2016 IEEE 9th International Conference on Cloud Computing

Energy Optimal VM Placement in the Cloud

Yi Wang and Ye Xia
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL, USA. {yiwan, yx1}@cise.ufl.edu

***Abstract*—In this paper, we investigate the issue of minimizing data center energy usage. In particular, we formulate a problem of virtual machine placement with the objective of minimizing the total power consumption of all the servers. To do this, we examine a CPU power consumption model and then incorporate the model into an mixed integer programming formulation. In order to find optimal or near-optimal solutions fast, we resolve two difficulties: non-linearity of the power model and integer decision variables. We first show how to linearize the problem, and then give a relaxation and iterative rounding algorithm. Computation experiments have shown that the algorithm can solve the problem much faster than the standard integer programming algorithms, and it consistently yields near-optimal solutions. We also provide a heuristic min-cost algorithm, which finds less optimal solutions but works even faster.**

***Index Terms*—Cloud Computing, Data Center, Energy Consumption, Resource Management, Mixed Integer Programming**

I. Introduction

Cloud computing gives organizations the flexibility to leverage a vast amount of computing resources and handle datasets on the peta-byte scale. As more companies and users move to the cloud, the data centers are facing increasing workload and diverse jobs, which put pressure on the resource management component of cloud systems. The current systems are becoming inadequate for the growing demands. Large data clusters in Yahoo!, LinkedIn and Amazon are already experiencing performance degradation, with syndromes such as occasional long job latency or violated client service level agreement. Inappropriate resource management also results in wasted cloud resources and excessive power consumption, thus significantly reducing the return on investment.

Though highly desired, a smart and effective cloud resource management component is hard to design. One of the main reasons is that there are many different and complex constraints and optimization objectives to take into account, which means there are many different and difficult resource management problems to solve. This leads to two challenges: It is difficult to capture the resource management problems comprehensively and accurately in usable formulations, and it is difficult to find high-quality solutions. To come up with a high-performance, widely applicable method for cloud resource management, we resort to mixed integer programming (MIP) as the foundation [1]. MIP formulations are capable of describing a broad range of resource management problems, as is well known in related fields involving complex resource management or scheduling [2], [3]. Together with a large repository of sophisticated MIP algorithms, the MIP framework gives us powerful tools to manage the cloud resources optimally. Achieving optimality or near-optimality is important in this context, since even a small percentage of improvement may translate into significant monetary value and competitive advantage in the market place.

In this paper, we focus on an energy-optimal virtual machine (VM) placement problem, with the objective of minimizing the total power consumption of all the physical machines (PM). We will formulate it as an MIP problem, derive algorithms to solve it and analyze their performance. There are reasons for highlighting energy efficiency. The energy cost and environmental impact of large data centers have caused significant concerns. According to NRDC [4], data centers are one of the largest and fastest growing electricity consumers in the United States. In 2013, U.S. data centers consumed an estimated 91 billion kilowatt-hours of electricity, resulting in 9 billion dollars in electricity bills and 97 million metric tons of CO_2 emission. The electricity consumption is expected to reach 140 billion kilowatt-hours by 2020. Therefore, optimizing data center energy usage is increasingly important.

The energy-minimizing VM placement problem is technically difficult. In the paper, we will present important techniques to address the difficulties. In addition to having a large number of integer decision variables, the problem involves non-linear relations. Non-linearity makes it difficult to find good heuristic algorithms, since heuristics are based on intuitions and intuitions are usually not accurate enough to reflect the subtle relationship characterized by non-linear functions. In such a situation, general MIP techniques are indispensable for either finding a complete optimal solution or providing a basis to find near-optimal solutions. Proper linearization is usually the first step. We will show how to suitably convert our non-linear problem into a linear one. However, even for linear MIP problems, standard algorithms can take a long time to find optimal solutions. We will provide a relaxation and iterative rounding algorithm, and demonstrate that it can solve fairly large instances of our problem quickly. Importantly, the obtained solutions are nearly optimal.

This work is relatively unique in that it develops MIP algorithms for solving large-scale VM placement problems under a non-linear power consumption model. The most related prior research is [5]. However, that work tackles a different energy-minimizing VM placement problem, namely, the minimization of data center network energy (as opposed to PM energy). Moreover, the energy model in [5] is linear in the traffic load (as apposed to non-linear). That problem has a challenge of a different kind: It is a quadratic assignment problem, which is known to be very difficult. Our main challenge is the non-linearity. Finally, although it starts with an MIP formulation for its problem, the final algorithm in [5] deviates significantly from MIP algorithms, whereas our main algorithm stays within the class of MIP heuristics.

We next summarize the contributions of this paper.

1) We investigate an important energy-minimizing VM placement problem, with the objective of minimizing the total power consumption of the PMs under a non-linear power consumption model. The non-linearity makes the problem distinct from prior studies.
2) We develop an MIP-based algorithm capable of solving large instances of the problem. Technically, we show how to cope with two main challenges: non-linear constraints or objectives, and integer decision variables. We introduce effective techniques to reduce the computation time, including linearization, relaxation and iterative rounding. These techniques may also be useful for solving other cloud resource management problems.
3) We synthesize the non-linear power consumption model based on a literature survey, and then incorporate that model into an MIP formulation. The model and problem formulation can be useful to the research community for further study of the subject.

We next briefly review other relevant work. VM placement and related cloud resource management problems have been studied by many, under different circumstances and assumptions. Prior studies generally avoid MIP formulations all together. In the cases where MIP formulations are used, the algorithms are usually not MIP algorithms; instead, more specialized combinatorial algorithms are developed, such as multi-dimensional bin-packing [6], [7], graph algorithms [8] or sophisticated heuristics [9], which are only applicable to special problems with the structures required by those algorithms. For many other resource management problems with far more complex constraints, often unknown ahead of the time, those specialized algorithms will not directly apply.

