on/off mechanisms, (2) the case of server heterogeneity, (3) virtualized server environments, (4) an efficient optimization approach based on Column Generation (CG) techniques. The key aspects of our approach are the basis on rigorous and robust optimization techniques, given by high quality solutions in short amount of processing time, and experimental results on the cluster configuration problem for large-scale heterogeneous server clusters that can make use of virtualization techniques.

The remainder of the paper proceeds as follows. Section 2 formaly describes the optimzation problem dealt in the current work. Section 3 explains the proposed CG approach. Section 4 presents and discusses the results obtained. Section 5 contains the concluding remarks of this work.

## 2 Optimization Problem

The cluster optimization problem is to determine the most efficient configuration of servers in terms of power consumption that meets the required performance of services provided by multiple services on a shared cluster of servers, as well as to decide on the best mapping of services to servers. The goal of our optimization approach is to reduce power consumption in the virtualized cluster while meeting service performance demands.

To make decisions on which CPU frequency a server must operate, our approach relies on DVFS technique available on current microprocessors. This technique allows for dynamically adjusting the performance states (P-States) at which the server can operate when the CPU is active, which consists of a predefined set of frequency and voltage combinations. As an example of this power management capability, Table 1 shows the P-states and power consumption for the Intel Pentium M 1.6 GHz processor, whose data are available on a White Paper from Intel[1]. Notice that performance state $P0$ is the highest P-state and $Pn$ is the lowest one.

Table 1: Example of CPU operating states and power consumption (Intel Pentium M 1.6 GHz)

| P-state | Frequency (GHz) | Voltage (V) | Power (Watts) |
|---|---|---|---|
| P0 | 1.6 | 1.484 | 25 |
| P1 | 1.4 | 1.42 | 17 |
| P2 | 1.2 | 1.276 | 13 |
| P3 | 1 | 1.164 | 10 |
| P4 | 0.8 | 1.036 | 8 |
| P5 | 0.6 | 0.956 | 6 |

With DVFS technique the processor's frequency, which is associated with the operating voltage, can be automatically adjusted at run time, decreasing the power consumption and reducing the amount of heat generated on the chip. Since DVFS enables the processor to change it's operating frequency, it results in a corresponding change in performance due to reduction in the number of instructions the processor can issue in a given amount of time. This poses relevant power and performance optimization trade-offs to consider in server clusters, depending on the particular server hardware capabilities, the incoming service demand workload, and power and performance management goals.

### 2.1 Cluster Configuration

A cluster configuration solution is given by (1) which servers must be active and their respective CPU frequencies and (2) a corresponding mapping of the

[1]Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor: `ftp://download.intel.com/design/network/papers/30117401.pdf`

services (running on top of VMs) to physical servers. The cluster configuration must be carefully employed in the cluster, considering individual service performance levels, in order to guarantee QoS (*Quality of Service*) related to their corresponding SLA's (*Service Level Agreements*). The configuration problem is far to be simple since large clusters include many heterogeneous machines and each machine could have different capacity and power-consumption according to the number of CPU cores, their frequencies, their specific devices, and so forth. According to [21], server clusters are typically composed of heterogeneous servers.

Additionally, the incoming workload of multiple services in a server cluster can significantly change over time. This requires the optimization problem to be solved periodically and the solution is used to configure the cluster. The optimization algorithm proposed, thus, have a time-constrained processing requirement, considering the usual cluster configuration control period (e.g., few minutes).

To implement the optimization solution over time, considering that the cluster services have individual time-varying workloads, we need an optimization strategy to enable the virtualized server system to react to load variations and adapt its configuration accordingly. We refer to [29] for the optimization control loop design and implementation, which relies on the underlying cluster optimization problem investigated in this work.

## 2.2 Mathematical Formulation

The optimization problem of selecting the most efficient configuration of a *virtualized server cluster* can be defined as follows: let $N = \{1, 2, ..., i, ..., n\}$ be the set of *server types* of the cluster, $S_i = \{1, 2, ..., s, ..., q_i\}$ are the sets of *servers* of type $i \in N$, and $F_i = \{1, 2, ..., j, ..., r_i\}$ are the sets of available CPU *frequencies* for each server of type $i \in N$. Without lack of generality, we assume that the frequencies in $F_i, i \in N$ are in increasing order and, thus, $r_i$ is the highest frequency of a server type $i \in N$. The set of *services* intended to run on the server cluster is denoted by $K = \{1, 2, ..., k, ..., m\}$. Each server of type $i \in N$ running at a CPU frequency $j \in F_i$ has a *capacity* (e.g., requests/s) $D_{ij}$ and a *cost* (e.g., power consumption) given by $C_{ij}$. The *demand* (e.g., requests/s) of each service $k \in K$ is given by $d_k$.

The decision variables are defined as follows: $y_{isj}$ is a binary variable which indicates whether a server $s \in S_i$ of type $i \in N$ is running at CPU frequency $j \in F_i$, and $x_{isjk}$ is a binary variable which indicates if a service $k \in K$ is assigned to a server $s \in S_i$ of type $i \in N$ running at CPU frequency $j \in F_i$.

The problem formulation as a ILP (Integer Linear Programming) problem is thus given as follows:

$$(\mathcal{F}_1) \quad z = \; Min \sum_{i \in N} \sum_{s \in S_i} \sum_{j \in F_i} C_{ij} y_{isj} \qquad (1)$$

Subject to

$$\sum_{k\in K} d_k x_{isjk} \le D_{ij} y_{isj} \qquad \forall i \in N, s \in S_i, j \in F_i \tag{2}$$

$$\sum_{i\in N}\sum_{s\in S_i}\sum_{j\in F_i} x_{isjk} = 1 \qquad \forall k \in K \tag{3}$$

$$\sum_{j\in F_i} y_{isj} \le 1 \qquad \forall i \in N, s \in S_i \tag{4}$$

$$x_{isjk} = \{0,1\} \qquad \forall i \in N, s \in S_i, j \in F_i, k \in K \tag{5}$$

$$y_{isj} = \{0,1\} \qquad \forall i \in N, s \in S_i, j \in F_i. \tag{6}$$

The objective function given by (1) aims to find a cluster configuration that minimizes the operational cost of the server cluster in terms of power consumption. Constraints (2) avoid the capacity of a server $s \in S_i$ of type $i \in N$ running at CPU frequency $j \in F_i$ to be exceeded. Constraints (3) ensure that each service $k \in K$ is assigned to run in the cluster. The constraints given by (4) state that a server of type $i \in N$ is allowed to run at most in one CPU frequency $j \in F_i$. The constraints (5) and (6) define the domain of the variables of the problem.

