Xiao P, Hu ZG, Zhang YP. An energy-aware heuristic scheduling for data-intensive workflows in virtualized datacenters. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(6): 948–961 Nov. 2013. DOI 10.1007/s11390-013-1390-9

An Energy-Aware Heuristic Scheduling for Data-Intensive Workflows in Virtualized Datacenters

Peng Xiao[1,2] (肖 鹏), *Student Member*, *CCF*, *ACM*, *IEEE*, Zhi-Gang Hu[2,*] (胡志刚), *Senior Member*, *CCF* and Yan-Ping Zhang[2,3](张艳平), *Student Member*, *ACM*, *IEEE*

[1] *School of Computer and Communication, Hunan Institute of Engineering, Xiangtan 411104, China*

[2] *School of Information Science and Engineering, Central South University, Changsha 410083, China*

[3] *College of Computation and Bioinformatics, Technical University of Munich, Freising 85354, Germany*

E-mail: {xpeng4623, zghu.csu}@gmail.com; yanping.zhang@wzw.tum.de

Received September 2, 2012; revised May 6, 2013.

Abstract With the development of cloud computing, more and more data-intensive workflows have been deployed on virtualized datacenters. As a result, the energy spent on massive data accessing grows rapidly. In this paper, an energy-aware scheduling algorithm is proposed, which introduces a novel heuristic called Minimal Data-Accessing Energy Path for scheduling data-intensive workflows aiming to reduce the energy consumption of intensive data accessing. Extensive experiments based on both synthetical and real workloads are conducted to investigate the effectiveness and performance of the proposed scheduling approach. The experimental results show that the proposed heuristic scheduling can significantly reduce the energy consumption of storing/retrieving intermediate data generated during the execution of data-intensive workflow. In addition, it exhibits better robustness than existing algorithms when cloud systems are in presence of I/O-intensive workloads.

Keywords cloud computing, energy efficient, heuristic scheduling, data-intensive workflow

1 Introduction

Cloud computing is emerging as a promising distributed computing paradigm providing a highly available, scalable, and flexible computing platform for a variety of applications[1-2]. In cloud environments, high-performance datacenters play an important role in resource management and service provision, and their performance and reliability directly affect the QoS (Quality of Service) satisfaction of cloud users[3]. To maintain desirable QoS, many datacenters tend to equip with advanced IT devices and keep them available in 24 hours[4]. Consequently, the energy consumption of datacenters grows rapidly significantly increasing the operational cost of infrastructure providers[5-6]. Furthermore, the increasing energy consumption in datacenters also raises many other issues, such as reliability[7], security[8], and CO_2 emission problems[9]. Therefore, energy-efficiency management has become a major concern in the design of modern datacenters.

Recently, energy-aware scheduling for workflow applications attracts plenty of attentions[10-12]. Typically, workflow applications can be categorized as computation-intensive and data-intensive. Cloud infrastructures have several advantages over traditional high-performance computing (HPC) systems for executing data-intensive workflows, such as configurable virtual execution environment[13], on-demand resource provision[14], and elastic service capability[15]. However, these advantages also raise many challenging issues when implementing energy-aware scheduling for data-intensive workflows, which are briefly summarized as: 1) Improper intermediate data transferring and moving will result in degraded execution performance as well as high energy consumption[10-11]; 2) VM schedulers often adapt proportional fairness strategy for vCPU allocation[16]. However, high-frequent I/O requests will significantly increase the context-switch overheads and make the proportional fairness strategy inefficient and unpredictable[17-19]; 3) Data-intensive

Regular Paper

Supported by the National Natural Science Foundation of China under Grant Nos. 60970038, 61272148, the Science and Technology Plan Project of Hunan Province of China under Grant No. 2012GK3075, and the Scientific Research Fund of Hunan Provincial Education Department of China under Grant No. 13B015.

*Corresponding Author

workflows are usually interwoven with computing tasks and data accessing tasks. As the energy consumption models of two classes of tasks are quiet different, it is difficult to reduce both two kinds of energy consumption at the same time[20-21].