The work in [10]–[13] investigates energy-aware resource management for cloud computing. Bin-packing related algorithms are presented in [10] to schedule each server's workload around a pre-determined optimal efficiency point. Experimental results about cluster energy consumption under different workload and job types are reported in [11], as well as simple ways to allocate a fixed power budget among servers. Best-fit-decreasing heuristics with multiple thresholds for VM migration are proposed in [12] to assign VMs in a cluster. In these studies, the problems and power consumption models are quite different from ours. The proposed algorithms are not of the MIP type, but are various heuristics. They usually cannot achieve as high performance as MIP-based algorithms, which seek optimal solutions (see experimental comparison with [10] in Section IV). Furthermore, they are usually less adaptable to changes of the problem, such as the addition of extra constraints. Our work is based on the MIP framework, which can address those limitations and provide general methods for solving a wide range of resource management problems.

The rest of this paper is organized as follows. In Section II, we formulate the energy-optimal VM placement problem. In Section III, we describe our techniques to reduce the computation time. A heuristic algorithm is also presented. Section IV reports the experimental results and analysis. We conclude the paper in Section V.

II. Model and Problem Formulation

In this section we describe the energy-minimizing VM placement problem. A highlight is the incorporation of a non-linear power consumption model. We will develop an MIP formulation of the problem.

A. Usual Resource Constraints

We consider a problem of assigning N VMs to M PMs in a data center. Each PM has certain resource capacity, including computing power in number of vCPUs, memory size, local storage such as hard disks or SSDs, and other resources such as network I/O speed. Correspondingly, each VM requested by a user has certain resource requirement, including the number of vCPUs, memory size, storage volumes, and possibly network or disk I/O performance guarantee. In any valid VM-to-PM assignment, the capacity constraint must be satisfied with respect to each resource of the PM. That is, for each resource, the total amount requested by all the VMs assigned to a PM cannot exceed that PM's capacity of the resource.

Our problem formulation will include the vCPU and memory constraints, but will omit the constraints posed by the local disk capacity or by other resources. If needed, those constraints can be added to the formulation. Part of the reason for not considering the local disk constraints is that, in many cloud systems, the VMs' storage space is often allocated on network storage clusters and the storage space there is usually abundant. More detailed consideration of the storage options and their possible constraints is left to future research.

B. Overview of Data Center Energy Consumption

The optimization objective of our problem is to minimize the energy usage of the data center. Before getting into the problem formulation, we will discuss a data center power consumption model, which captures how the power consumption depends on the workload on each PM.

In a data center, the total energy usage consists of the energy consumption of the servers (PMs), network equipments, cooling system and other auxiliary systems. We consider only the server energy cost in our problem formulation as it is a major contributor to the total energy cost. The energy usage of the cooling system is directly related to the server power consumption, as it deals with the generated heat. Controlling the network energy usage is also important. But, what can be controlled there is quite different from how the server energy usage can be managed. That problem is left to future work.

C. Power Consumption Model of PMs

Thus, our objective is to minimize the total power consumption of all the PMs in a data center. For each PM, its energy usage consists of several contributors: CPU, memory, storage disks, power and cooling system, and peripheral equipments. For simplicity and usefulness, we focus on the energy usage of the CPU and memory because they consume a large portion of the total energy. In addition, they are among the main resources to be managed for VM placement problems. Their energy consumption depends on the PM's workload, as we will discuss more in the following.

Most modern CPUs have the ability to dynamically adjust its working voltage and frequency (DVFS) according to the workload in order to conserve energy. The power consumption, denoted by P, of a CPU is an increasing function of its operating frequency. In [11], [14], the authors describe the relationship as a cubical power function:

$$P(f) = \hat{\alpha} + \hat{\beta} f^3. \tag{1}$$

Here, f is the operating frequency of the CPU, which can be dynamically adjusted from $f_{\min}$ to $f_{\max}$ in several discrete steps; $\hat{\alpha}$ is the static power consumption of the CPU; and $\hat{\beta}$ is a constant coefficient. The range of the allowed frequencies varies for different CPU types. When a CPU is running at $f_{\max}$, it reaches its Thermal Design Power (TDP), which is the maximum amount of heat generated by the CPU. When the frequency range of a CPU is large, it can potentially run at a much lower frequency and use much less power than TDP.

However, a lower CPU frequency or lower power level comes at the cost of the CPU's computing capability. At a lower frequency, the execution time of a job is longer. [15] proposes the following model that relates the application execution time T to the CPU operating frequency f:

$$T(f) = T(f_{\max}) \cdot \big(\gamma(\frac{f_{\max}}{f} - 1) + 1\big), \tag{2}$$

where γ is a positive constant related to the CPU type, as well as other computing related resources, such as memory and disk space. As γ decreases, the CPU has more potential to reduce the frequency while maintaining certain computing capability. In the remaining part of the paper, we assume $\gamma = 1$, in which case the execution time is an inverse function of the frequency, i.e., $T(f) = T(f_{\max})\frac{f_{\max}}{f}$. The case of general γ is not more difficult conceptually, but clutters the notations.

Let $C(f)$ denote the computing capability of the CPU as a function of the frequency, measured by the number of identical jobs that can be completed in a unit time. By taking the inverse of $T(f)$, we get

$$C(f) = C(f_{\max})\frac{f}{f_{\max}}. \tag{3}$$

We see from (3) that the computing capability of a CPU is a linear function of the operating frequency. Thus, the CPU becomes more powerful and supports more vCPUs as the frequency increases; but it also consumes more power.

