J Supercomput (2017) 73:782–809
DOI 10.1007/s11227-016-1797-5

Energy efficiency of VM consolidation in IaaS clouds

Fei Teng[1,2] · Lei Yu[3] · Tianrui Li[1] ·
Danting Deng[1] · Frédéric Magoulès[4]

Published online: 5 July 2016

Abstract The energy efficiency of cloud computing has recently attracted a great deal of attention. As a result of raised expectations, cloud providers such as Amazon and Microsoft have started to deploy a new IaaS service, a MapReduce-style virtual cluster, to process data-intensive workloads. Considering that the IaaS provider supports multiple pricing options, we study batch-oriented consolidation and online

This work was partially supported by the National Natural Science Foundation of China (No. 61202043), the Soft Science Foundation of Sichuan Province (No. 2016ZR0034), the Open Research Fund of Key Laboratory of Network Intelligent Information Processing (No. SZJJ2014-049) and the State Key Laboratory of Software Development Environment of P. R. China under Grant (No. SKLSDE-2016ZX-25).

✉ Lei Yu
yulei@buaa.edu.cn

Fei Teng
fteng@swjtu.edu.cn

Tianrui Li
trli@swjtu.edu.cn

Danting Deng
ddt@swjtu.edu.cn

Frédéric Magoulès
frederic.magoules@ecp.fr

1 School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China

2 State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China

3 Sino-French Engineering School, Beihang University, Beijing 100191, China

4 Ecole Centrale Paris, Grande Voie des Vignes, 92295 Châtenay-Malabry, France

placement for reserved virtual machines (VMs) and on-demand VMs, respectively. For batch cases, we propose a DVFS-based heuristic TRP-FS to consolidate virtual clusters on physical servers to save energy while guarantee job SLAs. We prove the most efficient frequency that minimizes the energy consumption, and the upper bound of energy saving through DVFS techniques. More interestingly, this frequency only depends on the type of processor. FS can also be used in combination with other consolidation algorithms. For online cases, a time-balancing heuristic OTB is designed for on-demand placement, which can reduce the mode switching by means of balancing server duration and utilization. The experimental results both in simulation and using the Hadoop testbed show that our approach achieves greater energy savings than existing algorithms.

Keywords Cloud computing · VM consolidation · Energy efficiency · DVFS · MapReduce

1 Introduction

Infrastructure as a Service (IaaS), as implemented by Amazon EC2 and Microsoft Azure, has quickly moved into the spotlight within the realm of cloud computing. Large-scale, virtualized datacenters are severing the ever growing demand for the IaaS computing resources. However, these datacenters result in a significant level of energy wastage, and cause both a high expenditure and environmental concern. According to Amazon's estimates, the energy-related costs of datacenters account for 42 % of the total operating costs. One reason that contributes to the high energy cost is that servers in datacenters are typically only 20–30 % utilized, resulting in energy efficiency of less than 50 % [18]. If energy efficiency of the servers during low utilization periods is increased, cloud providers may reduce the total energy consumption as well as the operation overhead of the datacenters.

A general method for improving energy efficiency in datacenters is to consolidate virtual machines (VMs) on a reduced set of physical servers to switch unused servers off or at least to a sleep mode [24]. Although a large number of researches have examined energy-saving VM consolidation in datacenters [2,14,25,30,38,40,43], most of them have the same assumption that VMs are independent of each other. Notice that a leading MapReduce-style framework for big data analytics becomes a new service provided directly by IaaS clouds. For example, Amazon EMR views Hadoop as a service and charges users for the computation as they actually use. EMR can build a virtual Hadoop cluster in minutes and tear it down after job completion. In such circumstances, the VMs to consolidate are no longer independent, but are in groups. The VMs of the same group are identical, with the same configuration requirement. In this paper, our aim is to help IaaS cloud providers (e.g., Amazon EMR) achieve energy efficiency when they operate datacenters.

Another novelty of the study is our energy-efficient decision including a cluster-level VM consolidation and a server-level frequency scaling. Dynamic Voltage Frequency Scaling (DVFS) is a traditional technique that provides fine-grained energy optimizations for IT equipments [39]. Although DVFS-enabled processors have been widely used in cloud datacenters, DVFS-based VM consolidation is still an open research direction, especially lacking of theoretical analysis to show to what extent scaling frequency can save energy. To the best of our knowledge, this is the first study that works on a co-optimization solution for IaaS clouds.

In this paper, the problem of consolidating virtual clusters is modeled as minimizing the energy consumption of the IaaS provider with a number of DVFS-enabled servers. The cloud provider predefines a certain number of VM types from which customers can choose, and meanwhile it offers two pricing options, a discounted reserved-instance rental rate and a pay-as-you-go rate. Two cases are discussed, a batch-oriented consolidation for reserved VMs and online VM placement for on-demand VMs.

The main contributions of this paper are as follows. For batch-oriented cases, a DVFS-based heuristic is proposed to consolidate separate virtual clusters on physical servers to reduce the energy consumption while guaranteeing job Service-Level Agreements (SLA). We prove the most efficient frequency that minimizes the energy consumption, and deduce the upper bound of energy saving through DVFS. More interestingly, this frequency only depends on the type of processor. For online cases, a time-balancing heuristic is designed for on-demand placement, which can reduce the mode switching by means of balancing server duration and utilization. The online placement heuristic is relatively simple, avoiding parameter-tuning efforts, and fits for homogeneous and heterogenous systems.

The paper is structured as follows. Section 2 presents an analysis of existing VM consolidation algorithms in datacenters. Section 3 formulates the optimization problem. Section 4 introduces batch solutions for the energy-efficient consolidation of VMs, while Sect. 5 extends VM placement for online cases. In Sect. 6, we present experimental results based on simulations and Hadoop testbeds. Section 7 concludes the paper and highlights areas of future work.

2 Related work

VM consolidation has been intensively investigated in the last decade. For the sake of readability, we organize the related work along three main research directions. We start with the criteria that classify the work on energy consumption reduction. Next, we discuss works on VM consolidation for online and offline workloads, respectively. Finally, we highlight works on the energy efficiency of a MapReduce-related datacenter.

There has been a significant amount of prior work on reducing the energy consumption of large-scale datacenters, divided into two categories, cluster-level and server-level. The cluster-level approaches apply vary-on/vary-off to reduce the cluster energy consumption of physical servers [2]. These approaches are effective to achieve high cluster-wide energy proportionality through dynamic resizing, load dispatching, VM consolidation and migration to power down idle servers [40]. The server-level approaches target individual components within a server, such as DVFS for processors [35,37], MemScale for memory [9], and disk spin-down for hard drives [9]. Both Verma et al. [35] and Von Laszewski et al. [37] scale down processor frequencies to minimize power dissipating by heuristics. The two heuristics can optimize an existing placement during VM migration, but are not suitable for VM consolidation that determines a placement for a new set of VMs. For VM requests in IaaS clouds, we make a co-optimization of the cluster-level and server-level energy efficiency through VM consolidation and frequency scaling, respectively.

The VM consolidation problem usually takes server capability as a pre-given constraint either for the homogeneous or for the heterogeneous environment, so we distinguish existing works by the characteristics of the workloads [1]. For offline workloads, random-sized VMs arrive and need to be placed into finite-sized physical servers. VMs do not leave or move between servers, and a typical objective is to minimize the number of occupied servers in order to turn off a subset of servers. For online workloads, VMs arrive at random times to be placed into servers and leave after a random service time. Servers can host multiple types of VMs, as long as the packing constraints are satisfied. VM requests are queued, and a typical objective is to minimize service latency to reduce power-on periods.

VM consolidation for offline workloads can be formulated as a bin packing (BP) problem where each server is a bin and each VM is an item to be packed [31]. Considering BP has an NP-hard computational complexity, there are three kinds of solutions to address this type of problem. First, the most popular solutions are greedy heuristics [27]. Verma et al. [35] applied First Fit (FF) descending to generate an initial assignment of VMs and later used Best Fit (BF) to reassign VMs picked from servers that violate resource constraints. Goiri et al. [11] and Quang-Hung et al. [28] studied core-based heuristic and BF descending heuristic to choose the most energy-efficient server for mapping each VM, respectively. These heuristic methods are applicable to stateless requests [34], based on the recognition that we can use the capacities of a bin to the fullest extent after the current allocation is performed. Second, evolutionary computation methods are used to perform approximating optimal solutions. The examples are simulated annealing algorithm [41], hybrid genetic algorithm [32], frog-leaping algorithm [23], machine learning algorithm [3] and artificial bee colony algorithm [14]. These approaches yield better results, but require much longer runtimes than greedy heuristics. In addition, there is no guarantee whatsoever that they will work similarly well under other circumstances (e.g., other types of hosts and VMs, other workload characteristics, etc.). Third, there have also been some attempts to solve the VM consolidation problem optimally, by integer linear programming [13], pseudo-Boolean optimization [29] and boolean integer linear programming [42]. These solutions can guarantee the results to be optimal, but they are nearly unusable in practice. The runtime becomes prohibitively large for instances of even moderate size. Notice that the above works are based on the same assumption that VMs are independent of each other, which means that each packing VM has its own constraints. Our problem is different. The packing item becomes a virtual cluster, where VMs in the same cluster are identical, with the same constraints. In this paper, we also apply the heuristic method, but show more effective than traditional greedy heuristics.