To address the above issues, in this work we propose a novel scheduling algorithm, which applies a new heuristic called *Minimal Data-Accessing Energy Path* (MDEP) for scheduling data-intensive workflows. The proposed scheduling algorithm consists of two phases: firstly, it uses MDEP heuristic for deploying and configuring virtual machine (VM) instances; secondly, it schedules workflow activities to VM instances according to a novel priority, which is calculated by combining the execution-dependency and the per-VM power model together. In this way, both the execution performance and energy-efficiency are fully taken into consideration in the proposed algorithm. The main contributions of this work are as follows: 1) We present a general energy model for data-intensive workflow applications, which fully takes the energy consumption of data transferring into account. 2) Based on the general energy model, we implement an energy-aware algorithm and present some properties of the proposed algorithm theoretically. 3) By comparing the results with four existing heuristic algorithms, we are convinced that the proposed algorithm is effective to conserve the energy consumption for data-intensive workflows.

The rest of this paper is organized as follows. In Section 2, we summarize the related work. In Section 3, we present the formal definitions on workflow scheduling and the problem description. In Section 4, energy models for both VM instance and workflow scheduling are presented. In Section 5, we analyze the energy-aware scheduling model and present the implementation of our heuristic. In Section 6, extensive experiments are conducted and the results are carefully investigated and evaluated. Finally, Section 7 concludes the paper with a brief discussion of our future work.

2 Related Work

Generally speaking, the problem of workflow scheduling is NP-hard, but some of them still can be solved in polynomial time under certain conditions. For example, Benoit *et al.* proposed an algorithm with polynomial complexity for scheduling pipeline workflow under energy constraint[22]. Although they only theoretically proved that one-to-one or interval-mapping strategy for scheduling pipeline workflows are polynomial time-complexity if the objective function is singleton, their work pointed out a valuable methodology, that is, the structural features of a workflow have significant effects on the complexity of the scheduling algorithm.

In the past few years, many efforts have been taken into designing energy-aware scheduling algorithm for computation-intensive applications[23-26]. As the processors contribute to the major part of the overall energy consumption, most of the scheduling algorithms are based on DVFS (Dynamical Voltage and Frequency Scaling) mechanism[27]. For example, in [24], Rizvandi *et al.* combined the traditional Max-Min algorithm with DVFS mechanism and designed an MMF-DVFS heuristic for precedence-constrained applications. In [25], Lee and Zomaya proposed two energy-conscious scheduling heuristics, in which the scheduling scheme is repeatedly modified until the energy consumption cannot be reduced any longer. In [26], Mezmaza *et al.* designed a bi-objective hybrid scheduling framework, in which makespan and energy consumption are defined as the objective functions, and the underlying scheduling mechanism is based on a genetic algorithm. As most of the existing algorithms assume that the energy spent on data accessing is ignorable, it makes them unsuitable for scheduling data-intensive applications.

Traditionally, studies on conserving the data-accessing related energy consumption mainly concentrate on storage architecture. For example, in [28], Zong *et al.* proposed a buffer-based framework, in which a buffer controller component is responsible for accumulating small-writing operations together and then sends these accumulated data to physical disks. In this way, the data-center can keep a large number of idle disks in sleeping modes as long as possible. In [29], Manzanares *et al.* proposed a pre-fetching buffering technique, which uses the block accessing frequency as a heuristic to configure the corresponding buffer settings. The above studies are effective for saving the energy consumption of storage systems at the physical level. However they are not application-oriented, which means that they are only suitable for coarse-grained energy consumption management instead of fine-grained energy consumption optimization. As noted in [30], more and more cloud providers wish to charge their customers for applications' energy consumption. Therefore, fine-grained energy consumption optimization and control seems to be the mainstream in the future.

Recently, many researchers have taken their efforts into studying the co-relation between energy consumption and application's characteristics. For example, Cho and Melhem presented an excellent theoretical work on the interplay between energy consumption and application structure[31]. In [18], Kang *et al.* studied the performance and energy costs of scheduling MapReduce applications, and proposed an I/O group schedu-

ler, which collectively delegates the I/O requests to the privileged VM instance (dom-0) so as to reduce the extra costs spent on VM context-switches.