Let ρ be the normalized operating frequency defined by $\rho = f/f_{\max}$. It can be understood as the *utilization* of the CPU. Then, from (3), we can write $C(f) = \rho\, C(f_{\max})$. Let the maximum power consumption (which is equal to TDP) of the CPU be denoted by $P_{\max}$. We can rewrite the power function in (1) in terms of the CPU utilization ρ:

$$P(\rho) = (\alpha + \beta\rho^3)P_{\max}, \tag{4}$$

where $\alpha = \hat{\alpha}/P_{\max}$ and $\beta = \hat{\beta} f_{\max}^3/P_{\max}$ are constants. Note that we have $\alpha + \beta\rho^3 \le 1$ for $\rho \in [0, 1]$ and $\alpha + \beta = 1$.

The power consumption of the computer memory is mainly related to its working frequency and the number of data read/write transactions per unit time. Without detailed information of the latter, which depends on the workload type, we make the simplifying assumption that the memory has approximately constant power consumption, denoted by P_{mem}, no matter how much of its capacity is allocated[1]. Then, the sum of the CPU and memory power consumption is $\bar{P}(\rho) = (\alpha + \beta\rho^3)P_{\max} + P_{mem}$. This is for a generic PM. For PM k, we add subscript k to appropriate places:

$$\bar{P}_k(\rho) = (\alpha_k + \beta_k\rho^3)P_{k,\max} + P_{k,mem}. \tag{5}$$

If there is no VM assigned to a PM, we can turn off the PM so that it does not consume energy. But if there is at least one VM assigned to the PM, it needs to be turned on.

Our problem is to minimize the total power consumption of the PMs by placing the VMs intelligently. A naive thinking may be that the VMs should be consolidated as much as possible, leaving more PMs unused, which can be shut down. However, that may result in the active PMs being heavily loaded and consuming much more power. Precisely how much power is consumed by each PM depends on the assigned load in complicated ways due to the cubic relation and the constants in (5), and due to the fact that the constants and the variable range vary for different types of PMs. Thus, it is not easy to come up with a good VM placement strategy based on simple intuitions or heuristics alone. What is needed is, first, a precise problem formulation, and second, a high-quality solution. We next give an MIP formulation of the problem.

D. Problem Formulation

Let the sets of VMs and PMs be denoted by $\mathcal{V} = \{1, 2, \dots, N\}$ and $\mathcal{P} = \{1, 2, \dots, M\}$, respectively. For each VM i, let c_i be its required number of vCPUs and b_i be its memory size (GB). For each PM k, let C_k be the number of vCPUs it can support and B_k be its memory capacity (GB).

For each PM k, we use the binary variable y_k to represent the on/off status of the PM, with $y_k = 1$ indicating the PM is turned on and $y_k = 0$ indicating otherwise. For each VM i and each PM k, we use the binary variable x_{ik} to indicate whether VM i is assigned to PM k, with $x_{ik} = 1$ if VM i is assigned to PM k and $x_{ik} = 0$ otherwise.

For each PM k, if we view its CPU frequency f_k as a continuous variable, f_k can be dynamically adjusted within the range $[f_{k,\min}, f_{k,\max}]$. Accordingly, the CPU utilization ρ_k varies within $[\rho_{k,\min}, 1]$ where the minimum utilization $\rho_{k,\min} = f_{k,\min}/f_{k,\max}$. In reality, the CPU frequency can be adjusted in discrete steps. We let $\Omega_k \subseteq [\rho_{k,\min}, 1]$ be a finite set of CPU utilization values and ρ_k takes values in Ω_k.

Our optimization problem is as follows.

$$\min \sum_{k\in\mathcal{P}} \Big((\alpha_k + \beta_k\rho_k^3)P_{k,\max}\, y_k + P_{k,mem}\, y_k \Big) \tag{6}$$

$$\text{s.t.} \sum_{i\in\mathcal{V}} c_i x_{ik} \le \rho_k C_k,\ \forall k \in \mathcal{P} \tag{7}$$

$$\sum_{i\in\mathcal{V}} b_i x_{ik} \le B_k\, y_k,\ \forall k \in \mathcal{P} \tag{8}$$

$$\sum_{k\in\mathcal{P}} x_{ik} = 1,\ \forall i \in \mathcal{V} \tag{9}$$

$$\rho_k \in \Omega_k,\ \forall k \in \mathcal{P} \tag{10}$$

$$y_k, x_{ik} \in \{0, 1\},\ \forall i \in \mathcal{V}, \forall k \in \mathcal{P}. \tag{11}$$

[1]It may be more appropriate to model the memory power consumption as a linear function of the amount of memory actually allocated to the VMs. This linear model does not make the problem more difficult.

Inequality (7) is the CPU constraint, which ensures that the total number of requested vCPUs by all the VMs assigned to a PM shall not exceed that PM's chosen computing capability measured in the number of vCPUs. Inequality (8) is the memory size constraint for each PM. It also ensures that when $y_k = 0$, i.e., when the PM k is turned off, no VMs are assigned it (note that the required memory size b_i of any VM i is positive); conversely, as long as $x_{ik} = 1$ for some i, y_k must be set to 1. Finally, (9) ensures that each VM should be assigned to exactly one PM.

III. Solving the Optimization Problem

Our optimization problem presents two major challenges that make it difficult to solve. The first is the non-linear objective function, and the second is the integer decision variables. This section describes techniques and introduces algorithms for solving the problem efficiently.

A. Linearization

Our problem is non-linear since it has a non-linear objective function. Aside from enumeration, there are no general algorithms that solve integer problems with non-linear objective functions or constraints. Our first step towards solving the problem is to linearize it, which can then be solved by applying linear programming techniques.