VM consolidation for online workloads has heavily relied on prediction-based or statistical approaches [15]. Chen et al. [7] applied a multiplicative seasonal auto regressive moving average method to predict server workloads in each time interval and then made power control decisions based on steady-state queuing analysis and feedback control theory. Govindan et al. [12] characterized the statistical properties of the power needs of hosted workloads through a measurement-driven profiling and prediction framework. The common problem with such approaches lies in the dubious feasibility of making accurate predictions of future request patterns in the IaaS cloud. Urgaonkar et al. [33] considered the full effects of queuing dynamics by making use of the queue backlog information to make online control decisions. Zhou et al. [43] took advantage of Lyapunov optimization techniques to control the admission of requests from diverse applications at any given time, so that the SaaS cloud can maximize its operating revenue by increasing the datacenter performance per watt. However, the main focus of these works was on building an analytical framework, without real system implementation and benchmarking applications to evaluate its performance.

Some researchers began to investigate energy-saving mechanisms for MapReduce on homogeneous machines, such as DVFS [39], data placement [42] and compression [8]. Wirtz and Ge [39] used an experimental approach to study the scalability of performance, energy, and efficiency of MapReduce, but they only discussed the relationship between the three factors without giving energy-saving solutions. Lang and Patel [18] observed that using all available nodes for workload execution can benefit both performance and energy consumption for certain applications. Leverich and Kozyrakis [20] showed that running a Hadoop cluster with a covering subset of nodes and powering off the others can save energy with performance tradeoffs. Kaushik and Bhandarkar [16] partitioned HDFS into disjointed hot and cold zones, placing frequently accessed data in the hot zone, which is always powered. This scheme is problematic because the output of every job would thereby be located on a small number of machines, creating a severe data hotspot for future access. Similar to the partition scheme in GreenHDFS [16], MIA studied mixed workloads on MapReduce by assigning different SLAs to services [6], allowing interactive jobs to operate on a small pool of dedicated machines while less time-sensitive jobs run on the rest of the cluster in a batch fashion. Mashayekhy et al. [26] proposed schedulers to assign server slots to minimizing energy consumed when executing MapReduce jobs. In all above algorithms, MapReduce is deployed on physical servers, rather than VMs. Li et al. [21] assigned MapReduce jobs to geo-distributed virtual clusters, but their focus was to optimize the traffic routing among different clouds. Cardosa et al. [4] explored an energy-aware MapReduce jobs in the cloud and proposed a bin-based recipe packing heuristic to make spatio-temporal tradeoffs for VM placement. The performance of recipe packing is determined by a tunable parameter, which should be well chosen through a large number of empirical tests. This parameter limits the scope where recipe packing can be applied. In addition, Cardosa et al. [4] did not incorporate fine-grained energy optimizations on the sever level.

Our work adopts the recipe idea in [4], but differs substantially from the previous work at least three important aspects. We optimize the energy consumption on two different levels, the server level and the cluster level. (1) On the server level, we propose a DVFS-based algorithm that can reduce the energy consumption by scaling processor's frequency. From theoretical aspect, we analyze to what extent scaling frequency can save energy. More interestingly, the most efficient frequency only depends on the type of processor. (2) On the cluster level, we propose a heuristic to consolidate VMs on servers, whose performance is not sensitive to tunable parameters. This heuristic also reduces the complexity of generating the possible combinations of VMs residing on

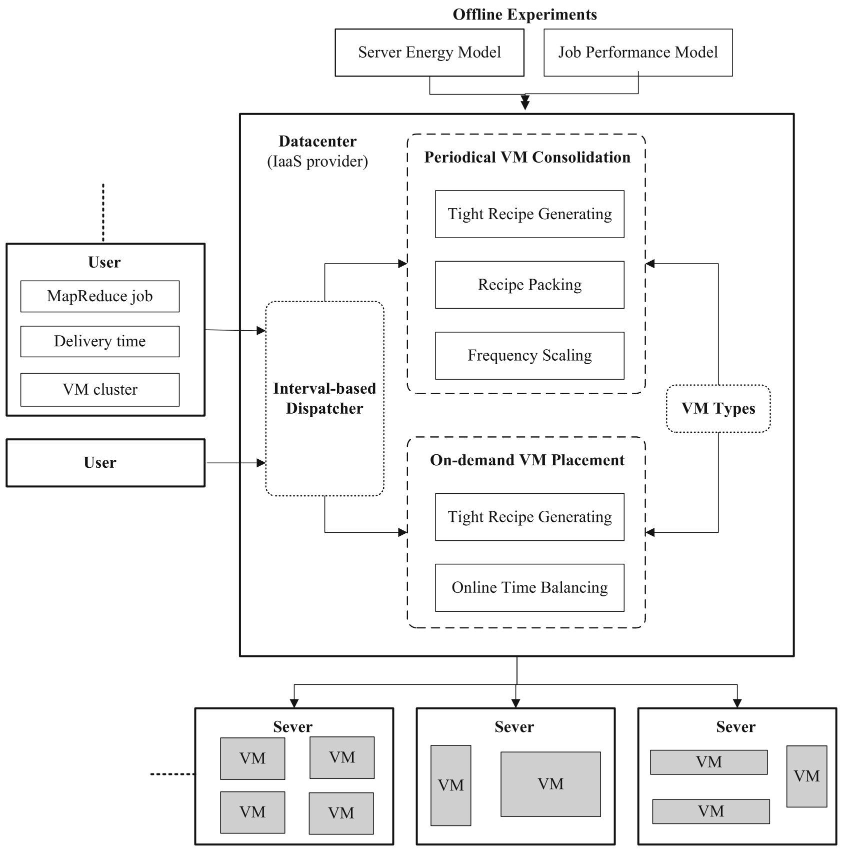


Fig. 1 System architecture

a particular server. (3) The demand of the MapReduce-style platform is burst, so an optimal solution for VM consolidation cannot maintain its good performance upon the arrival of new requests. We, therefore, investigated a non-optimal but fast heuristic. Our solution is simple and straightforward, which works well on the homogenous and heterogeneous systems.

3 System architecture

This section introduces the basic assumptions of the system architecture shown in Fig. 1, including the cloud datacenter, users, servers and VMs, as well as the energy consumption model and the MapReduce performance model.

3.1 Datacenter and user

In this paper, IaaS provider has a single datacenter, which manages a set of physical servers hosting VMs. A server $\overrightarrow{r}$ is characterized by the amount of capacity for each resource (CPU, memory, disk). The datacenter predefines a selection of VM types optimized to fit different use cases. VM types are also characterized by virtual resource capacity $V = \{\overrightarrow{v_1}, \overrightarrow{v_2}, \ldots, \overrightarrow{v_k}\}$. The number k of VM types is small compared with the potentially large number of hardware configurations. For example, Amazon EMR provides four VM types for general purpose, three types for compute optimized purpose, four types for memory optimized and two types for storage optimized.

SLA is a standardized contract where particular aspects of the service are agreed between the service provider and the service user [10]. IaaS user j specifies his service as $\{\overrightarrow{v_j}, n_j, A_j, D_j\}$, where $\overrightarrow{v_j}$ is the VM type most appropriate for the MapReduce job, n_j is the number of VMs to build a private virtual cluster, A_j is the arriving time of the request, and D_j is the delivery time of the virtual cluster. For example, Amazon users customize the VM instance, the number of instances and the Hadoop version before accessing an EMR cluster.

IaaS provider needs to find efficient packing solutions for these VM requests across a large pool of servers. A recipe is a list $\overrightarrow{w} = \{w_1, w_2, \ldots, w_k\}$ for filling a single physical server with zero or more VMs of each type without exceeding the capacity of the server [4]. The resource utilization for a physical server with recipe $\overrightarrow{w}$ is presented by

$$U = \frac{\sum_{q=1}^{q=k} w_q * g(\overrightarrow{v_q})}{g(\overrightarrow{r})}, \tag{1}$$

where $g(\overrightarrow{r})$ is a scalar value of capacity for the multidimensional resource $\overrightarrow{r}$. Notice that $\overrightarrow{w} * \overrightarrow{v} \leq \overrightarrow{r}$.

The datacenter supports two different assignment strategies based on pricing schemes. Batch-oriented assignment collects the reserved requests within a certain period and processes VM consolidation periodically, while online assignment makes the VM placement decision immediately upon receiving a new request.

In batch scenarios, users subscribe delivery time D for their MapReduce workloads. Generally speaking, delivery time D should be longer than the real execution time. Besides, as the arriving requests are batched for some time before being provisioned, we assume arriving time A is the same for all requests.

In online scenarios, the requests arrive in a random fashion, thus A varies with different workloads. On-demand users pay for compute capacity without duration commitments. The delivery time D is equal to the real execution time of the MapReduce job.