The cluster configuration problem is NP-hard, since it can be seen as a generalization of the 1-D Variable-sized Bin Packing Problem, which is known to be NP-hard [17], when each type of server has an unlimited number and only a single frequency available. Clearly, it is not practical to use $\mathcal{F}_1$ directly for configuring a large cluster, since the number of variables and constraints would be far too large to be handled by a Mixed Integer Programming (MIP) solver. However, in order to obtain a Lower Bound (LB) on the optimal power consumption, we propose solving a much smaller ILP relaxation of the problem, where demands are allowed to be split into different servers. Note that this case of service fragmentation is not a valid assumption to our optimization problem. Let $v_{ij}$ be an integer variable that indicates how many servers of type $i \in N$ run at frequency $j \in F_i$. The Model $\mathcal{F}_2$ is as follows.

$$(\mathcal{F}_2) \quad z = Min \sum_{i\in N}\sum_{j\in F_i} C_{ij} v_{ij} \tag{7}$$

Subject to

$$\sum_{j\in F_i} v_{ij} \le q_i \qquad \forall i \in N \tag{8}$$

$$\sum_{i\in N}\sum_{j\in F_i} D_{ij} v_{ij} \ge \sum_{k\in K} d_k \tag{9}$$

$$v_{ij} \in \mathbb{Z}_+ \qquad \forall i \in N, j \in F_i. \tag{10}$$

The objective function (7) minimizes the sum of the costs. Constraints (8) ensure that the number of servers of type $i \in N$ does not exceed $q_i$. Constraint

(9) guarantees that the total capacity of all servers is greater or equal than the sum of the demands of all services. Constraints (10) define the domain of the decision variables. It can be proven that $z_{\mathcal{F}_2} \leq z_{\mathcal{F}_1}$. Although this formulation is not meant to provide a feasible solution to our problem, its solution cost can be used to measure the quality of a given feasible solution.

## 3 Column Generation Approach

In this section we describe our proposed solution approach which aims at obtaining integer feasible solutions via CG based algorithms. As the numbers of servers and services increase, the direct use of formulation $\mathcal{F}_1$ to obtain integer feasible solutions for the problem becomes prohibitive, due to its large number of variables and constraints. To handle this issue, we can reformulate the problem applying a Dantzig-Wolfe Decomposition [12]. The linear relaxation of the resulting master problem can be solved by a CG based algorithm. To use this approach, the following definitions must be considered:

**Definition 1.** We define a *pattern* as the incidence vector in $\{0,1\}^m$ associated to a subset of $K$. For a pattern $p$, $p_k$ denotes the $k$-th component of $p$. Define $P(i)$ as the set of all feasible patterns for a server of type $i$, i.e., those such that

$$\sum_{k \in K} p_k d_k \leq D_{ir_i} \qquad \forall i \in N, \tag{11}$$

where $D_{ir_i}$ is the maximum capacity of a server of type $i \in N$ and is related to its highest frequency.

**Definition 2.** $f(p,i)$ is the lowest frequency such that $\sum_{k \in K} p_k d_k \leq D_{if(p,i)}$.

Given the above, the reformulation of the problem by Dantzig-Wolfe Decomposition results in the following problem:

$$Min \sum_{i \in N} \sum_{p \in P(i)} C_{if(p,i)} \lambda_{ip} \tag{12}$$

Subject to

$$\sum_{i \in N} \sum_{p \in P(i)} p_k \lambda_{ip} = 1 \qquad \forall k \in K \tag{13}$$

$$\sum_{p \in P(i)} \lambda_{ip} \leq q_i \qquad \forall i \in N \tag{14}$$

$$\lambda_{ip} = \{0,1\} \qquad \forall i \in N, p \in P(i). \tag{15}$$

In this formulation, a binary variable $\lambda_{ip}$ is 1 if the services in the pattern $p \in P(i)$ are assigned to a server of type $i \in N$ running at frequency $f(p,i)$.

## 3.1 Row Aggregation

Formulation (12-15) has less rows than $\mathcal{F}_1$. In order to obtain an even more compact formulation, we can aggregate the rows that correspond to services with identical demands. Therefore, we define $K' = \{1, 2, ..., k, ..., m'\}$ as the set of indices of the groups of aggregated services and $n_k$ as the number of services with demand $d_k$. In addition, the set $K'$ is sorted in decreasing order according to the values of demands $d_k$, $k \in K'$. Furthermore, we rewrite Definition 1 as follows.

**Definition 3.** We define a *pattern* as the incidence vector in $\mathbb{Z}_+^{m'}$ associated to a subset of $K'$. For a pattern $p$, $p_k \leq n_k$ denotes the $k$-th component of $p$. Define $P(i)$ as the set of all feasible patterns for a server of type $i$, i.e., those such that

$$\sum_{k \in K'} p_k d_k \leq D_{ir_i} \qquad \forall i \in N, \tag{16}$$

where $D_{ir_i}$ is the maximum capacity of a server of type $i \in N$ and is related to its highest frequency.

Thus, by using this row aggregation scheme we can replace constraints (13) with (17). Moreover, relaxing the integrality of $\lambda$ variables leads us to the so-called Master Problem (MP) which is defined by (12), (14), (17) and (18). Note that constraints (17) are also relaxed to greater than or equal, since this does not change the lower bound provided by MP, but it helps on the convergence of the CG.

$$\sum_{i \in N} \sum_{p \in P(i)} p_k \lambda_{ip} \geq n_k \qquad \forall k \in K' \tag{17}$$

$$\lambda_{ip} \geq 0 \qquad \forall i \in N, p \in P(i). \tag{18}$$

This resulting formulation is a generalization of the Multiple Length Cutting-Stock Problem [1, 4].