To linearize the problem, we first use the following piece-wise linear function $h_k(\rho)$ to approximate the non-linear function $\alpha_k + \beta_k \rho^3$ for each $k \in \mathcal{P}$ (see Fig. 1).

$$h_k(\rho) = \begin{cases} \alpha_k + \beta_k \frac{1}{4}\rho, & \text{for } \rho_{\min} \le \rho \le \frac{1}{2} \\ \alpha_k + \beta_k(\frac{19}{16}\rho - \frac{15}{32}), & \text{for } \frac{1}{2} < \rho \le \frac{3}{4} \\ \alpha_k + \beta_k(\frac{37}{16}\rho - \frac{21}{16}), & \text{for } \frac{3}{4} < \rho \le 1. \end{cases} \tag{12}$$

It is easy to see that $h_k(\rho)$ is continuous on $[0, 1]$ and it can be written as

$$\begin{aligned} h_k(\rho) = \max\{&\alpha_k + \beta_k \frac{1}{4}\rho,\ \alpha_k + \beta_k(\frac{19}{16}\rho - \frac{15}{32}), \\ &\alpha_k + \beta_k(\frac{37}{16}\rho - \frac{21}{16})\}. \end{aligned} \tag{13}$$

For each $k \in \mathcal{P}$, we introduce a variable $t_k = h_k(\rho)$, where $\alpha_k \le t_k \le 1$. It represents the normalized CPU power level chosen for PM k relative to the maximum power $P_{k,\max}$. The optimization objective can be written as

$$\min \sum_{k \in \mathcal{P}} \big(t_k P_{k,\max}\, y_k + P_{k,mem}\, y_k\big). \tag{14}$$

For each $k \in \mathcal{P}$, we can replace $t_k = h_k(\rho)$ by $t_k \ge h_k(\rho)$ without affecting the optimal solutions. Due to (13), the latter in turn is the same as the three constraints: (19), (20) and (21).

We still have quadratic terms such as $t_k\, y_k$. For each $k \in \mathcal{P}$, let us introduce a variable $z_k = t_k\, y_k$. The range of z_k is $0 \le z_k \le 1$. We replace $t_k y_k$ in the objective function in (14) by z_k. The optimization objective now becomes $\min \sum_{k\in\mathcal{P}} \big(z_k P_{k,\max} + P_{k,mem}\, y_k\big)$. However, we do not wish to explicitly add the constraint $z_k = t_k y_k$ since it is non-linear. Instead, we add the following four linear constraints $z_k \le y_k$, $z_k \le t_k$, $z_k \ge t_k + y_k - 1$ and $z_k \ge 0$, which are equivalent to $z_k = t_k y_k$. To see the equivalence, $z_k \le y_k$ and $z_k \ge 0$ together ensure that $0 \le z_k \le 1$ and that $z_k = 0$ when $y_k = 0$. When $y_k = 1$, $z_k \ge t_k + y_k - 1$ is the same as $z_k \ge t_k$; together with $z_k \le t_k$, we have $z_k = t_k$. Thus, these four constraints imply $z_k = t_k y_k$. It is also easy to see that $z_k = t_k y_k$ together with $0 \le t_k \le 1$ implies the four linear constraints.

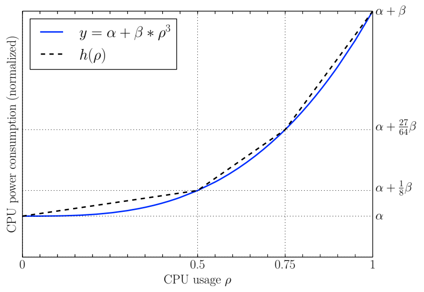

Fig. 1. Comparison of $\alpha + \beta\rho^3$ with the piece-wise linear approximation $h(\rho)$; $h(\rho)$ is as in (12) with the subscript k removed; $\alpha = 0.25$, $\beta = 0.75$.

Putting things together, we write down the linearized formulation of the problem.

$$\min \sum_{k \in \mathcal{P}} \big(z_k P_{k,\max} + P_{k,mem}\, y_k\big) \tag{15}$$

$$\text{s.t.} \sum_{i \in \mathcal{V}} c_i x_{ik} \le \rho_k C_k,\ \forall k \in \mathcal{P} \tag{16}$$

$$\sum_{i \in \mathcal{V}} b_i x_{ik} \le B_k\, y_k,\ \forall k \in \mathcal{P} \tag{17}$$

$$\sum_{k \in \mathcal{P}} x_{ik} = 1,\ \forall i \in \mathcal{V} \tag{18}$$

$$\alpha_k + \beta_k \frac{1}{4}\rho_k \le t_k,\ \forall k \in \mathcal{P} \tag{19}$$

$$\alpha_k + \beta_k(\frac{19}{16}\rho_k - \frac{15}{32}) \le t_k,\ \forall k \in \mathcal{P} \tag{20}$$

$$\alpha_k + \beta_k(\frac{37}{16}\rho_k - \frac{21}{16}) \le t_k,\ \forall k \in \mathcal{P} \tag{21}$$

$$z_k \le y_k,\ \forall k \in \mathcal{P} \tag{22}$$

$$z_k \le t_k,\ \forall k \in \mathcal{P} \tag{23}$$

$$z_k \ge t_k + y_k - 1,\ \forall k \in \mathcal{P} \tag{24}$$

$$z_k,\ t_k \in [0, 1],\ \rho_k \in \Omega_k,\ \forall k \in \mathcal{P} \tag{25}$$

$$y_k,\ x_{ik} \in \{0, 1\},\ \forall i \in \mathcal{V}, \forall k \in \mathcal{P}. \tag{26}$$

The decision variables are x_{ik}, y_k, z_k, t_k and ρ_k.

B. Relaxation and Iterative Rounding

After linearization, we can use linear and integer optimization software, such as CPLEX and Gurobi, to solve fairly large problems. We are able to solve problems with up to 300 – 500 VMs and PMs in a reasonable amount time (see Section IV). However, when the problem size grows larger, the computation time increases dramatically. To solve even larger problems, we next describe a particular relaxation and rounding technique.

Our main algorithm (Algorithm 1) is named relaxation and iterative rounding, or *iterative rounding* for short. The idea is to relax some integer decision variables to continuous variables in order to accelerate computation, and after solving the new problem, we round the relaxed variables back to integer values.