3.2 MapReduce performance

MapReduce jobs are data dependent and inherently parallel. A typical MapReduce job consists of a map stage and a reduce stage. In each stage, tasks have the same function (map/reduce) and the same size of input data. For example, every map task takes a default 64M split as input. Additionally, the VMs running individual job are of the same type, so that the execution time for map/reduce tasks are very close. Therefore, the MapReduce job runtime depends on two main factors, the size of the input data and the virtual cluster assigned to the job. We estimate the runtime T as:

$$T = tm * \left\lceil \frac{mNum}{mSlot * n} \right\rceil + tr * \left\lceil \frac{rNum}{rSlot * n} \right\rceil + \gamma \tag{2}$$

where tm and tr are the average runtime for a single map/reduce task, which can be easily obtained using historical information [36]. γ is a correction constant calculated from offline experiments. $mNum$ is determined by the size of the input data and default split. A single VM has $mSlot/rSlot$ slots to run map/reduce tasks in parallel. n is the deployment size.

For simplicity, we assume that every request contains only one MapReduce job. Laxity α is defined to represent the ratio of the MapReduce runtime to the request delivery time. Clearly, $\alpha \leq 1$.

$$\alpha = \frac{T}{D}. \tag{3}$$

3.3 Energy consumption

Power consumption by computing servers in datacenters is mostly determined by the CPU, memory and disk storage. In comparison to other system resources, the CPU consumes the main part of energy [2], and hence this paper applies a basic power consumption model, in which the CPU processing capacity is the main bottleneck for severs [33]. CPUs can be operated at different speeds at runtime using DVFS techniques. For a DVFS-based server, the power consumption is represented by frequency f.

$$p(t) = C_1 f^3 * u(t) + C_2, \tag{4}$$

where C_1 and C_2 are two constants. C_1 is the scale coefficient and C_2 is the static part existing even when there is no workload (due to the leakage). This function is justified by close match between the data sheet curves of real DVFS processors and the analytical curves [44]. u is CPU utilization level. As modern servers usually contain multi-cores, CPUs follow a linear power-utilization relationship [17,19]. At time t, the server hosts VMs as the recipe $w(t)$, so its normalized CPU utilization is

$$u(t) = \frac{\sum_{q=1}^{q=k} w_q(t) * v_q^1}{r^1} \tag{5}$$

where v_q^1 and r^1 are the CPU dimensions in configuration vectors $\overrightarrow{v_q}$ and $\overrightarrow{r}$, respectively. Notice that recipe $w(t)$ changes over time. Once a MapReduce job is finished, the idle VMs turn to hibernating states. Once the reserved delivery time is due, the idle VMs are cleared away from their corresponding servers. Therefore, no VMs exist if all the jobs on the server are finished.

The energy consumption $e(t)$ is defined as an integral of the power consumption function over a period of time

$$e(t) = \int_{t^s}^{t^f} p(t) \mathrm{d}t \tag{6}$$

Table 1 Notations in system architecture

$\overrightarrow{r}$	Resource capacity of server
$\overrightarrow{v}$	Resource capacity of VM
$\overrightarrow{w}$	Recipe
m	Number of server
k	Number of VM type
U	Multidimensional resource utilization
u	CPU utilization
f	Frequency (speed) of server
$f_{\max}$	Maximum frequency (speed) of server
f_{eff}	Energy-efficient frequency (speed) of server
A	Arriving time of virtual cluster
D	Delivery time of virtual cluster
T	Runtime of MapReduce job
α	Laxity of delivery time
p	Power
e	Energy

where t^s is the time when the server starts to run the first job and t^f is the time when the last job finishes. The total energy consumption E is measured as:

$$E = \sum_{i=1}^{m} e_i(t) \tag{7}$$

In (7), m is the number of active servers in the datacenter. Table 1 lists the notations in the system model.

4 Batch-oriented VM consolidation

The batch-oriented VM consolidation problem is stated as follows. Given a set of requests to the MapReduce-style virtual cluster, the cumulative energy of active servers is minimized through selecting recipes and frequencies. Therefore, a consolidation decision is made by two steps: recipe packing and frequency scaling.

4.1 Recipe packing

A recipe is a list for filling a single server without exceeding the capacity of the physical server. For instance, suppose a cloud provider supports three VM types VM1, VM2, and VM3, then a recipe can be represented by $\langle 2, 1, 2\rangle$, corresponding to placing two VMs of type VM1, one of type VM2, and two of type VM3 together on a server. A saturated recipe cannot be covered by any other recipes. A pure recipe contains only one VM type.

As the number of predefined VM types is typically small, the complexity of enumerating all possible recipes is computationally feasible. If the server capacity and the VM types are fixed, the list of recipes is invariant. For a homogeneous datacenter, the list can be applied to every physical server, so generating recipes is a one-time process.

We first generate saturated recipes, and then sort them by VM types inside recipes. A recipe with the less types is assigned to the higher priority, that is to say, the pure recipes are ranked first before the mixed recipes. By doing this, the VMs belonging to the same job can be concentrated on a few number of servers. If some recipes containing the same number of VM types, they are sorted by their CPU utilizations.

Algorithm 1 GenerateSaturatedRecipes(VMs vms, Server s)

```
create a $recipe$, a set of $vms.size()$ zeros;
set $rList$, $fullList$ and $tempList$ to be empty;
for $i = 1$ to $vms.size()$ do
  for $j = 1$ to $fullList.size()$ do
    let $newRecipe$ be $fullList.get(j)$ adding one VM of type $vms.get(i)$;
    while $newRecipe.getUtiliztion() < 1$ do
      add $newRecipe$ to $tempList$;
      add one VM of type $vms.get(i)$ to $newRecipe$;
    end while
    delete one VM of type $vms.get(i)$ to $newRecipe$;
    $satuRecipe = newRecipe$;
    $rList.add(satuRecipe)$;
  end for
  $fullList.addAll(tempList)$;
  set $tempList$ to be empty;
end for
calculate VM $types$ for every recipe in $rList$;
calculate CPU $utilization$ for every recipe in $rList$;
sort $rList$ by $types$ in an ascending order;
sort $rList$ by $utilization$ in a descending order;
return $rList$
```

In Algorithm 1, all possible recipes for server s are generated by incrementally considering possible VM coplacement configurations $tempList$. In each $tempList$, $satuRecipe$ is selected if the recipe cannot add any VM within server capacity. After a basic $rList$ is obtained, we calculate the VM types and CPU utilization for each recipe, and sort $rList$ according to VM $types$. If some recipes contain the same number of VM types, recipes are further ordered by their CPU $utilization$. The saturated $rList$ can be precomputed and maintained in a database for future lookup.

The recipe placing problem can be solved by the combination of Algorithms 1 and 2, named Tight Recipe Packing (TRP). First, the VM requirements of each user are filled into a private poll. Algorithm 2 places VMs inside each private pool on servers according to the saturated list. Second, the rest VMs in private pools that cannot pack a full server are filled into a public pool. Algorithm 2 repeats again till no saturated recipe is found for the VMs in the public pool. The rest VMs are then filled into the last server.

Algorithm 2 PlaceVMs(RECIPELIST $rlist$, VMPOOL vp)

```
while $vp$ is not empty do
  $flag = $ TRUE;
  create an empty server $s$;
  for $i = 1$ to $rlist.size()$ do
    $r = rlist.size()$;
    if $r$ matches a subset of $vp$ then
      remove $r$ from VM pool $vp$;
      pack $s$ as the recipe $r$;
      add $s$ to $servers$;
      // VMs in $vp$ can pack a full server
      $flag = $ FALSE;
      break;
    end if
  end for
  if $flag$ then
    break;
  end if
end while
return $servers$
```

4.2 Frequency scaling

After a server is packed by a particular recipe, the server utilization is fixed. We are interested in investigating the relationship between server energy and CPU frequency. The fundamental challenge is how to minimize the energy consumption on the server while ensuring to finish jobs within the user-defined delivery time.

$$\begin{aligned} \min \quad & e \\ \text{s.t.} \quad & T \leq D \end{aligned} \tag{8}$$

The minimization is complicated due to MapReduce parallel computing model. On the one hand, an individual job is executed on multi-VMs or even on multi-servers. On the other hand, a server supports multiple jobs running simultaneously. The total completion time of a MapReduce job is straggled by the slowest task. If the host servers operate at different frequencies, the runtime estimation turns to be more difficult. Especially when one task is assigned to the server at a much lower frequency than others, the probability of exceeding delivery time increases greatly. We propose Frequency Scaling algorithm (FS) that only changes the frequency of the server, on which all VMs execute the same MapReduce job. Notice that TRP firstly uses private pools to assign VMs. If the VMs required by one user can fill at least one server, over 50 % of tasks of the MapReduce job can scale frequency. FS makes a majority of tasks slow down, and a minority of tasks keep the same execution speed. Thus, the estimated runtime T calculated by Eq. (2) is actually longer than the real MapReduce runtime T_{real}. We take T, instead of T_{real}, as the target for frequency scaling.

As all VMs on the server execute the same MapReduce job, utilization u is a constant value when the job is running. The cumulative energy consumption is simplified as:

$$e(t) = \int_0^D p(t)\mathrm{d}t = p_u * T \tag{9}$$

where p_u is the power when MapReduce job running, and T is the estimated runtime for the MapReduce job.