## 3.2 Exchange Variables

In addition to the row aggregation scheme, the convergence of the CG algorithm can be even more enhanced if we add a small number of exchange variables to MP resulting in a extended formulation for the problem.

The additional variables $t_k \in \mathbb{Z}_+$ represent the replacement of a certain number of aggregated services $k$ by the same amount of aggregated services $k-1$, $k \in K'$. These variables always exchange a service with a given demand by another one immediately smaller, which in turn can be exchanged similarly by another one and so forth. As a result, each $\lambda$ variable added to the MP, normally used in solutions that have exactly that unique pattern, can now be used in other solutions that make use of different patterns with smaller demands. This is equivalent to add inequalities to the dual problem, i.e., restricting its solution space, as described by [38].

We now modify the MP by replacing (17) with (19) and by adding the domain constraints (20), leading us to an extended master problem called $\text{MP}_{\text{ex}}$.

$$\sum_{i \in N} \sum_{p \in P(i)} p_k \lambda_{ip} - t_k + t_{k-1} \geq n_k \qquad \forall k \in K' \tag{19}$$

$$t_k \geq 0 \qquad \forall k \in K'. \tag{20}$$

## 3.3 Restricted Master Problem (RMP)

Due to the large number of columns in a master problem obtained by Dantzig-Wolfe Decomposition we should deal implicitly with such columns, which is the fundamental idea of the CG algorithm. Hence, we consider only a small version of the master problem, called RMP, which has a limited number of columns sufficient to hold feasibility. The columns are generated by solving a subproblem, known as pricing subproblem, and only the “interesting” ones are added to RMP.

## 3.4 Pricing Subproblem

Let $\alpha_i$, $i \in N$, be the dual variables associated to constraints (14) and let $\pi_k$, $k \in K'$, be the dual variables associated to constraints (17), in the case of MP, and to constraints (19), in the case of $\text{MP}_{\text{ex}}$. From Definitions 2 and 3, for each pair $(i, j)$, $i \in N$ and $j \in F_i$, we can formulate the pricing subproblem $\text{SP}_{ij}$ as follows.

$$(\text{SP}_{ij}) \ Min \ C_{ij} - \alpha_i - \sum_{k \in K'} \pi_k p_k \tag{21}$$

Subject to

$$\sum_{k \in K'} d_k p_k \leq D_{ij} \tag{22}$$

$$p_k \in \mathbb{Z}_+ \qquad \forall k \in K'. \tag{23}$$

This problem can be seen as an Integer Knapsack Problem and, in spite of being NP-hard, it is known to be quite well solved by pseudo-polynomial algorithms. The solution given by $\text{SP}_{ij}$ in terms of $p_k$, $k \in K'$, represents a pattern for the servers of type $i \in N$ running at a CPU frequency $j \in F_i$. In a CG algorithm, the columns are generated by the pricing subproblem and added to the RMP only if the value of the objective function (21), which represents the reduced cost of the column, is negative.

## 3.5 Generating Initial Basis

In order to generate an initial feasible basis for the CG algorithms, we developed a simple greedy heuristic (Alg. 1) which works as follows. Let SL be a list composed of all the servers and let AL be a list composed of all the application services (line 3). At first, we sort SL according to the maximum capacities and AL according to the demands, both in descending order (line 4). Next, while AL is not empty, we try to assign the remaining services to the maximum frequency of the current first element of SL, always giving preference to those with larger demands (lines 7-14). Finally, we add the $\lambda$ variables (patterns) associated to the complete feasible solution of the problem (line 15).

**Algorithm 1** GenerateInitialBasis()

```
1:  RMP ← ∅
2:  sol ← ∅
3:  Initialize Servers List (SL) and Application Services List (AL)
4:  Sort SL and AL in descending order
5:  while AL is not empty do
6:      Remove the first element of SL (currentServer)
7:      while at least one service $k \in$ AL can be assigned to the maximum
        frequency of currentServer do
8:          for $k = 1 \dots |\text{AL}|$ do
9:              Try to assign element $k \in$ AL to currentServer
10:         end for
11:         Update partial solution sol by including the assigned services
12:         Update AL by removing the assigned services
13:     end while
14: end while
15: Update RMP by adding the $\lambda$ variables (patterns) associated to sol
16: return RMP
```

## 3.6 The Column Generation Algorithm

The pseudocode of the CG algorithm is presented in Alg. 2. We assume that both the LP and SP (Knapsack Problem) solvers are black-boxes and that the initial RMP basis is provided by the heuristic described in the previous subsection. In the case of $\text{MP}_{\text{ex}}$, the exchange variables $t$ must also be included to the initial RMP. Let *colsAdded* be a boolean variable that has value true if a column was added in a given iteration and false otherwise. We first solve the initial RMP (line 3). While *colsAdded* is true, for each pair $(i, j)$, $i \in N$, $j \in F_i$, we solve the corresponding $\text{SP}_{ij}$ and, if necessary, we add the generated columns and solve the updated RMP (lines 4-19).

**Algorithm 2** ColumnGeneration(RMP)

```
1: $sol_{SP} \leftarrow \emptyset$
2: $colsAdded \leftarrow$ true
3: $sol \leftarrow$ LPsolver(RMP)
4: **while** $colsAdded$ **do**
5:     $colsAdded \leftarrow$ false
6:     **for** each type of server $i \in N$ **do**
7:         **for** each frequency $j \in F_i$ **do**
8:             Update SP$_{ij}$ with the values of $\alpha_i$ and $\pi_k$, $\forall k \in K$
9:             $sol_{SP} \leftarrow$ SPsolver(SP$_{ij}$)
10:            **if** $f(sol_{SP}) < 0$ **then**
11:                Add corresponding column $\lambda$ to RMP
12:                $colsAdded \leftarrow$ true
13:            **end if**
14:        **end for**
15:    **end for**
16:    **if** $colsAdded$ **then**
17:        $sol \leftarrow$ LPsolver(RMP)
18:    **end if**
19: **end while**
20: **return** $sol$
```

## 3.7 Primal Heuristics

Two CG heuristics were developed for obtaining feasible primal solutions. The first one is ILP-based, while the second one is LP-based.