Based on the existing studies, our work takes both the application's structural feature and energy consumption model into account when designing the heuristic. Unlike those heuristic schedulings for computation-intensive applications, we do not rely on DVFS mechanism, since the major part of energy consumption of running data-intensive workflows is spent on storing/retrieving a large volume of intermediate data.

3 Problem Description and Definitions

To run workflow applications in a cloud system, many services and middleware are involved as shown in Fig.1. At first, users submit their workflow description files through the submission portal service. Then, a workflow management engine is responsible for translating the abstract workflow into the concrete workflow and mapping their activities onto a set of VM instances which are provided by certain VM hypervisor. The VM hypervisor is responsible for scheduling active VMs on physical resources in coordinated manners.

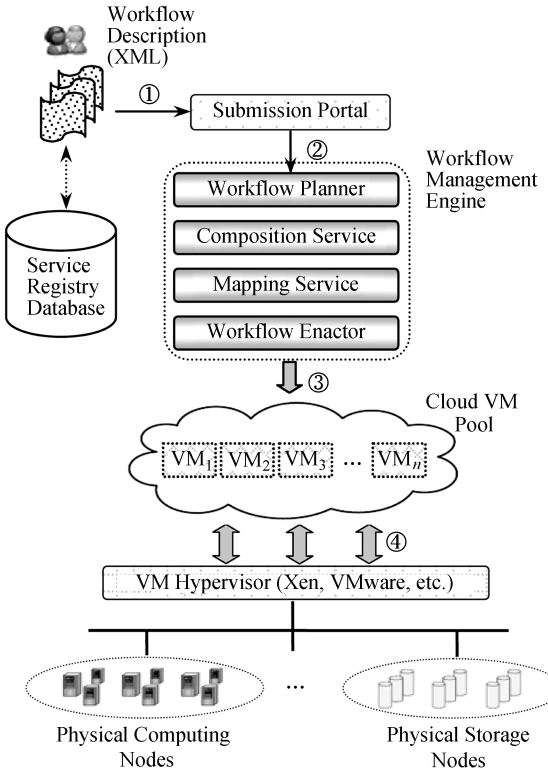


Fig.1. Framework of executing workflow applications in a virtualized environment.

In this work, we mainly concentrate on data-intensive workflows, which have some significant differences from computation-intensive workflows. For example, the intermediate date generated by a computation-intensive workflow is relatively small and can be easily stored in computing nodes' memory or local disks. While a data-intensive workflow will generate a large volume of intermediate data, which requires being stored in independent storage nodes. To illustrate the execution procedure, Fig.2 shows an example of mapping scheme for a data-intensive workflow.

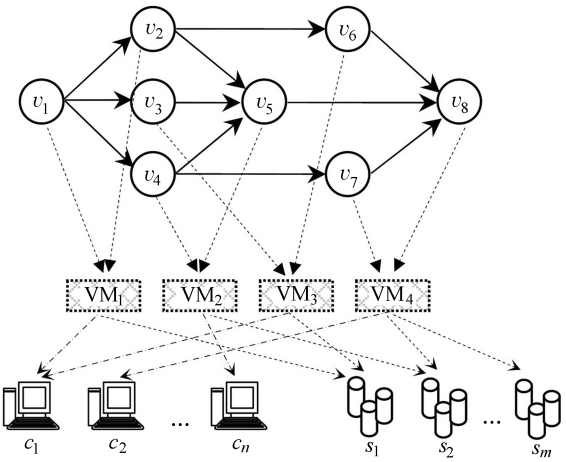


Fig.2. Example of mapping scheme for data-intensive workflow.

Typically, a workflow is represented by a DAG (directed acyclic graph). When running a data-intensive workflow, input data of an activity node is transferred from an independent storage node to the execution node, and output data is transferred back to the original storage node or others. It is clear that not only the computing activities ($v_1 \sim v_8$) should be mapped onto computing resources ($c_1 \sim c_n$), but also the communication edges should be properly mapped onto storage nodes ($s_1 \sim s_m$). Such an interweaving of computing and data-accessing tasks makes it more complex for scheduling data-intensive workflows.