Theorem 1 *For a server running one MapReduce job, the energy consumption is minimized if the following condition holds:*

$$f_{\text{scale}} = \max\left[\left(\frac{C_2}{2uC_1}\right)^{1/3}, \alpha f_{\max}\right] \tag{10}$$

Proof Let C_0 be the execution requirement in CPU cycles. We replace T by C_0/f, and obtain:

$$e = p_u * \frac{C_0}{f} = C_0 C_1 u f^2 + \frac{C_0 C_2}{f}.$$

Take the partial derivative with respect to f, and then set the first-derivative condition to zero:

$$\frac{\partial e}{\partial f} = 2uC_0C_1 f - \frac{C_0C_2}{f^2} = 0.$$

We obtain $f = (\frac{C_2}{2uC_1})^{1/3}$, which means that e could have an extreme point. Take the second-derivative condition as:

$$\frac{\partial^2 e}{\partial f^2} = 2uC_0C_1 + 2\frac{C_0C_2}{f^3} > 0,$$

which means e can be minimized. We then obtain the most energy-efficient frequency

$$f_{\text{eff}} = \left(\frac{C_2}{2uC_1}\right)^{1/3}.$$

As the server utilization u is fixed, the frequency f_{eff} only depends on C_1 and C_2, which are constants in power function. This result reveals that the energy-efficient frequency is actually determined by the CPU model of server.

Notice that the job runtime T is constrained by the user-defined delivery time D. The scaling frequency must guarantee

$$f_{\text{scale}} > \frac{C_0}{D} = \alpha f_{\max},$$

where $f_{\max}$ is the highest frequency of the processor, and α is the ratio of the estimated runtime to delivery time.

In conclusion, we have $f_{\text{scale}} = \max[(\frac{C_2}{2uC_1})^{1/3}, \alpha f_{\max}]$. □

Algorithm 3 ScaleFrequency(SEVER s, FREQLIST $freqs$)

```
**if** $s$ contains a PURE recipe **then**
    $u = s.getUtilization()$;
    $f_e = (\frac{C_2}{2u(i)C_1})^{1/3}$;
    $f_b = f_{\max} * T/D$;
    $f_s = \max[f_e, f_b]$;
    choose the smallest value $f$ from $freqs$ while guaranteeing $f \geq f_s$;
    $s.setFreq(f)$;
**end if**
```

Algorithm 3 shows how to scale the frequency for a specific server. If the server contains one MapReduce job, the efficient frequency f_s is calculated according to Eq. (10). Since the frequencies scaling in the server are discrete, FS scales f as the smallest frequency in freqs which satisfies $f \geq f_s$. f is the frequency for the server and the VMs residing on the server.

The upper bound of energy saving by FS is

$$\beta = \frac{e_{\max} - e_{\text{eff}}}{e_{\max}} = 1 - \frac{(C_1 u f_{\max}^3 + C_2)/f_{\max}}{(C_1 u f_{\text{eff}}^3 + C_2)/f_{\text{eff}}}, \tag{11}$$

The theoretical bound is actually the upper limit for energy saving, which helps us measure energy-efficiency for different algorithms.

Although Algorithm 3 is designed for the server executing one MapReduce job, FS can be extended to scale frequencies of the servers with multiple applications requesting the same type of VM. In such case, different jobs have their own laxity α. We could take the maximum α^* among all jobs to scale the frequency for the server according to Eq. (10). As a result, energy consumption could be further reduced.

5 Online VM placement

In online scenarios, the cloud provider reacts immediately to on-demand service requests, so multiple jobs cannot be binned together a priori for provisioning. A new job is processed in the presence of other already-running jobs in the datacenter. As the delivery time of virtual cluster exactly depends on the job runtime, frequency scaling is not suitable for online scenarios. We assume every server works at its dominant (maximum) frequency.

Online VM placement needs to address three issues: how to manage the active resources, how to utilize the nonempty servers, and the point when the datacenter changes the state (active/hibernating) of servers.

For the first issue, a database is maintained to profile server characteristics such as resource utilization, running VM types and numbers, and the estimated completion time of running jobs. This information is updated with the arrivals and departures of virtual clusters.

For the second issue, a Time Balance Index (TBI) δ_i is defined to indicate the time balance of server i to host the coming VMs. When a new request j arrives, its delivery time is predicted as the job runtime added to its arriving time $A_j + T_j$. We then check for nonempty servers in the database and update the completion time L_i of the latest job on server i. TBI for server i is defined as:

$$\delta_i = L_i - (A_j + T_j). \quad (12)$$

The non-empty servers are sorted based on two rules. (1) The server with a positive value of δ_i has a higher priority to host VMs than the sever with a negative δ_i. This implies that VM placement should efficiently use the non-empty server without extending the active duration. (2) The absolute value of δ_i represents the time balance of the running jobs and the incoming job, so non-empty servers are sorted in an ascending order of $|\delta_i|$.

The time complexity for placing one virtual cluster is $O(m)$, where m is the number of non-empty servers. After placing one VM on a certain server, δ_i keeps the same value as long as the server is still non-empty. The current server is still the first choice for next VM. Therefore, it is unnecessary to update δ_i right after VM placement. By contrast, recipe-based placement has a much higher complexity than the TBI-based. Enumerating all possible recipes is a time-consuming process. If all the servers are empty, these homogenous servers can use the same recipe list. However, non-empty servers have different residual capacities, which need to repeat the process of recipe generating for m times.

For the third issue, if the active servers cannot satisfy the demand of the coming request, new servers are powered on to place the remaining VMs. If one server has no jobs running, it changes to a hibernating state to save energy. Our solution tries to keep the active servers busy, ignoring extra overhead caused by mode switching.

Algorithm 4 shows how to carry out online VM placement on physical servers for an incoming request. Once a new request $newVMs$ arrives, its absolute job completion time is estimated. For each active server, δ_i is calculated by (12). The active servers are then sorted according to δ_i. If the capacity of active servers is not enough for resource requirements, new servers turn active to place the subset of the remaining set of VMs. These servers are added to $activeServers$. Algorithm 4 is named as Online Time Balancing (OTB).

6 Performance evaluation

This section highlights the benefits of using batch-oriented consolidation and online placement. Power consumption function is obtained through simulation with Sniper [5] and McPAT [22]. We develop a simulator-based java, and configure the important input parameters such as VM types as Amazon EMR. We also build a Hadoop cluster consisting of 14 PowerEdge R710 servers, and use Cpufrequtils to control the CPU frequency scaling.

6.1 Power function simulation

The power function of CPU plays an important role for selecting the energy-efficient frequency for servers. In the following simulation, Intel Xeon E5640 is taken as an example to evaluate the consolidation performance.

Algorithm 4 OnlinePlace(SEVERLIST $activeSevers$, VMPOOL $newVMs$)

```
create $\delta$, a vector of $activeSevers.size()$ zeros;
for $i = 1$ to $activeSevers.size()$ do
    calculate $\delta_i$ for server $i$;
end for
if $\delta_i \geq 0$ then
    sort $activeSevers$ in an ascending order of $\delta_i$;
else
    sort $activeSevers$ in a descending order of $\delta_i$;
end if
for $i = activeSevers.size()$ to 1 do
    //for non-empty servers
    while $newVMs$ is not empty do
        assign a VM to $activeSevers.get(i)$;
        if $activeSevers.get(i).getUtilization() > 1$ then
            delete the VM from $activeSevers.get(i)$;
            break;
        end if
        remove the VM from pool $newVMs$;
        update $activeSevers.get(i)$;
    end while
end for
//for empty servers
if $newVMs$ is not empty then
    $rList$ = GenerateSaturatedRecipes($newVms$, $s$);
    $newServers$ = PlaceVMs($rlist$, $newVms$);
    add $newServers$ to $activeSevers$;
end if
return $activeSevers$
```

Table 2 Power consumption and frequency for E5640

Frequency (Ghz)	2.66	2.53	2.40	2.13	1.86
Power (1 core)	92.2	90.1	87.5	83.3	81.2
Power (4 core)	140.6	131.4	123.4	108.1	98.7

E5640 is based on the Nehalem microarchitecture and uses the 45 nm manufacturing methods. It supports DVFS techniques using the Enhanced Intel SpeedStep Technology. The base frequency set contains {2.66, 2.53, 2.40, 2.26, 2.13, 2.0, 1.86}.

We use Sniper [5] and McPAT [22] to simulate the power consumption of processors varying with CPU's frequency. Sniper integrates with McPAT, allowing for estimating the program execution and processor power consumption. For each frequency in base set, an one-rank and a four-rank MPI applications are launched on a four-core E5640 processor. The power consumption for the running cores is nearly the same, and the core leakage is very close, no matter the core is running or idle. This result agrees with our assumption. Table 2 shows the simulation results of power consumption and frequency for E5640. Based on Table 2, the values of C_1 and C_2 can be obtained by linear regression methods.

The power consumption function for four-core E5640 is $p(s) = 3.45 * u * s^3 + 75.62$, s in GHz, $p(s)$ in Watts. According to Theorem 1, $f_{\text{eff}} = (\frac{C_2}{2uC_1})^{1/3} = 2.23$. Since the base frequency set contains {2.66, 2.53, 2.40, 2.26, 2.13, 2.0, 1.86}, the energy-efficient frequency for E5640 is 2.26Ghz.