In Algorithm 1, we relax all the x_{ik} variables such that each takes values on $[0, 1]$. However, we keep the y_k variables as binary variables and the ρ_k variables as discrete variables taking values in Ω_k. The reason for doing so is somewhat subtle. Earlier, we discussed how to replace the non-linear constraint $z_k = t_k y_k$ by four linear constraints $z_k \leq y_k$, $z_k \leq t_k$, $z_k \geq t_k + y_k - 1$ and $z_k \geq 0$. The two descriptions are equivalent under the assumption that the y_k variables take values of 0 or 1. Should we relax the y_k variables, the four linear constraints would have become substantially different from the intended constraint $z_k = t_k y_k$. That would have implications on the achievable objective value by the relaxed algorithm. Nevertheless, it is a valid option to relax *all* the integer variables, including the y_k variables. The relaxed problem then becomes a linear programming problem, which can solved much faster than the MIP counterpart[2].

After relaxing the x_{ik} variables, we still have a mixed integer problem with y_k and ρ_k as the integer decision variables. However, the number of integer variables has been substantially reduced, and the resulting problem can be solved much faster. The values of x_{ik} in the solution are mostly fractional numbers on $[0, 1]$. Recall that each x_{ik} is an assignment variable about whether to assign VM i to PM k. A fractional assignment does not make practical sense.

To get a solution for the original binary assignment problem, we constructed a rounding scheme to convert the fractional numbers to binary numbers. A typical rounding method may use a certain threshold to determine whether a fractional number should be rounded to 0 or 1. However, in optimization problems with complex constraints such as ours, this simple rounding method almost always results in infeasible solutions with respect to the constraints of the problem.

To deal with this issue, we designed a new iterative rounding scheme, which gives not only feasible but near-optimal solutions most of the time. It rounds the fractional numbers in iterations, instead of rounding them all at once. During each iteration, we use the software solver to solve the (relaxed) MIP problem. Among the fractional assignment numbers in the solution, we attempt to round the top $\theta\%$ of the largest numbers to 1. Let this set of top-$\theta\%$ largest numbers be denoted by X_θ. For each $x_{ik} \in X_\theta$, we check the feasibility of rounding it to 1 by verifying whether PM k has sufficient physical resources remaining. If rounding is feasible, then we accept the rounding and set $x_{ik} = 1$ permanently; that is, we consider the assignment of VM i to PM k to be final. At this moment, we can set $x_{ik'} = 0$ permanently for all PM $k' \neq k$, since VM i can only be assigned to one PM.

When we go to the next iteration and solve the next (relaxed) MIP problem, all the assignment variables that have been permanently assigned a value 1 or 0 will remain fixed. The rest of the variables are the decision variables for the new problem, again with the yet-to-be-determined x_{ik} treated as continuous variables and y_k treated as binary variables. The process repeats until all VMs have been assigned, or when an integer solution is unattainable, or when the number of iterations exceeds a pre-defined limit, denoted by LIMIT.

[2]Linear programming can be efficiently solved by interior point methods (polynomial complexity) or simplex method, etc.

Algorithm 1 Relaxation and Iterative Rounding

```
1: $R \leftarrow \{(i,k) : \forall i, \forall k\}$;            ▷ the set of remaining $(i,k)$
2: $F \leftarrow \emptyset$;                                  ▷ the set of assigned VMs
3: $iterations \leftarrow 0$;
4: while $R \neq \emptyset$ and $iterations \leq$ LIMIT do
5:     solve relaxed problem with $x_{ik}$ fixed for $(i,k) \in R$;
6:     if the solution status is infeasible then
7:         exit;
8:     end if
9:     $LF \leftarrow \emptyset$;                    ▷ the set of the largest fractions
10:    for each VM $i \notin F$ do
11:        add $\max_{k \in \mathcal{P}} x_{ik}$ to $LF$;
12:    end for
13:    sort the set $LF$ in decreasing order of $x_{ik}$;
14:    for each $x_{ik}$ in top $\theta\,\%$ of $LF$ do
15:        if $x_{ik} = 1$ is feasible then
16:            $x_{ik} \leftarrow 1$; add $i$ to $F$; delete $(i,k)$ from $R$;
17:            for each $k' \neq k$ do
18:                $x_{ik} \leftarrow 0$; delete $(i,k')$ from $R$;
19:            end for
20:            $y_k \leftarrow 1$;                        ▷ PM $k$ is turned on
21:        else
22:            $x_{ik} \leftarrow 0$; delete $(i,k)$ from $R$;
23:        end if
24:    end for
25:    $iterations \leftarrow iterations + 1$;
26: end while
```

In line 11 of Algorithm 1, we add the largest fractional assignment number x_{ik} for each VM i to the set LF. The reason is that if we want to round one of the fractional assignment numbers for a VM to 1, the natural choice is the largest one. Then, we choose the largest $\theta\%$ of the numbers in LF and attempt to round them to 1.

In line 15, when trying to round x_{ik} to 1, we first check whether this is feasible, i.e., whether there are enough remaining physical resources at PM k to accept VM i. If the feasibility check is passed, we set x_{ik} to 1. At the same time, $x_{ik'}$ is set to 0 permanently for all other $k' \neq k$ (line 18). At this point, the variable y_k is also permanently set to 1, indicating that PM k should be turned on (line 20). On the other hand, if rounding x_{ik} to 1 violates some resource constraints at PM k, we set $x_{ik} = 0$ permanently (line 22), so that it will not be re-considered in future iterations. This is the correct thing to do, since PM k's remaining resources can only decrease over the iterations.

C. Heuristic Algorithms

The iterative rounding algorithm shown in Section III-B is able to solve problems with up to several thousands of VMs and PMs. But, sometimes, there is a need to solve even larger problems. We next discuss two faster heuristic algorithms.