### 3.7.1 Restricted Master Integer Program Heuristic (RMIPH)

The RMIPH is a straightforward approach that modifies the final Restricted Master Linear Program (RMLP) by setting all variables to be integral and solves the resulting Restricted Master Integer Program (RMIP). Nevertheless, it is important to mention that the RMIP may not necessarily contain a feasible integer solution. Also, this method strongly depends on a efficient MIP solver for obtaining optimal/near-optimal solutions in an acceptable computational time.

### 3.7.2 Rounding Heuristic (RH)

Unlike the RMIPH, the RH is not ILP-based and therefore it does not rely on any MIP solver. The idea of this heuristic is to obtain a feasible integer solution by rounding up the lower bound of the fractional variables of a CG solution. The outline of the heuristic is presented in Alg. 3, where it can be seen that the proposed solution approach has three steps: (i) generation of an initial basis; (ii) construction of a feasible integer solution; and (iii) a refinement procedure.

**Algorithm 3** RoundingHeuristic()

```
1: $sol \leftarrow \emptyset$
2: RMP $\leftarrow$ GenerateInitialBasis()
3: $sol \leftarrow$ Construction(RMP)
4: $sol \leftarrow$ Refinement(RMP, $sol$)
5: return $sol$
```

The pseudocode of the constructive procedure is presented in Alg. 4. In this case, let $i \in \{1, \ldots, |N|\}$ be the index associated to a type of server and we assume that the larger the value of $i$, the smaller is the capacity of the server running at its highest CPU frequency. At first, we set $i = |N|$ (line 1). Let $maxCols$ be the maximum number of columns allowed during the construction phase and after preliminary experiments we adopted $maxCols = 1000$ (line 2). Next, we build an initial solution, which is possibly fractional, by solving the CG algorithm (line 3). In this case, the pricing subproblem is firstly solved using a 0-1 Knapsack solver and, when it is no more possible to add binary columns, we then use an Integer Knapsack solver until the convergence of the GC algorithm. While the solution is fractional we try to generate a feasible integer solution by modifying the fractional solution of the CG (lines 4-18). If the CG solution is infeasible or it is no more possible to round up the lower bound of at least one $\lambda$ variable in such a way that the constraints of the problem are not violated, we unfix those variables that are associated with server $i$ (lines 5-6). This is done by restoring the bounds of the original $\lambda$ variables that are associated with server $i$, that is, $\lambda \geq 0$. We then update the value of $i$ by decreasing its value or by setting $i = |N|$ if $i = 0$ (lines 7-10). If a feasible fractional solution is found we select a variable to round up according to the following criterion. Let $\mathcal{P}$ be the set of variables that have fractional values in a CG solution. We choose to round up the lower bound of the variable $\lambda_p$, $p \in \mathcal{P}$, associated with $g^{min} = \min\{g(p) \mid \forall p \in \mathcal{P}\}$, where $g(p) = \lceil \lambda_p \rceil - \lambda_p$ (line 12). If the number of columns of RMP is larger than $maxCols$ we remove the columns associated to the $\lambda$ variables that are out of the basis (line 15). Finally, we solve the CG algorithm (line 17). Here, in contrast to what occurs in line 3, we only use the 0-1 Knapsack solver.

A refinement procedure is applied to the solution obtained by the constructive routine (see Alg. 5). The improvement approach works as follows. Let RMP$'$ be a backup of the RMP basis (lines 2-3). For each server $i \in N$ we unfix the $\lambda$ variables related to $i$ and we call the constructive algorithm as an attempt of producing an improved feasible integer solution (lines 4-11). In case of improvement we update the best solution (lines 7-9). At the end of each iteration we update RMP using the model stored in RMP$'$ (line 10). During the refinement procedure, we only use the 0-1 Knapsack solver.

**Algorithm 4** Construction(RMP)

```
1: $i \leftarrow |N|$
2: $maxCols \leftarrow 1000$
3: $sol \leftarrow$ ColumnGeneration(RMP)
4: while $sol$ is fractional do
5:   if $sol$ is infeasible or no more $\lambda$ variables can be rounded up then
6:     Update RMP by restoring the original bounds of those variables, i.e. $\lambda \geq 0$, associated to server $i$ that are currently fixed
7:     $i \leftarrow i - 1$
8:     if $i = 0$ then
9:       $i = |N|$
10:    end if
11:  else
12:    Update RMP by rounding up the lower bound of fractional variable $\lambda_p$ associated to $g^{min} = \min\{\lceil\lambda_p\rceil - \lambda_p \mid \forall p \in \mathcal{P}\}$
13:  end if
14:  if number of columns of RMP $> maxCols$ then
15:    Remove columns associated to $\lambda$ variables that are out of the basis
16:  end if
17:  $sol \leftarrow$ ColumnGeneration(RMP)
18: end while
19: return $sol$
```

**Algorithm 5** Refinement(RMP, *sol*)

```
1: $sol^* \leftarrow sol$
2: Update RMP by removing columns associated to $\lambda$ variables that are out of the basis
3: RMP$' \leftarrow$ RMP
4: for each type of server $i \in N$ do
5:   Update RMP by restoring the original bounds of those variables, i.e. $\lambda \geq 0$, associated to server $i$ that are currently fixed
6:   $sol \leftarrow$ Construction(RMP)
7:   if $f(sol) < f(sol^*)$ then
8:     $sol^* \leftarrow sol$
9:   end if
10:  RMP $\leftarrow$ RMP$'$
11: end for
12: return $sol^*$
```

## 4 Computational Results

In this section we present the computational results obtained to evaluate our optimization approach. As a first phase we provide real measurements of power consumption and processing capacity for a set of server types, which are described in Section 4.1. These measurements are used as input data to guide the generation of test instances introduced in Section 4.2. In Section 4.3 we present and discuss the results obtained by the proposed solution approach in the test instances generated.