6.2 Simulation results

6.2.1 Batch cases

For batch-oriented VM consolidation, TRP is compared with Recipe Packing (RP) [4]. RP sorts the enumerated recipes by utilization u and applies FF heuristic to place VMs. Both TRP and RP can work with Bin Duration (BD) algorithm [4]. BD has an important tunable parameter T, which divides the runtime range of the MapReduce jobs into bins of width T. In each bin, VM is placed according to the recipe list that is generated by RP or TRP.

VM types are configured as the Amazon EMR instances for different purposes in Table 3. We assume that the physical server has 32core CPUs, 144 G memory and 16T storage. If processors in the server are based on the 45 nm Nehalem microarchitecture as four-core E5640, the power function of 32-core server may follow a linear power-utilization relationship, that is $p(s) = 27.6 * u * s^3 + 604.96$.

Cloud users specify the VM type, deploy size and delivery time. The laxity α represents the accuracy of prediction for the job runtime. $\alpha = 0.5$ means that one user subscribes a twice longer period than the job execution time. We generate 100 jobs in the simulation, and their configurations vary as the following:

1. the required VM type, randomly taken from Table 3;
2. the required VM number, taken from a uniform distribution on [8, 128];
3. the delivery time, taken from a uniform distribution on [1, 100] (hour);
4. the laxity of delivery time, taken from a uniform distribution on [0.5, 1].

TRP and RP generate recipes based on the same configuration of server and VM types. The size of the recipe list for TRP and RP is 8019 and 49,228, respectively. TRP can reduce the complexity by 84 % compared with RP.

Three performance metrics are measured, including sever number, energy consumption and Cumulative Machine Uptime (CMU). CMU presents energy usage in [4]. We extend CMU definition to make it support DVFS techniques

Table 3 Predefined VM types for simulation

VM type	CPU (core)	Memory (G)	Storage (G)
VM1	1	3.75	410
VM2	2	7.5	840
VM3	4	15	1680
VM4	2	1.7	350
VM5	8	7	840
VM6	32	60.5	3360
VM7	16	22.5	1680
VM8	2	17.1	420
VM9	4	34.2	850
VM10	8	68.4	1680

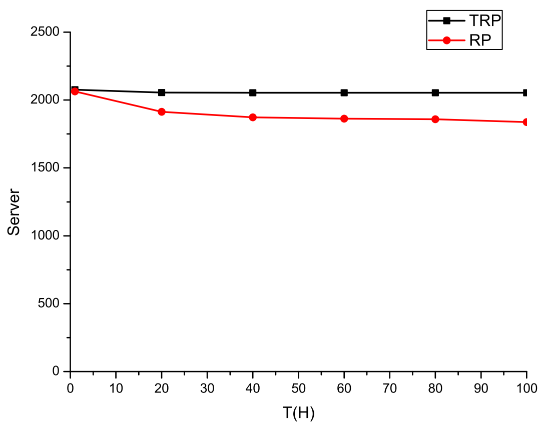


Fig. 2 Sever number for batch consolidation (simulation)

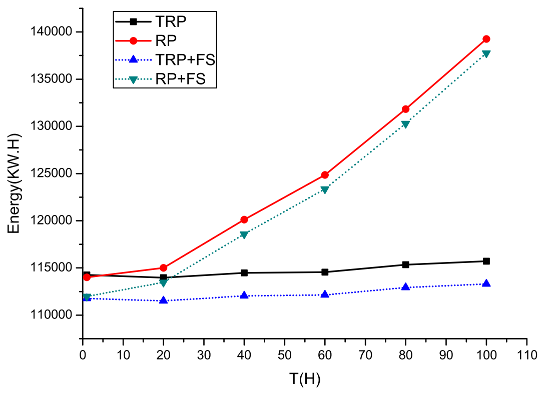


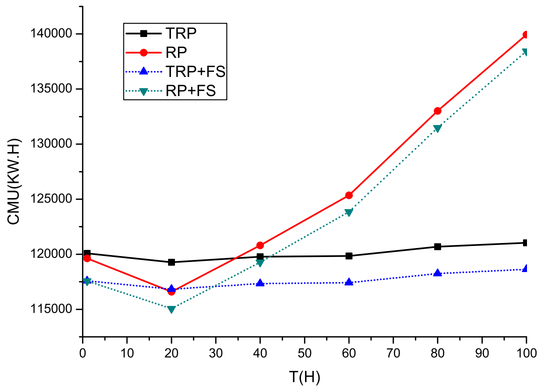
Fig. 3 Energy consumption for batch consolidation (simulation)

$$\mathrm{CMU}(t) = \int_{t^s}^{t^f} C_1 f^3 + C_2 \mathrm{d}t. \tag{13}$$

Figure 2 indicates the number of active servers for the two algorithms. RP sorts the recipe according to utilization, so that it needs less servers to host all VMs than TRP. Before placement, both algorithms partition VMs into distinct bins based on their delivery time. With the increasing bin size T, RP achieves better space fitting than TRP. Especially when $T = 100$ (also represents infinity in this simulation), the gap between the two reaches 7 %.

The metric of server number only shows space efficiency, but cannot measure energy efficiency. Figure 3 shows that TRP has better energy efficiency than RP. The energy consumption by TRP algorithm varies slightly with bin size T. The minimum value of energy consumption is taken when $T = 10$. The two extreme cases $T = 1$ and $T = 100$ increase the energy consumption by 0.35 and 1.63 %, respectively. This result indicates that TRP is not sensitive to bin size T. The energy consumption by RP algorithm performs like a convex function, reaching bottom at $T = 10$. When all VMs are placed in the same bin ($T = 100$), the datacenter consumes extra 22.3 %

Fig. 4 CMU for batch consolidation (simulation)



energy compared with the minimum value ($T = 10$). Therefore, the bin size should be well chosen to obtain a good tradeoff between time balancing and space fitting. As T changes along with different workloads, RP needs a large number of empirical tests to tune a reasonable T, which limits the scope where RP can be applied. For TRP, we suggest to apply $T = 1$ instead of the most energy-efficient $T = 10$, avoiding the time-consuming parameter selection.

In Fig. 4, CMU metric is measured for both algorithms. Similar results can be obtained as shown in Fig. 3. One exception is that the performance of TRP in range $T \in [1, 20]$ is worse than that of RP. The reason is that the power for an active server is a constant value in CMU model [4]. This assumption does not fit for modern servers, especially for those using multi-core processers. In reality, the power function has a close relationship with CPU utilization [19].

Energy consumption and CMU by FS are shown in Figs. 3 and 4, respectively. FS does not increase the number of servers but scales server frequency. FS can be applied to any existing consolidation algorithms. TRP with FS shows a 2.2 % (energy) and 2.1 % (CMU) improvement over its base algorithm. RP can save 1.5 % energy consumption and 1.3 % CMU usage, if FS is applied. Notice that the energy-efficient and maximum frequency are 2.26 and 2.66 Ghz in this experiment, respectively. The upper limit of energy saving is

$$\beta = \frac{e_{\max} - e_{\text{eff}}}{e_{\max}} = 3.4\,\%.$$

There is still a gap between our experimental result and the theoretical bound. The reason is that not every server scales its frequency. If we change another server with a higher frequency such as 3.06 Ghz, the upper bound for energy saving may improve to 12 %.

6.2.2 Online cases

For online VM placement, we compare OTB with Best Fit (BF) [35] and Online Bin Duration (OBD) [4]. BF places VMs into the active servers in order of increasing

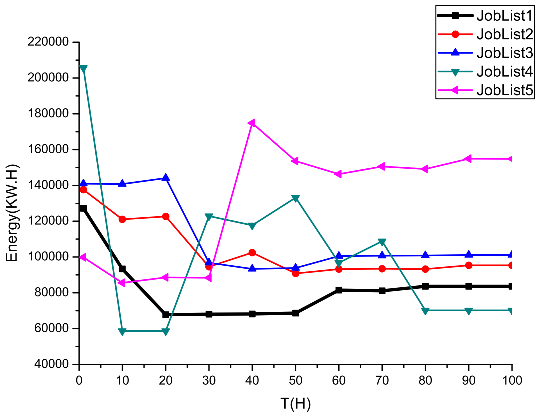


Fig. 5 Energy consumption of OBD (simulation)

utilization, which aims at maximizing the spatial utilization of resources. OBD only utilizes the servers on which the estimated remaining runtimes of all the VMs are within a given interval of the estimated runtime of the incoming job.

OTB and BF have no tunable parameters, while OBD has a critical parameter, bin size T. The tuning of bin size may result in differences in algorithm effectiveness. To select a reasonable bin size T for OBD, we randomly generated five workflows to find out if there is a general parameter for all cases. Each flow contains 100 random MapReduce jobs arriving in the datacenter as a Poisson process with the rate $\lambda = 5$. Other parameters are configured as the batch provisioning.

Figure 5 shows total energy consumption as T varies. Clearly, the bin size T for OBD has a great effect on energy consumption, much greater than that for batch consolidation. In these online tests, the random jobs are in a random sequence of workflow, which adds the uncertainty for OBD placement. In Fig. 5, different workflows have different bin sizes to obtain the minimal consumption value, so it is quite difficult to select a general T for all cases.