1) Min-Cost Algorithm: We developed the following simple greedy algorithm called the *min-cost algorithm*. The VMs are placed in sequence. For each VM i, the algorithm examines all the PMs and pretends to assign VM i to each PM k. If the assignment to PM k is feasible, i.e., it does not violate any resource capacity constraint, then the algorithm calculates the cost increase (i.e., power consumption increase) due to this assignment. After all the PMs are checked, VM i is permanently assigned to the PM that leads to the minimum cost increase. The algorithm iterates through all the VMs and all the PMs. Hence, the time complexity for N VMs and M PMs is $O(MN)$.

2) Optimal-Point Method: The related work [10] and [11] introduces a heuristic method, which tries to keep the PMs running at their energy-optimal utilization points. The optimal utilization point for each PM is obtained through numerical experiments, data profiling or analyzing the power consumption model. Then, for each VM assignment, the method checks all the PMs. It computes the distance between a PM's new utilization point, if the VM is assigned to it, and the PM's energy-optimal point. The VM is assigned to the PM with the smallest distance between the two points. This method has $O(MN)$ time complexity.

In Appendix A, we calculate the energy-optimal utilization point to be $\rho^* = 0.55$ under our problem formulation and experimental setup.

IV. Performance Evaluation and Analysis

In this section, we evaluate and analyze the performance of the algorithms discussed in Section III. We focus on the objective value and runtime as the performance metrics.

A. Experimental Setup

We use the VM types and PM types (Table I) from Amazon Web Service EC2 [16]. We do not have accurate information on the maximum power consumption $P_{k,\max}$ for different types of PMs. The 'Power' values in Table I are our estimations and are normalized with the least powerful PM type $s1$ set to 100. For better clarity in the evaluation, we assume $P_{k,mem} = 0$ for all $k \in \mathcal{P}$, since the memory part of the power consumption does not add complexity to the problem.

We experimented with 12 different mixes of VMs and PMs from the listed types to simulate different working environments in data centers. Each mix contains most of the VM and PM types but with different proportions. In the experiments, we set $\alpha_k = 0.25$ and $\beta_k = 0.75$ for all $k \in \mathcal{P}$ in the power function (5). Thus, the static part of the PM's power consumption is $1/4$ of its maximum power consumption, while the dynamic part can swing over a range that is equal to $3/4$ of the maximum power. We also set the $f_{k,\min} = 0.25 f_{k,\max}$ for each PM k, which implies the CPU utilization $\rho_k \in \Omega_k \subseteq [0.25, 1]$. We let each Ω_k contain seven discrete values. Note that, in practice, α_k, β_k and the CPU frequency adjustment range are dependent on the type of individual PMs. If the information is available, these parameters can be set individually for different PMs.

We use Gurobi Optimizer 6.0 [17] as the MIP solver. The numerical experiments are executed on a Linux PC with an AMD quad-core 3.5 GHz CPU and 16 GB memory. We limit the computation time for each experiment to 7200 seconds.

B. Results and Analysis

The simulation results are shown in Fig. 2, 3 and 4. The horizontal axis represents the 12 experiments and is labeled with the numbers of VMs and PMs in each experiment, e.g, 100 VMs and 50 PMs in experiment 1. Fig. 2 shows the optimal objective values obtained by the MIP algorithm. Fig. 3 shows the runtime of each algorithm. Fig. 4 shows the normalized objective values of each algorithm against the baseline, which is the optimal objective value for each experiment. The last experiment is too large for the MIP algorithm to finish within our time limit. In that case, the result of iterative rounding algorithm is used as the baseline.

1) The MIP Algorithm and Optimal Solutions: The MIP algorithm refers using the Gurobi MIP solver to directly solve the linearized problem (15) - (26). The MIP algorithm is a useful target of comparison since it yields the optimal objective values but takes the longest time. We have several observations from the MIP results. First, both the number of VMs and the number of PMs affect the power consumption. It is clear that having more VMs increases the total workload and therefore the total power consumption. On the other hand, if the set of VMs is kept the same, having more PMs results in less total power consumption. This is understandable because having more PMs potentially lowers the CPU utilization of each active PM, thus reducing individual PM's power consumption. The fast increasing power function (5) in our model makes it favorable to operate more PMs at medium CPU utilization, as opposed to activating fewer PMs but operating them at high utilization. Experiments 7 and 8, which have the same VM set but different PM sets, have clearly demonstrated this observation. Such subtle trade-offs cannot be known for certain without either solving the problem explicitly or running actual experiments. The latter will be time-consuming and expensive. The usefulness of our problem formulation and the need to find good algorithms are clearly demonstrated on this point alone.

TABLE I
VM Types and PM Types

VM Type	vCPU	Memory (GB)	Storage (GB)	Cost
m3.medium	1	3.75	1 × 4	67
m3.large	2	7.5	1 × 32	133
m3.xlarge	4	15	2 × 40	266
m3.2xlarge	8	30	2 × 80	532
c3.large	2	3.75	2 × 16	105
c3.xlarge	4	7.5	2 × 40	210
c3.2xlarge	8	15	2 × 80	420
c3.4xlarge	16	30	2 × 160	840
c3.8xlarge	32	60	2 × 320	1680
r3.large	2	15.25	1 × 32	175
r3.xlarge	4	30.5	1 × 80	350
r3.2xlarge	8	61	1 × 160	700
r3.4xlarge	16	122	1 × 320	1400
r3.8xlarge	32	244	2 × 320	2800
i2.xlarge	4	30.5	1 × 800	853
i2.2xlarge	8	61	2 × 800	1705
i2.4xlarge	16	122	4 × 800	3410
i2.8xlarge	32	244	8 × 800	6820
PM Type	vCPU	Memory (GB)	Storage (GB)	Power
s1	8	16	1 × 256	100
s2	8	32	1 × 512	150
s3	8	64	2 × 512	200
s4	8	64	4 × 512	250
m1	16	32	2 × 512	300
m2	16	64	4 × 512	400
m3	16	128	4 × 1000	500
m4	16	256	8 × 1000	700
m5	16	256	16 × 512	700
l1	32	256	4 × 1000	800
l2	48	512	8 × 1000	1200
l3	64	1024	4 × 1000	1500
l4	80	2048	16 × 1600	2200
l5	120	4096	4 × 1000	2500
l6	120	4096	24 × 1600	3000