## 4.1 Server capacity and power measurements

The test instances used to evaluate our approach are derived from power consumption and processing capacity measurements of four different server types. The server types used have different technologies including a wide range of working frequencies. For each server type at each CPU frequency available, we measured the average power consumption and processing capacity.

The processing capacity of a server is linearly proportional to its CPU frequency. The power consumption of the server also increases with frequency, but the curve follows a non-linear growth. Typically this curve starts with a sub-linear growth, and from a certain point there is a shift in higher frequencies and the growth would be approximately quadratic.

We measured AC power directly using the *WattsUP Pro* power meter with 1% accuracy [14]. The power measurements thus represent the whole machines, not only their CPUs. Table 2 shows the power measured values for each type of server used in our work.

Table 2: Power consumption (in Watts) for each server at each CPU frequency

| **Server type** | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ |
|---|---|---|---|---|---|---|---|---|---|
| Intel Core i7 | 154 | 158 | 163 | 168 | 175 | 181 | 189 | 198 | 211 |
| AMD Phenom II X4 | 118 | 151 | 175 | 217 | – | – | – | – | – |
| Intel Core2Duo | 111 | 132 | – | – | – | – | – | – | – |
| AMD Athlon 64 X2 | 88 | 106 | 111 | 121 | 132 | 146 | 161 | 180 | – |

The server processing capacity was measured, for each frequency available, in terms of the number of web requests per second that the given server can handle at maximum CPU utilization. The requests used in the experiments are CPU-bound and consume a fixed amount of CPU time. To generate the server benchmark workload, we used the *httperf* tool [28]. Table 3 shows the measured values of processing capacity for the server types.

In our case, recording the average values for power and processing capacity over a period of 2 minutes was sufficient to obtain a good average and low standard deviation. Note that the case of web servers is similar to any other CPU-intensive services in a cluster and different kinds of server capacity metrics could be used, such as MFLOP (Million Floating Point Operations Per Second).

Table 3: Processing capacity (in requests per second) for each server at each CPU frequency

| **Server type** | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ |
|---|---|---|---|---|---|---|---|---|---|
| Intel Core i7 | 820 | 893 | 965 | 1033 | 1102 | 1171 | 1238 | 1304 | 1373 |
| AMD Phenom II X4 | 300 | 792 | 938 | 1187 | – | – | – | – | – |
| Intel Core2Duo | 474 | 716 | – | – | – | – | – | – | – |
| AMD Athlon 64 X2 | 128 | 230 | 256 | 279 | 303 | 329 | 353 | 378 | – |

## 4.2 Test instances

We generated a set of instances to evaluate our approach given the power consumption and processing capacity measurements of the server types. Specifically, to construct a test instance, we need the following data inputs:

- Number of server types;
- Number of servers for each type;
- Number of services to run in the cluster;
- The interval in which the demand values will be generated.

Each test instance has the following notation: ct-$G$-$X$-$Y$-$Zw$. The parameter $G$ specifies a given group of instances. The parameter $X$ denotes the number of server types in the cluster, where the number of services to run is given by $Y \in \{1000, 2000, 4000\}$. The parameter $Z \in \{a, b, c\}$ determines the discrete interval in which the services demands were generated, where $a = \{1, ..., 100\}$, $b = \{20, ..., 100\}$ and $c = \{50, ..., 100\}$. In fact, since there is usually a precision limitation when estimating a service demand, in most cases it is reasonable to consider that even when there are several thousands demands, the number of distinct demand values do not grow beyond a few hundreds. Lastly, the parameter $w$ is used to make a distinction between instances with equal $X$, $Y$, and $Z$ parameters.

Two groups of instances were generated, where each of these contains 18 test-problems. In the fist group ($G = 1$), the number of CPU frequencies is associated to each server type and such frequencies define the costs and capacities as can be seen in Tables 2 and 3, respectively. Finally, the demand values for each service were randomly selected within the specified interval. As for the second group ($G = 2$), we divided the capacities of each type of server by a constant value, in our case it was 5, and we multiplied the number of available servers of each type also by this constant value. Thus, the difference between the two groups of instances is the relationship between the service demands and the processing capacities of the servers, so that this ratio is higher in instances of the second group.

## 4.3 Results and Discussion

The proposed algorithms were coded in C++ and the tests were executed in an Intel Core i7 with 2.66 GHz and 4 GB of RAM running under Linux 64 bits. Only a single thread was used in our experiments. Linear and integer programs were solved using CPLEX 12.1, while pricing subproblems were solved using the Minknap[2] algorithm [31] for the 0-1 Knapsack problems and the Bouknap[2] algorithm [32] for the Integer Knapsack problems.

In the tables presented hereafter, **Instance** denotes the test-problem, **LB** is the lower bound obtained by the respective approach, **Time** indicates the CPU time in seconds, **Iter** is the number of iterations of the CG, **Cols** corresponds to the number of columns generated, **Best LB** is the best LB obtained, **Gap (%)** denotes the gap between the UB and the best LB.

Table 4 illustrates the results obtained by the CG approaches and those found by the formulation $\mathcal{F}_2$ in the first set of instances. It can be seen that $\mathcal{F}_2$ was able to produce better LBs in less computational time when compared to MP and MP$_{\text{ex}}$. Moreover, we can observe that, except for the instance ct-1-4-4000-b1, the CG based on MP$_{\text{ex}}$ was always capable of generating less columns and to perform less iterations than the one based on MP.