Next, we choose the black joblist1 as an example to compare the performing details for these online algorithms. The bin size for OBD is set as its most energy-efficient value $T = 20$.

The number of active servers for the three algorithms is shown in Fig. 6. Among the three algorithms, the average number of active servers scheduled by OTB is the smallest. As more jobs arrive in the datacenter, all the algorithms turn on new servers to place the required VMs. With the completion of old jobs and the arrival of new jobs, the number of active servers fluctuates. There are no more jobs submitted after 500, so the active servers gradually turn to hibernating states until the last job finishes.

In Table 4, the average value represents active servers, while the variance reflects the frequency of mode switching for these active servers. OTB performs the best with the smallest average and variance among the three algorithms, maximizing the instantaneous utilization of current resources while balancing the completion time of jobs running on the same server. OBD ($T = 20$) can also limit the time imbalance in each server within duration T, but it often powers on extra servers if no servers are in T. BF places VMs into servers in order of increasing utilization, resulting in VMs

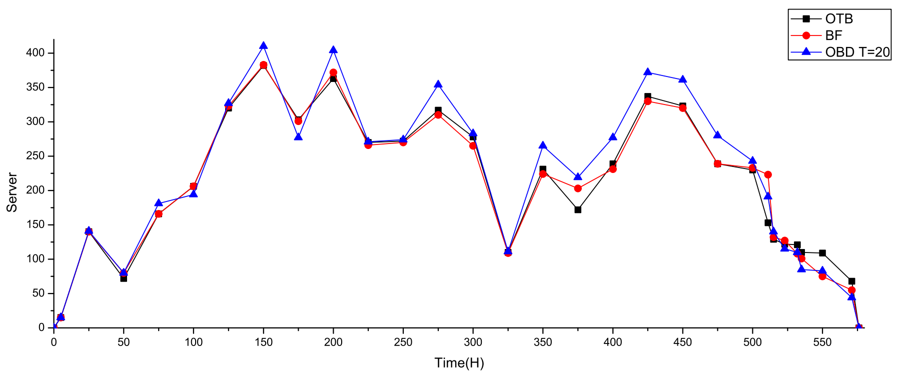


Fig. 6 Sever number for online placement (simulation)

Table 4 Server number for online workflow

Server number	OTB	OBD ($T = 20$)	BF
Average value	226	243	228
Variance value	106	120	108

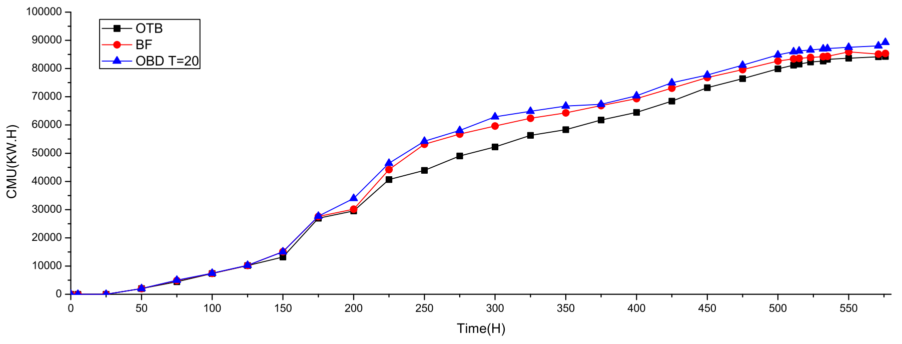


Fig. 7 CMU for online placement (simulation)

belonging to the same job scattered on different servers. BF also has a larger number of underutilized servers after previous jobs complete, as the duration of a server is determined by the slowest MapReduce job on the server. The time balance of BF is the worst compared with the other two algorithms.

The CMU and energy consumption for online workflows are shown in Figs. 7 and 8. As expected, the datacenter using OTB obtains the least CMU usage. The winner for energy saving is OBD, under the condition that T is well chosen as its best value. Although OTB and BF take similar number of servers to host virtual clusters, the energy consumption using OTB is much lower than that using BF in Fig. 8. From the point of view of complexity, OTB is still a promising heuristic for online VM placement.

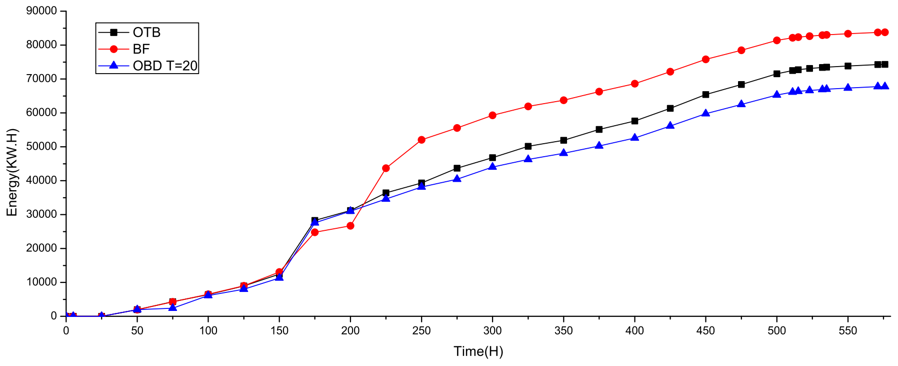


Fig. 8 Energy consumption for online placement (simulation)

Table 5 Predefined VM types for Hadoop

VM type	CPU	Memory (G)	Storage (G)
VM1	1 core	1	50
VM2	2 core	1	50
VM3	1 core	2	100
VM4	2 core	3	100

Table 6 MapReduce jobs

Benchmark	Data source	Data size (G)	VM type
Sort	Gridmix	10	VM1
Terasort	Teragen	3	VM2
Wordcount	Wekipedia	6.3	VM3
Wordcount	Wekipedia	29	VM4

6.3 Case study on Hadoop testbed

6.3.1 Batch cases

A case study is carried out on a Hadoop testbed with 14 homogenous servers. Each server is DELL PowerEdge R710 with one Intel Xeon E5640 processor, 6GB RAM and 600GB storage, running Xen 3.2 and Redhat operating system with the 2.6.18-164.e15 Linux kernel. Cpufrequtils is used to control the CPU frequency scaling. The base frequency set contains [2.66, 2.53, 2.40, 2.26]. All servers are connected using gigabit Ethernet. The operating environment supports four VM types that vary in CPU, memory and storage, as shown in Table 5.

There are 3 typical MapReduce benchmarks: sort, terasort and wordcount. Each job applies for a private virtual cluster running Hadoop 1.1.0 on JDK 1.7. The details of the four jobs and their VM requirements are shown in Table 6. Experiments are repeated 10 times by randomly varying the VM number in [3, 8].

Figure 9 shows the server number for RP and TRP with three different bin sizes. RP needs fewer servers to place the virtual clusters than TRP in most of cases, because the recipe list of RP intends to achieve good utilization fitting rather than energy efficiency. RP makes spatio-temporal balance through tuning bin size. The smaller the bin, the worse the utilization. The two algorithms have the same sever number when T takes the minimum value (T = Min). With the increase of T, TRP needs more servers to place the virtual clusters.

Notice that the columns (10 random experiments) in Figs. 9 and 10 are independent of each other. Due to hardware limitations, the testbed only supports 4 VM types, and it is quite easy to obtain the same results by the two algorithms. We, therefore, repeat each test 10 times randomly to validate the results.

Although RP has good spatial utilization, its energy consumption is higher than TRP in many cases. Figure 10 shows the corresponding energy consumption for these three bin sizes. When T = Min, RP assigns each job to a separate bin, while TRP prefers the pure recipe to place VM clusters. As a result, the two algorithms have the same placement for the virtual clusters, and generating the same energy consumption. Compared with RP, TRP saves 7 and 18 % energy in $T = 700$ and T = Max, respectively. FS can further reduce the energy consumption for both algorithms. These data are in agreement with the simulation results.

6.3.2 Online cases

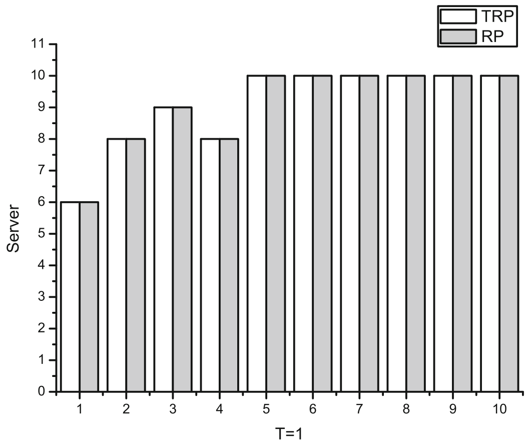
The online experiment tests a mixed workflow of 10 MapReduce jobs that fits a Poisson distribution with parameter $\lambda = 500$ s. The jobs in the workflow are randomly taken from Table 6. We repeat the test multiple times to ensure the results are consistent.