For the convenience of representation in the following sections, we firstly give the related definitions in this section.

Definition 1. *A workflow is noted as a directed acyclic graph $G = (V, W)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of activities representing computing tasks, $W = \{w_{i,j} | if \langle v_i, v_j \rangle \in W\}$ is the set of edges and each $w_{i,j}$ indicates that v_j depends on v_i in terms of data-flow or execution-dependence.*

Definition 2. *Each activity v_i is represented as a 3-triple $\langle a_i, d_i^{\text{in}}, d_i^{\text{out}} \rangle$, where a_i is the size of computing task, d_i^{in} is the size of input data required by v_i, d_i^{out} is the size of output data generated by v_i.*

Definition 3. *A VM instance is noted as $VM_i = \langle F_v, M_v, S_v \rangle$, where F_v is the working frequency of*

virtual processor allocated to VM_i, M_v is the size of virtual memory, S_v is the size of virtual disk configured for VM_i.

In many commercial cloud systems (i.e., Amazon EC2[①]), VM instances are pre-defined by the cloud provider and waiting for users' selection. In this work, we ignore some detailed parameters of VM instances such as OS type and price, since they are irrelevant with our topic. Before scheduling a workflow onto a cloud platform, a set of VM instances should be created and deployed at first. This phase is often called as VM provision or VM deployment[32]. A VM deployment scheme can be considered as a mapping from physical resources to virtual resources defined as follows.

Definition 4. *A VM deployment scheme is noted as $\boldsymbol{D} : C \times S \to \langle F_v, M_v, S_v \rangle$, where $C = \{c_1, c_2, \ldots, c_n\}$ is the set of computing nodes, $S = \{s_1, s_2, \ldots, s_m\}$ is the set of storage nodes.*

For a given VM deployment scheme, its output is a set of VM instances that will be deployed on the cloud platform for executing the target workflow. So, we can formalize the workflow scheduling scheme and its corresponding energy consumption as follows.

Definition 5. *A workflow scheduling scheme is noted as $M : V \times C \times S \to \{0, 1\}$, in which each element $M_{i,i',i''}$ indicates that activity v_i is scheduled on a VM instance whose virtual processor is allocated from computing node $c_{i'}$ and its virtual disk is from $s_{i''}$.*

Definition 6. *For a given scheduling scheme M, the total energy consumption of completing the target workflow G is noted as $E(G, M)$.*

Based on the above definitions, the problem of scheduling data-intensive workflow for optimal energy consumption can be formulized as:

$$\begin{aligned} &\min E(G, M) \\ &\text{s.t. } M : V \times C \times S \to \{0, 1\}, \quad V = \{v_1, v_2, \ldots, v_n\}, \\ &\qquad C = \{c_1, c_2, \ldots, c_m\}, \quad S = \{s_1, s_2, \ldots, s_k\}. \end{aligned}$$

It is clear that the above programming problem is NP-complete since the solution space of M is 2^{n+m+k}. Therefore, a heuristic algorithm seems to be necessary for obtaining a suboptimal solution. Before proposing any heuristics, we firstly need to figure out the approach to modeling the energy consumption of a data-intensive workflow under a given scheduling scheme, that is the formulation for calculating $E(G, M)$.

4 Energy Consumption Model of Data-Intensive Workflows

According to the previous analysis, it can be known that a final scheduling scheme consists of two phases: VM deployment and DAG scheduling. In the phase of VM deployment, physical resources are mapped into a number of VM instances. So, the original power model of a physical machine should be translated into the VM-oriented power model. In the phase of DAG scheduling, activities in the workflow is assigned onto a set of VM instances, therefore the total energy consumption is dependent on the VM power models and the execution time of each VM. So, we first present the per-VM power model, and then give the energy consumption model of a workflow under a given DAG scheduling scheme.