Table 4: Results obtained by $\mathcal{F}_2$ and the CG approaches in the first group of instances

| Instance | $\mathcal{F}_2$ | | MP | | | | MP$_{\text{ex}}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | LB | Time | LB | Time | Iter | Cols | LB | Time | Iter | Cols |
| ct-1-4-1000-a1 | **8748** | <0.01 | 8742 | 1.02 | 135 | 1247 | 8742 | **0.45** | **118** | **494** |
| ct-1-4-1000-a2 | **8237** | <0.01 | 8231 | 0.75 | 113 | 1127 | 8231 | **0.39** | **103** | **448** |
| ct-1-4-1000-b1 | **10105** | <0.01 | 10103 | 0.23 | 73 | 650 | 10103 | **0.14** | **60** | **295** |
| ct-1-4-1000-b2 | **10128** | <0.01 | 10123 | 0.22 | 67 | 623 | 10123 | **0.14** | **53** | **265** |
| ct-1-4-1000-c1 | **12961** | <0.01 | 12955 | 0.03 | 31 | 210 | 12955 | **0.02** | **30** | **190** |
| ct-1-4-1000-c2 | **12890** | <0.01 | 12889 | 0.03 | 28 | 188 | 12889 | **0.02** | **27** | **177** |
| ct-1-4-2000-a1 | **17109** | <0.01 | 17104 | 0.18 | 56 | 486 | 17104 | **0.12** | **50** | **282** |
| ct-1-4-2000-a2 | **17191** | <0.01 | 17188 | 0.16 | 57 | 431 | 17188 | **0.14** | **56** | **303** |
| ct-1-4-2000-b1 | **20541** | <0.01 | 20535 | **0.02** | 31 | 235 | 20535 | 0.05 | **29** | **218** |
| ct-1-4-2000-b2 | **20299** | <0.01 | 20295 | **0.04** | 33 | 240 | 20295 | **0.04** | **30** | **222** |
| ct-1-4-2000-c1 | **30582** | <0.01 | 30579 | 0.03 | 16 | 293 | 30579 | **0.02** | **14** | **267** |
| ct-1-4-2000-c2 | **30896** | <0.01 | 30895 | **0.02** | 17 | 314 | 30895 | 0.03 | **15** | **275** |
| ct-1-4-4000-a1 | **33913** | <0.01 | 33907 | **0.06** | 30 | 281 | 33907 | **0.06** | **29** | **244** |
| ct-1-4-4000-a2 | **34364** | <0.01 | 34359 | **0.07** | 29 | 293 | 34359 | **0.07** | **28** | **248** |
| ct-1-4-4000-b1 | **40946** | <0.01 | 40944 | **0.04** | **24** | **294** | 40944 | 0.06 | 25 | 302 |
| ct-1-4-4000-b2 | **40597** | <0.01 | 40594 | 0.05 | 21 | 279 | 40594 | **0.04** | **19** | **273** |
| ct-1-4-4000-c1 | **51809** | <0.01 | 51805 | **0.03** | 25 | 401 | 51805 | **0.03** | **22** | **380** |
| ct-1-4-4000-c2 | **51400** | <0.01 | 51395 | **0.03** | 25 | 397 | 51395 | 0.04 | **24** | **385** |
| Average | | <0.01 | | 0.17 | 45.06 | 443.83 | | **0.10** | **40.67** | **292.67** |

Table 5 shows the results found in the second group of instances. We can observe that, in terms of LBs, $\mathcal{F}_2$ obtained the best values, except for the instances ct-2-4-2000-c1 and ct-2-4-2000-c2. As for the number of columns and iterations, we can notice that the CG based on MP$_{\text{ex}}$ obtained, on average, smaller values than those of MP.

From the results presented in Tables 4 and 5, we can verify that the inclusion of the exchange variables helped to accelerate the convergence of the CG algo-

[2]The source code is available at `www.diku.dk/~pisinger/codes.html`

Table 5: Results obtained by $\mathcal{F}_2$ and the CG approaches in the second group of instances

| Instance | $\mathcal{F}_2$ | | MP | | | | MP$_{ex}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | LB | Time | LB | Time | Iter | Cols | LB | Time | Iter | Cols |
| ct-2-4-1000-a1 | **43827** | <0.01 | 43826 | **0.11** | **72** | 459 | 43826 | 0.14 | 77 | **445** |
| ct-2-4-1000-a2 | **41272** | <0.01 | 41268 | 0.13 | 75 | 437 | 41268 | **0.12** | **68** | **401** |
| ct-2-4-1000-b1 | **50645** | <0.01 | 50641 | **0.04** | **33** | **404** | 50641 | 0.07 | 37 | 408 |
| ct-2-4-1000-b2 | **50743** | <0.01 | 50741 | **0.04** | 36 | 413 | 50741 | 0.07 | **35** | **395** |
| ct-2-4-1000-c1 | **64926** | <0.01 | 64922 | **0.02** | 30 | 562 | 64922 | 0.03 | **28** | **527** |
| ct-2-4-1000-c2 | **64597** | <0.01 | 64594 | 0.03 | 35 | 591 | 64594 | **0.02** | **29** | **533** |
| ct-2-4-2000-a1 | **85735** | <0.01 | 85734 | 0.07 | 31 | 537 | 85734 | **0.06** | **30** | **531** |
| ct-2-4-2000-a2 | **86157** | <0.01 | 86155 | **0.07** | 37 | 573 | 86155 | 0.08 | **32** | **549** |
| ct-2-4-2000-b1 | **102921** | <0.01 | 102913 | **0.05** | 35 | 741 | 102913 | 0.06 | **32** | **712** |
| ct-2-4-2000-b2 | **101720** | <0.01 | 101712 | **0.06** | **29** | **705** | 101712 | **0.06** | 32 | 708 |
| ct-2-4-2000-c1 | 153580 | <0.01 | **154519** | 0.03 | 19 | 1126 | 154519 | **0.02** | **15** | **1086** |
| ct-2-4-2000-c2 | 155183 | <0.01 | **156179** | **0.02** | 18 | 1119 | 156179 | 0.04 | **17** | **1100** |
| ct-2-4-4000-a1 | **169976** | <0.01 | 169968 | 0.10 | **33** | **886** | 169968 | **0.09** | **33** | **886** |
| ct-2-4-4000-a2 | **172240** | <0.01 | 172232 | **0.08** | **32** | 898 | 172232 | **0.08** | **32** | **886** |
| ct-2-4-4000-b1 | **205212** | <0.01 | 205208 | **0.07** | **31** | 1213 | 205208 | **0.07** | 32 | **1210** |
| ct-2-4-4000-b2 | **203468** | <0.01 | 203455 | **0.07** | 33 | 1210 | 203455 | **0.07** | **31** | **1192** |
| ct-2-4-4000-c1 | **259590** | <0.01 | **259590** | 0.09 | 33 | 1838 | 259590 | **0.08** | **29** | **1795** |
| ct-2-4-4000-c2 | **257546** | <0.01 | 257535 | **0.08** | 32 | 1836 | 257535 | **0.08** | **29** | **1791** |
| Average | | <0.01 | | **0.06** | 35.78 | 863.78 | | 0.07 | **34.33** | **841.94** |

rithm, since the number of iterations was smaller in almost all instances when compared to the version without these variables. Also, the number of columns generated was, on average, smaller when adding these variables to the master problem.