Similarly to the simulation results, only OBD needs the critical parameter, bin size T, which determines the final energy consumption. Figure 11 shows that there is no general value of bin size for the five workflows, because each flow contains different jobs and is deployed on different size of virtual cluster. Next, we take the black joblist1 as an example. After 11 times of empirical tests, we tune the parameter T to its most efficient value 500, and compare OBD with OTB and BF in Figs. 12 and 13.

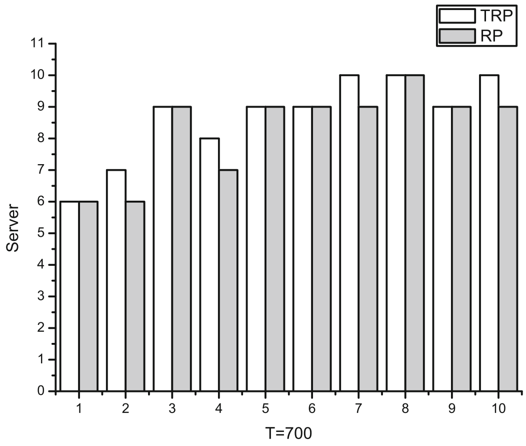
In Fig. 12, these algorithms need nearly the same number of servers to host coming jobs. The reason is that VM type, server capacity and job number on testbed are much smaller than that in simulation cases, and it is easy to obtain the same placement result by different algorithms. There are two humps in the curve, which means that the first five jobs are completed before the sixth job arrives. Notice that OBD needs one fewer than others in the second humps. The reason is that OBD bounds the jobs on one server within $T = 500$, making most of servers host only one job. As a result, servers can hibernate right after job completion.

In Fig. 13, energy consumption is in accordance with the server number results. Although OBD shows better energy-efficiency than OTB and BF, its superiority only occurs when $T \in [500-900]$. If bin size T is out of this range, the energy consumption for OBD is equal to or even greater than that for other heuristics.

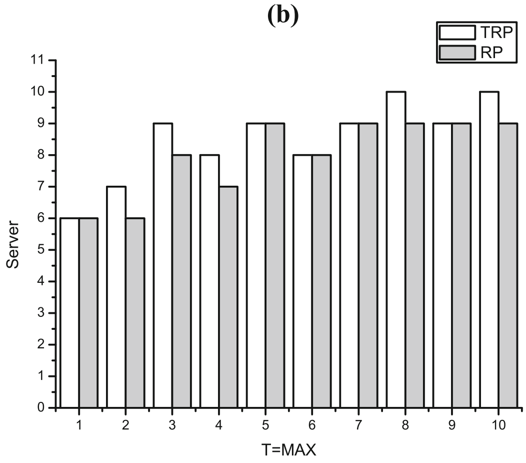
Fig. 9 Sever number for batch consolidation on Hadoop. **a** $T = \text{Min}$. **b** $T = 700$. **c** $T = \text{Max}$



(a)



(b)



(c)

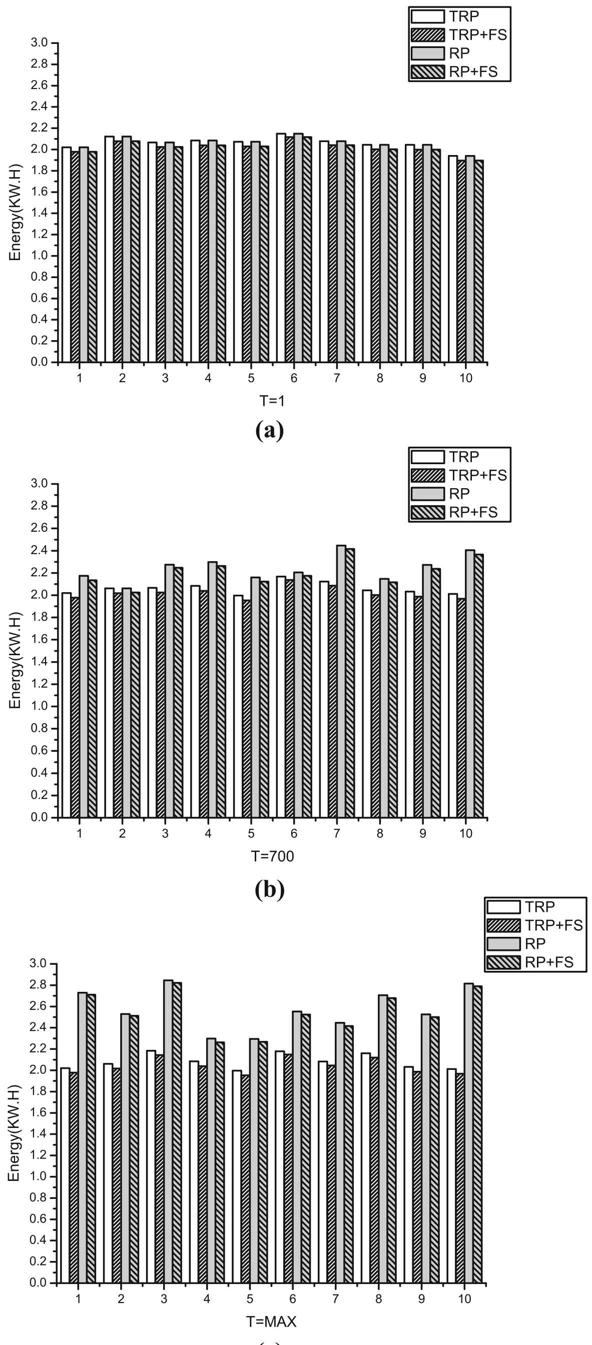


Fig. 10 Energy consumption for batch consolidation on Hadoop. **a** T = Min. **b** $T = 700$. **c** T = Max

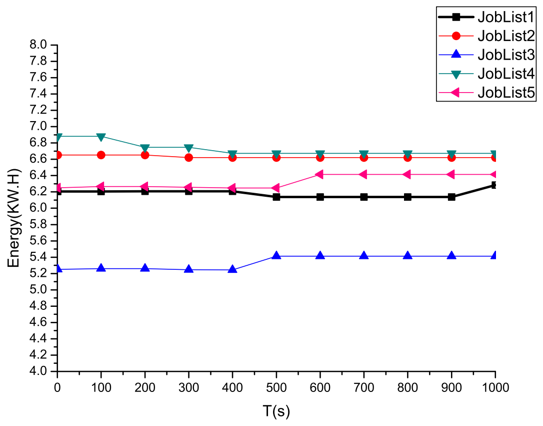


Fig. 11 Energy consumption of OBD on Hadoop

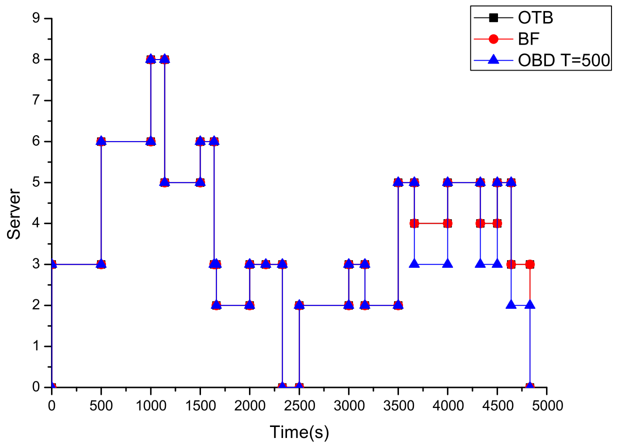


Fig. 12 Server number for online placement on Hadoop

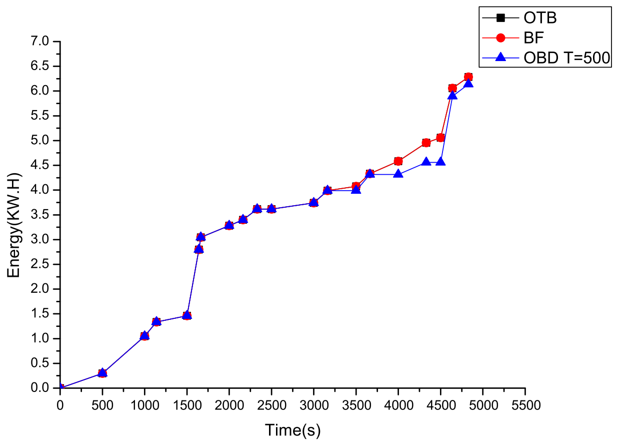


Fig. 13 Energy consumption for online placement on Hadoop

7 Conclusion

This paper discussed the problem of energy efficiency in an IaaS cloud provider that take a virtual Hadoop cluster as a service. An important novelty of the study is that our co-optimization solution includes a cluster-level VM consolidation and a server-level frequency scaling.

Although VM consolidation has been studied by many researchers in the past decades, very few of them consider the consolidation of virtual clusters. We proposed two heuristics, TRP and OTB, for batch-oriented consolidation and on-demand placement, respectively. Both the two algorithms outperformed their competitors, RP and OBD, by reducing energy consumption for a cloud datacenter. The complexity of TRP is lower than that of RP due to a reduced size of recipe list. Moreover, our heuristics are not sensitive to tunable parameters, while RP and OBD have a critical parameter, bin size. The tuning of bin size may result in differences in algorithm effectiveness. In addition, OTB can reduce the mode switching by means of balancing server duration and utilization, and fits for homogeneous and heterogeneous systems.