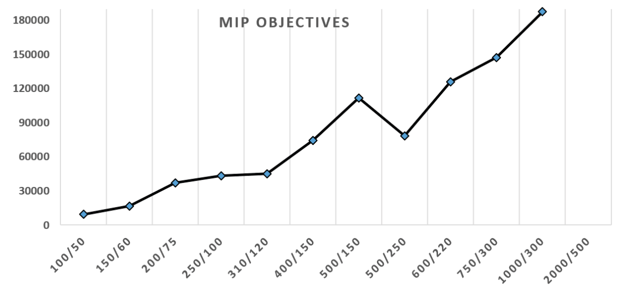

Fig. 2. Objective values of MIP algorithm in 12 experiments.

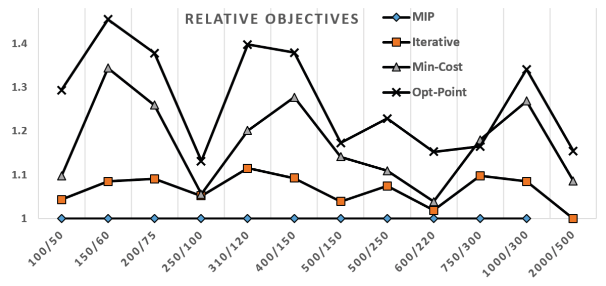

Fig. 4. Normalized objective values of all algorithms in 12 experiments.

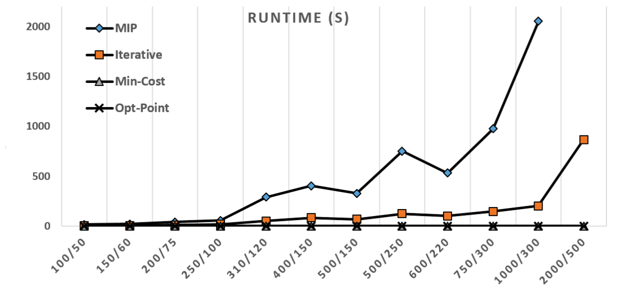

Fig. 3. Runtime of all algorithms in 12 experiments.

The second observation is that the runtime of the MIP algorithm increases rapidly when the problem size increases. This is expected due to the NP-hard nature of general MIP problems. The MIP algorithm failed to solve large problems within our time limit, such as experiment 12.

The number of PMs has a larger impact on the runtime than the number of VMs. This has to do with the fact that most of the constraints are related to the PMs. In general, the MIP algorithm does not scale very well for large data centers with more than thousands of PMs. When it is impractical to find an optimal solution, we need to resort to the other algorithms.

2) The Iterative Rounding Algorithm: The iterative rounding algorithm is much faster than the MIP algorithm. Even though the algorithm has to solve many relaxed problems over the iterations, it still takes much less time to finish. The runtime is on average 5.26 times shorter than the MIP algorithm. The speed gain becomes larger when the problem size increases, and it reaches 10.27 (times) in experiments 11. The iterative rounding algorithm can solve some of the experiments that are not solvable by the MIP algorithm. Thus, it has much better scalability than MIP.

Another desirable feasture of the iterative rounding algorithm is that it consistently returns near-optimal solutions. For the 11 experiments that the MIP algorithm is able to solve, the objective values obtained by the iterative rounding algorithm is only 7.22% worse (higher) on average than the optimal values. The worst case is 11.51% higher (experiment 5). With the iterative rounding algorithm, we are trading 1% to 11% loss in the objective values for 3 to 10 times or even larger gain in the computation speed. The algorithm is fairly robust against the choice of the parameter θ in Algorithm 1. For the results presented here, θ is set to 20. We have experimented with $\theta = 10$ to 30 and the results are generally similar. The iterative rounding algorithm can solve problems with 2000 VMs and 500 PMs. To scale to even larger problems, we suggest adding hierarchical decomposition (see Section V for discussion).

3) The Heuristic Algorithms: As expected, the two heuristic algorithms are much faster than the iterative rounding algorithm. The $O(MN)$ time complexity makes them easily scalable to thousands of PMs. However, their achieved objective values are worse than the iterative rounding algorithm. On average, our min-cost algorithm achieves 17.92% higher objective values than the optimal values. The worst case is 34.35% higher (experiment 2). The optimal-point method runs a bit faster than the min-cost algorithm, but achieves worse objective values. Its objective values are on average 28.12% higher than the optimal values; the worst case is 45.48% higher (experiment 2). Furthermore, the relatively large variations indicate that the performance of the heuristic algorithms is not as consistent as the iterative rounding algorithm. This is not surprising since the heuristic algorithms are designed based on simple intuitions that are unlikely to be correct in all situations.

In general, the two heuristic algorithms work very fast but find less optimal solutions. They can be used for large problem instances in near real-time, if the increase in the objective value is acceptable. However, heuristic algorithms have some notable drawbacks in general. First, they are tailored for each specific problem, and are usually not directly applicable to different but related problems, such as when extra constraints are added. When the problem changes, which can happen frequently in real applications, one has to find a different heuristic algorithm and verify its effectiveness. Second, their performance is not uniform across all problem instances, and it is hard to predict the quality of the solution.