Table 6 shows the results found by the heuristic approaches, within a time limit of 10 seconds, in the first group of instances. We can observe that the RH obtained the best UB in 10 of the 18 test instances with an average execution time of 4.75 seconds compared to 9.73 seconds of the RMIPH.

Table 6: Results obtained by the proposed heuristics in the first group of instances

| Instance | Best LB | RMIPH (MP$_{ex}$) | | | RH | | |
|---|---|---|---|---|---|---|---|
| | | UB | Time | Gap (%) | UB | Time | Gap (%) |
| ct-1-4-1000-a1 | 8748 | **8765** | 10.00 | 0.19 | 8828 | 6.58 | 0.91 |
| ct-1-4-1000-a2 | 8237 | 8331 | 10.00 | 1.14 | **8262** | 6.33 | 0.30 |
| ct-1-4-1000-b1 | 10105 | 10209 | 10.00 | 1.03 | **10139** | 2.73 | 0.34 |
| ct-1-4-1000-b2 | 10128 | **10215** | 10.00 | 0.86 | 10224 | 4.09 | 0.95 |
| ct-1-4-1000-c1 | 12961 | 12996 | 10.00 | 0.27 | **12979** | 1.06 | 0.14 |
| ct-1-4-1000-c2 | 12890 | 12929 | 10.00 | 0.30 | **12911** | 1.33 | 0.16 |
| ct-1-4-2000-a1 | 17109 | 17326 | 10.00 | 1.27 | **17233** | 5.77 | 0.72 |
| ct-1-4-2000-a2 | 17191 | **17203** | 5.12 | 0.07 | 17250 | 12.12 | 0.34 |
| ct-1-4-2000-b1 | 20541 | 20695 | 10.00 | 0.75 | **20588** | 4.50 | 0.23 |
| ct-1-4-2000-b2 | 20299 | **20403** | 10.00 | 0.51 | 20418 | 3.58 | 0.59 |
| ct-1-4-2000-c1 | 30582 | **30598** | 10.00 | 0.05 | 30607 | 1.35 | 0.08 |
| ct-1-4-2000-c2 | 30896 | **30920** | 10.00 | 0.08 | 30951 | 1.76 | 0.18 |
| ct-1-4-4000-a1 | 33913 | 34254 | 10.00 | 1.01 | **33965** | 9.66 | 0.15 |
| ct-1-4-4000-a2 | 34364 | 34614 | 10.00 | 0.73 | **34510** | 9.75 | 0.42 |
| ct-1-4-4000-b1 | 40946 | 41104 | 10.00 | 0.39 | **41065** | 5.09 | 0.29 |
| ct-1-4-4000-b2 | 40597 | 40799 | 10.00 | 0.50 | **40735** | 7.53 | 0.34 |
| ct-1-4-4000-c1 | 51809 | **51848** | 10.00 | 0.08 | 51889 | 1.08 | 0.15 |
| ct-1-4-4000-c2 | 51400 | **51426** | 10.00 | 0.05 | 51455 | 1.23 | 0.11 |
| Média | | | 9.73 | 0.51 | | **4.75** | **0.36** |

Table 7 presents the results obtained by the developed heuristics, also within a time limit of 10 seconds, for the second group of instances. The RH was found capable of producing high quality solutions faster than RMIPH in terms of average computational time.

Table 7: Results obtained by the proposed heuristics in the second group of instances

| | | RMIPH ($MP_{ex}$) | | | RH | | |
|---|---|---|---|---|---|---|---|
| Best LB | Best LB | UB | Time | Gap (%) | UB | Time | Gap (%) |
| ct-2-4-1000-a1 | 43827 | 43979 | 10.00 | 0.35 | **43905** | 1.11 | 0.18 |
| ct-2-4-1000-a2 | 41272 | 41357 | 10.00 | 0.21 | **41310** | 1.17 | 0.09 |
| ct-2-4-1000-b1 | 50645 | 50687 | 10.00 | 0.08 | **50682** | 0.92 | 0.07 |
| ct-2-4-1000-b2 | 50743 | **50759** | 10.00 | 0.03 | 50826 | 0.67 | 0.16 |
| ct-2-4-1000-c1 | 64926 | **64926** | 0.15 | 0.00 | 64938 | 0.23 | 0.02 |
| ct-2-4-1000-c2 | 64597 | **64597** | 0.09 | 0.00 | **64597** | 0.24 | 0.00 |
| ct-2-4-2000-a1 | 85735 | 85874 | 10.00 | 0.16 | **85760** | 1.23 | 0.03 |
| ct-2-4-2000-a2 | 86157 | 86276 | 10.00 | 0.14 | **86250** | 0.93 | 0.11 |
| ct-2-4-2000-b1 | 102921 | **102927** | 10.00 | 0.01 | 102929 | 0.62 | 0.01 |
| ct-2-4-2000-b2 | 101720 | **101739** | 10.00 | 0.02 | 101741 | 0.64 | 0.02 |
| ct-2-4-2000-c1 | 154519 | **154533** | 0.09 | 0.01 | 154582 | 0.28 | 0.04 |
| ct-2-4-2000-c2 | 156179 | **156192** | 0.10 | 0.01 | 156198 | 0.24 | 0.01 |
| ct-2-4-4000-a1 | 169976 | 170207 | 10.00 | 0.14 | **170079** | 1.07 | 0.06 |
| ct-2-4-4000-a2 | 172240 | **172403** | 10.00 | 0.09 | 172436 | 1.06 | 0.11 |
| ct-2-4-4000-b1 | 205212 | **205245** | 10.00 | 0.02 | 205325 | 0.69 | 0.06 |
| ct-2-4-4000-b2 | 203468 | **203475** | 4.16 | <0.01 | 203477 | 0.73 | <0.01 |
| ct-2-4-4000-c1 | 259590 | **259603** | 0.16 | 0.01 | 259672 | 0.26 | 0.03 |
| ct-2-4-4000-c2 | 257546 | 257559 | 0.11 | 0.01 | **257555** | 0.22 | <0.01 |
| Average | | | 6.38 | 0.07 | | **0.68** | **0.06** |