On the server level, we proposed FS algorithm to minimize the energy consumption by scaling frequency to the most efficient value. From theoretical aspect, we proved a bound to what extent scaling frequency can save energy. FS can be applied on TRP and also can work together with other VM consolidation heuristics. We built the power consumption function through simulation with Sniper and McPAT. The simulation results showed that the power function of multi-core architecture fits in well with our assumption.

At this point, we are interested in other important research directions. First, the power model in this paper is relatively simple, mainly considering energy consumption by processers. We plan to investigate other energy-proportionality to better understand how server consolidation affects individual components within a server, such as memory, hard drives and networking. Furthermore, this paper only considered Hadoop MapReduce (MRV1). Its newest version Yarn (MRV2) has become a general resource management system that supports multiple distributed computing frameworks running on it, such as Spark, Tez, and Storm, and enables users to process data simultaneously across batch, interactive and real-time methods. Thus, our future research will address platform heterogeneity by developing a general framework to improve the energy efficiency of a cloud datacenter when running multiple workloads with different SLAs.

References

1. Akhter N, Othman M (2016) Energy aware resource allocation of cloud data center: review and open issues. Clust Comput 26(1):1–20
2. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gen Comput Syst 28(5):755–768
3. Berral JL, Goiri Í, Nou R, Julià F, Guitart J, Gavaldà R, Torres J (2010) Towards energy-aware scheduling in data centers using machine learning. In: IEEE Proceedings of the International Conference on Energy-Efficient Computing and Networking, pp 215–224
4. Cardosa M, Singh A, Pucha H, Chandra A (2012) Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud. IEEE Trans Comput 61(12):1737–1751

5. Carlson TE, Heirman W, Eyerman S, Hur I, Eeckhout L (2014) An evaluation of high-level mechanistic core models. ACM Trans Archit Code Optim 11(3):1–25
6. Chen Y, Alspaugh S, Borthakur D, Katz R (2012) Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In: ACM Proceedings of the European Conference on computer systems (EuroSys), pp 43–56
7. Chen Y, Das A, Qin W, Sivasubramaniam A, Wang Q, Gautam N (2005) Managing server energy and operational costs in hosting centers. In: ACM Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), pp 303–314
8. Chen Y, Ganapathi A, Katz RH (2010) To compress or not to compress-compute vs. io tradeoffs for mapreduce energy efficiency. In: ACM Proceedings of ACM SIGCOMM workshop on green networking, pp 23–28
9. Deng Q, Meisner D, Ramos L, Wenisch TF, Bianchini R (2011) Memscale: active low-power modes for main memory. ACM SIGARCH Comput Archit News 39(1):225–238
10. Ghamkhari M, Mohsenian-Rad H (2013) Energy and performance management of green data centers: a profit maximization approach. IEEE Trans Smart Grid 4(2):1017–1025
11. Goiri I, Julia F, Nou R, Berral JL, Guitart J, Torres J (2010) Energy-aware scheduling in virtualized datacenters. In: IEEE Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), pp 58–67
12. Govindan S, Choi J, Urgaonkar B, Sivasubramaniam A, Baldini A (2009) Statistical profiling-based techniques for effective power provisioning in data centers. In: ACM Proceedings of European Conference on Computer Systems (EuroSys), pp 317–330
13. Guenter B, Jain N, Williams C (2011) Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In: IEEE INFOCOM, pp 1332–1340
14. Jiang J, Feng Y, Zhao J, Li K (2016) Dataabc: a fast abc based energy-efficient live vm consolidation policy with data-intensive energy evaluation model. Future Gen Comput Syst 87(5):1–33
15. Kaur T, Chana I (2015) Energy efficiency techniques in cloud computing: a survey and taxonomy. ACM Comput Surv 48(2):1–46
16. Kaushik RT, Bhandarkar M (2010) Greenhdfs: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. In: Proceedings of the International Conference on Power Aware Computing and Systems (HotPower), USENIX Association, pp 1–9
17. Khosravi A, Garg SK, Buyya R (2013) Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In: Proceedings of the 19th International Conference on Parallel Processing, pp 317–328
18. Lang W, Patel JM (2010) Energy management for mapreduce clusters. VLDB Endow 3(1):129–139
19. Lee YC, Zomaya AY (2012) Energy efficient utilization of resources in cloud computing systems. J Supercomput 60(2):268–280
20. Leverich J, Kozyrakis C (2010) On the energy inefficiency of hadoop clusters. ACM SIGOPS Oper Syst Rev 44(1):61–65
21. Li P, Guo S, Yu S, Zhuang W (2015) Cross-cloud mapreduce for big data. IEEE Trans Cloud Comput 26(3):1–14
22. Li S, Ahn JH, Strong RD, Brockman JB, Tullsen DM, Jouppi NP (2009) Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Proceedings of the 42nd Annual Symposium on Microarchitecture, pp 469–480
23. Luo J-P, Xia L, Chen M-R (2014) Hybrid shuffled frog leaping algorithm for energy-efficient dynamic consolidation of virtual machines in cloud data centers. Expert Syst Appl 4(2):1–13
24. Mann ZA (2015a) Allocation of virtual machines in cloud data centersa survey of problem models and optimization algorithms. ACM Comput Surv 48(1):1–34
25. Mann ZA (2015b) Rigorous results on the effectiveness of some heuristics for the consolidation of virtual machines in a cloud data center. Future Gen Comput Syst 51(4):1–6
26. Mashayekhy L, Nejad M, Grosu D, Zhang Q, Shi W (2015) Energy-aware scheduling of mapreduce jobs for big data applications. IEEE Trans Parallel Distrib Syst 26(10):2720–2733
27. Meng X, Pappas V, Zhang L (2010) Improving the scalability of data center networks with traffic-aware virtual machine placement. In: IEEE Proceedings of INFOCOM, pp 1–9
28. Quang-Hung N, Thoai N, Son NT (2013) Epobf: energy efficient allocation of virtual machines in high performance computing cloud. J Sci Technol 51(4):173–182
29. Ribas BC, Suguimoto RM, Montaño RANR, Silva F, de Bona L, Castilho MA (2012) On modelling virtual machine consolidation to pseudo-boolean constraints. Springer, Heidelberg

30. Sindelar M, Sitaraman RK, Shenoy P (2011) Sharing-aware algorithms for virtual machine colocation. In: ACM Proceedings of the 23th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp 367–378
31. Song W, Xiao Z, Chen Q, Luo H (2013) Adaptive resource provisioning for the cloud using online bin packing. IEEE Trans Comput 63(11):2647–2660
32. Tang M, Pan S (2014) A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers. Neural Process Lett 1(5):1–11
33. Urgaonkar R, Kozat U, Igarashi K, Neely M (2010) Dynamic resource allocation and power management in virtualized data centers. In: IEEE Proceedings of Network Operations and Management Symposium (NOMS), pp 479–486
34. Van HN, Tran FD, Menaud J-M (2010) Performance and power management for cloud infrastructures. In: IEEE Proceedings of the 3rd International Conference on Cloud Computing (CLOUD), pp 329–336
35. Verma A, Ahuja P, Neogi A (2008) pmapper: power and migration cost aware application placement in virtualized systems. In: Middleware. Springer, pp 243–264
36. Verma A, Cherkasova L, Campbell RH (2011) Aria: automatic resource inference and allocation for mapreduce environments. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp 249–256
37. Von Laszewski G, Wang L, Younge AJ, He X (2009) Power-aware scheduling of virtual machines in dvfs-enabled clusters. In: Proceedings of the IEEE International Conference on Cluster Computing (Clusters), pp 1–10
38. Wang L, Zhang F, Zheng K, Vasilakos AV, Ren S, Liu Z (2014) Energy-efficient flow scheduling and routing with hard deadlines in data center networks. In: IEEE Proceedings of IEEE 34th International Conference on Distributed Computing Systems (ICDCS), pp 1–11
39. Wirtz T, Ge R (2011) Improving mapreduce energy efficiency for computation intensive workloads. In: IEEE Proceedings of the International Green Computing Conference and Workshops (IGCC), pp 1–8
40. Wong D, Annavaram M (2014) Implications of high energy proportional servers on cluster-wide energy proportionality. In: IEEE Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA), pp 142–153
41. Wu Y, Tang M, Fraser W (2012) A simulated annealing algorithm for energy efficient virtual machine placement. In: IEEE Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp 1245–1250
42. Xie R, Jia X, Yang K, Zhang B (2013) Energy saving virtual machine allocation in cloud computing. In: IEEE Proceedings of the 33rd International Conference on Distributed Computing Systems Workshops (ICDCSW), pp 132–137
43. Zhou Z, Liu F, Jin H, Li B, Li B, Jiang H (2013) On arbitrating the power-performance tradeoff in saas clouds. In: IEEE Proceedings of INFOCOM, pp 872–880
44. Zhuo J, Chakrabarti C (2008) Energy-efficient dynamic task scheduling algorithms for dvs systems. ACM Trans Embed Comput Syst 7(2):421–434