C. Dynamic Assignment

Most of this paper concerns a static resource allocation problem where the sets of VMs and PMs are fixed. In practice, a data center usually encounters dynamics allocation problems, as new VM requests arrive and existing VMs are terminated or suspended. Our proposed methods can be extended to handle dynamic VM assignment. Since VMs usually persist for a relatively long time, ranging from hours to months, we can run the optimization process periodically. For example, re-optimization can be performed every hour or every day, using the most up-to-date VM and PM information. Each re-optimization may be restricted to small subsets of the VMs, such as those arrived after the previous optimization, plus a small sample of the earlier VMs, such as those causing severe PM overload or underload. The rest of the VMs that were assigned previously will not be re-assigned. Since only a subset of the VMs needs to be assigned, the re-optimization can be done quickly. To eliminate any delay in assignment, new VMs can be tentatively assigned to some lightly-loaded PMs. But, they are tagged to participate in the next re-optimization.

We have experimented with periodic re-optimization using the sets of VMs and PMs as in experiment 10. We simulate 10 re-optimization periods. During each period, 10% of the VMs are chosen randomly and removed from the system while the same number of new VMs are randomly chosen and added to the system. Due to the smaller problem size, the average runtime of the 10 simulation runs is only 55.28 seconds, which should be compared with 973.66 seconds for the initial optimization.

V. Conclusion and Discussion

In this paper, we investigate the issue of minimizing data center energy usage. In particular, we formulate a VM-to-PM placement problem with the objective of minimizing the total power consumption of the PMs. To find fast, near-optimal solutions, we have resolved two difficulties: non-linearity and integer decision variables. We introduce a linearization technique and the relaxation and iterative rounding algorithm to improve the computation time. Experiments have shown that the iterative rounding algorithm solves the problem much faster than the standard MIP algorithm and consistently yields near-optimal solutions. Combined with the MIP formulations, we now have a general and effective method for capturing and solving many similar resource management problems. We also provide a heuristic min-cost algorithm, and generalize the optimal-point method from related work. Those two heuristic algorithms are even faster but yield less optimal solutions.

The largest data centers may have several million PMs, hosting even more VMs. To solve resource management problems at such a large scale, we suggest a hierarchical decomposition method. The idea is to take advantage of the natural grouping of the PMs in a data center to define PM groups, such as a server rack, a computer room or a company-defined service region. A set of global controllers first distributes all the VMs to different PM groups, according to a combination of criteria such as load balancing, proximity to clients, colocation requirements, thermal distribution, operation costs, etc. Then for each PM group, we have a much smaller management problem, which can be solved quickly. Furthermore, since these smaller problems are distributed and independent from each other, they can be solved locally and in parallel.

Appendix

A. Finding the Energy-Optimal Utilization Point

In Section II-C, we show the CPU power function is $P(\rho) = \alpha + \beta\rho^3$, where $\rho = f/f_{\max}$ is the normalized frequency (CPU utilization). The application execution time $T(f)$ can also be defined as a function of ρ: $\bar{T}(\rho) = T(f_{\max})/\rho = \bar{T}(1)/\rho$, where $\bar{T}(1)$ is the execution time when the CPU is running at full speed. We now derive the energy-optimal utilization point.

Assume the CPU is operating at a certain utilization steadily throughout the execution of an application. The energy consumption for running the application is

$$E(\rho) = P(\rho) \times \bar{T}(\rho) = (\alpha + \beta\rho^3)\big(\bar{T}(1)/\rho\big). \tag{27}$$

To find the minimum energy and the optimal ρ, we take the derivative of $E(\rho)$ with respect to ρ and solve for its root.

$$\frac{dE(\rho)}{d\rho} = \bar{T}(1)\,(-\frac{\alpha}{\rho^2} + 2\beta\rho\,) = 0. \tag{28}$$

The minimum energy is obtained at $\rho = \sqrt[3]{\frac{\alpha}{2\beta}}$. When taking $\alpha = 0.25$ and $\beta = 0.75$, as in our experimental setup (Section IV-A), the energy-optimal utilization point is $\rho = 0.55$.

References

[1] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization.* John Wiley & Sons, Aug. 2014.

[2] M. Goetschalckx, C. J. Vidal, and K. Dogan, "Modeling and design of global logistics systems: A review of integrated strategic and tactical models and design algorithms," *European journal of operational research*, vol. 143, no. 1, pp. 1–18, 2002.

[3] D. Wedelin, "An algorithm for large scale 0–1 integer programming with application to airline crew scheduling," *Annals of operations research*, vol. 57, no. 1, pp. 283–301, 1995.

[4] NRDC, *America's Data Centers Consuming and Wasting Growing Amounts of Energy*, 2013, http://www.nrdc.org/energy/data-center-efficiency-assessment.asp.

[5] M.-T. Chen, C.-C. Hsu, M.-S. Kuo, Y.-J. Cheng, and C.-F. Chou, "GreenGlue: Power optimization for data centers through resource-guaranteed VM placement," in *IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom), and Cyber, Physical and Social Computing(CPSCom)*, 2014.

[6] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," in *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2011, pp. 594–601.

[7] A. T. Yasuhiro Ajiro, "Improving packing algorithms for server consolidation," in *International Conference Computer Measurement Group*, 2007, pp. 399–406.

[8] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proceedings IEEE INFOCOM*, Apr. 2011, pp. 71–75.

[9] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks*, vol. 57, pp. 179–196, Jan. 2013.

[10] 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.* USENIX Association, 2008, pp. 10–10.

[11] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proceedings of the ACM SIGMETRICS*, 2009, pp. 157–168.

[12] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, pp. 755–768, May 2012.

[13] A. Berl, E. Gelenbe, M. D. Girolamo, G. Giuliani, H. D. Meer, M. Q. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The Computer Journal*, vol. 53, pp. 1045–1051, Sep. 2010.

[14] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *Proceedings of 2005 ACM SIGMETRICS*, 2005, pp. 303–314.

[15] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, 2003, pp. 38–48.

[16] *Amazon AWS EC2 Instances*, 2015, http://aws.amazon.com/ec2/.

[17] *Gurobi Optimizer*, 2015, http://www.gurobi.com.