By observing the results of Tables 6 and 7, we argue that the proposed solution approaches are highly effective in terms of solution quality and computational time. These results play an important role with regard to (1) the high quality of the solutions obtained that allows for great power minimization, helping the development of energy-efficient virtualized server clusters, and (2) the scalability of our optimization approach, which can be applied to large-scale clusters supporting and consolidating thousands of services.

A promising way to improve even more the effectiveness of our solution approach is to take advantage of the multi-thread architectures, where each thread could run concurrently different versions of our algorithms within a given time limit. We could then choose, for example, the best solution obtained among all execution threads.

With these encouraging results obtained, we would like to analyze the behavior of the optimization approach based on real measurements of a real cluster test-bed. As a case study, we can implement our proposed optimization for a typical web cluster system, and evaluate the solution by running simulations to measure energy savings while meeting service demands driven by actual time-varying workloads, such as WorldCup'98 web traces available in [2].

## 5 Conclusions and Future Work

We have described a novel solution approach, based on column generation techniques, for power-aware optimization in virtualized server clusters. Given the computational results obtained, the proposed approach was found effective in providing high quality solutions in short amount of processing time, considering instances generated for large-scale heterogeneous server clusters.

In the future work, we plan to extend our approach to handle other types of demands for the services in the virtualized cluster, which would include a new dimension to the optimization problem. For example, in the case of specific I/O-intensive services, it would be interesting to make run time decisions on the best allocation of storage or RAM demands, which could also vary over time. Also, there is a trend of improving energy efficiency in multi-core server architectures by focusing on fine-grain power management [33] and dynamic voltage and frequency scaling (and on/off mechanisms) at core level [8]. This would allow for interesting configuration possibilities to consider in the power-aware optimization problem.

## References

[1] C. Alves and J. M. Valério de Carvalho. A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, 35(4):1315–1328, 2008.

[2] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical report, IEEE Network, 1999.

[3] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[4] G. Belov and G. Scheithauer. A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, 141(2):274–294, 2002.

[5] L. Bertini, J. Leite, and D. Mossé. Statistical qos guarantee and energy-efficiency in web server clusters. *Real-Time Systems, Euromicro Conference on*, pages 83–92, 2007.

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

[7] M. Bichler, T. Setzer, and B. Speitkamp. Capacity Planning for Virtualized Servers. *Workshop on Information Technologies and Systems (WITS), Milwaukee, Wisconsin, USA*, 2006.

[8] W. L. Bircher and L. K. John. Analysis of dynamic power management on multi-core processors. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 327–338, New York, NY, USA, 2008. ACM.

[9] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. *ACM Comput. Surv.*, 34(2):263–311, 2002.

[10] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. *SIGOPS Oper. Syst. Rev.*, 35(5):103–116, 2001.

[11] K. Church, A. Greenberg, and J. Hamilton. On delivering embarrassingly distributed cloud services. In *HotNets*, 2008.

[12] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[13] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 164–177. New York, 2003.

[14] Electronic Educational Devices. Watts Up PRO. http://www.wattsupmeters.com/, 2010.

[15] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Power-Aware Computer Systems*, volume 2325 of *Lecture Notes in Computer Science*, pages 179–197, 2003.

[16] X. Fan, W.-D. Weber, and L. A. 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.

[17] D. Friesen and M. Langston. Variable sized bin packing. *SIAM journal on computing*, 15:222, 1986.

[18] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 157–168, New York, NY, USA, 2009. ACM.

[19] E. L. Haletky. *VMware ESX Server in the Enterprise: Planning and Securing Virtualization Servers.* 2008.

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

[21] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr, and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 186–195. ACM, 2005.

[22] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers*, pages 444–458, 2007.

[23] N. Kandasamy, S. Abdelwahed, and J. P. Hayes. Self-optimization in computer systems via on-line control: Application to power management. In *ICAC '04*, pages 54–61, 2004.

[24] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. *10th IEEE/IFIP Network Operations and Management Symposium*, pages 373–381, 2006.

[25] B. Khargharia, S. Hariri, and M. S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, 2008.

[26] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

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

[28] D. Mosberger and T. Jin. httperf – a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, 1998.

[29] V. Petrucci, E. V. Carrera, O. Loques, J. Leite, and D. Mossé. Optimized management of power and performance for virtualized heterogeneous server clusters. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid (CCGrid'11)*, 2011.

[30] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'01)*, Sept. 2001.

[31] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

[32] D. Pisinger. A minimal algorithm for the bounded knapsack problem. *INFORMS Journal on Computing*, 12(1):75, 2000.

[33] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: fine-grained power management for multi-core systems. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 302–313, New York, NY, USA, 2009. ACM.

[34] P. Ranganathan. Recipe for efficiency: principles of power-aware computing. *Commun. ACM*, 53(4):60–67, 2010.

[35] C. Rusu, A. Ferreira, C. Scordino, A. Watson, and D. Mossé. Energy-efficient real-time heterogeneous server clusters. In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 418–428. IEEE, 2006.

[36] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *In Proceedings of the 24th IEEE Real-Time systems Symposium (RTSSâ03)*, pages 63–72, 2003.

[37] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 10–10, Berkeley, CA, USA, 2008. USENIX Association.

[38] J. M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17(2):175, 2005.

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

[40] Y. Wang, X. Wang, M. Chen, and X. Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *Proceedings of the 2008 Real-Time Systems Symposium*, pages 303–312, Washington, DC, USA, 2008. IEEE Computer Society.