4.1 VM Power Model

The power consumption of machine consists of static part P_{static} and dynamic part P_{dynamic}. P_{static} is the fixed power consumption for keeping the machine in working state even there is no workload on it, while P_{dynamic} is related with the dynamic utilization of power-consuming components. Typically, the power model of a physical machine is formulated as

$$P(t) = P_{\text{static}} + \sum_{j \in \Omega} P_j(t),$$

where $P_j(t)$ is the dynamic power consumption of component j, Ω is the set of power-consuming components, which often consists of CPU, GPU, memory, disk, etc. When a machine is virtualized, its power model can be rewritten as

$$P(t) = P_{\text{static}} + \sum_{i=1}^{m} P_i^{\text{vm}}(t),$$

where $P_i^{\text{vm}}(t)$ is the power model of individual VM instances, m is number of active VM instances on this machine. As the capability of a physical machine is multiplexed by the VM hypervisor among multiple VM instances, the power model of a VM can be formulated as

$$P_i^{\text{vm}}(t) = \frac{P_{\text{static}}}{m} + \sum_{j \in \Omega} [r_j^i \times P_j(t)], \tag{1}$$

where r_j^i is the allocation ratio of component j configured to VM_i.

It is noteworthy that the physical components in a VM instance may come from different physical machines. For example, a number of processors from different machines may be consolidated together for running a computation-intensive application[33]. As our work focuses on data-intensive workflows, without loss of generality, we assume that the computing task of individual activities is relatively small. So, both virtual processors and memory allocated from a same physical

[①]Amazon EC2, http://aws.amazon.com/ec2, Apr. 2013.

machine are sufficient to finish the computing tasks. On the contrary, the virtual disk must be allocated from external storage systems (i.e., Amazon S3② and XtreemFS[34]), which are especially designed for massive and large volume data-accessing.

4.2 Energy Model Under Scheduling Scheme

After the deployment of VM instances, a workflow will be assigned onto these VMs. Given a scheduling scheme M, according to Definition 4, M can be noted as $\{M_{i,i',i''}|i \in (1,\ldots,n),\ i' \in (1,\ldots,m),\ i'' \in (1,\ldots,k)\}$, where i is the index of activities in the workflow, i' is the index of computing nodes, and i'' is the index of storage nodes. Therefore, we can note the energy consumption of completing the activity v_i as $E(v_i, M_{i,i',i''})$.

As shown in Fig.2, part of $E(v_i, M_{i,i',i''})$ is spent on virtual processor and virtual memory, and the other part is spent on virtual disk. In the following, we note them as $E_c(v_i, M_{i,i',i''})$ and $E_d(v_i, M_{i,i',i''})$, respectively. When an activity $v_i = \langle a_i, d_i^{\text{in}}, d_i^{\text{out}}\rangle$ is assigned onto VM_j, its execution time of computing task can be noted as

$$T_{\text{exec}} = \frac{a_i}{r_{\text{cpu}}^j \times F_v}, \tag{2}$$

where r_{cpu}^j is the allocation ratio of CPU configured to v_i. Combining (1) and (2), $E_c(v_i, M_{i,i',i''})$ can be measured as

$$E_c(v_i, M_{i,i',i''}) = P_j^{\text{vm}}(t) \times \frac{a_i}{r_{\text{cpu}}^j \times F_v}. \tag{3}$$

As the virtual disk is allocated from an external storage system, the disk component in (1) will be automatically ignored. To calculate the energy consumption spent on data accessing, we must take into account the location of the storage node as well as the structure of the workflow. Based on the illustration in Fig.2, we can have the formulation of $E_d(v_i, M_{i,i',i''})$ as below.

$$E_d(v_i, M_{i,i',i''}) = \sum_{j \in Pred(v_i)} \left(P_{\text{disk}}^{j''}(t)\frac{d_{j\to i}^{\text{in}}}{B_{i',j''}}\right) + \sum_{k \in Succ(v_i)} \left(P_{\text{disk}}^{i''}(t)\frac{d_{i\to k}^{\text{out}}}{B_{i',i''}}\right), \tag{4}$$

where $P_{\text{disk}}^k(t)$ is the power model of storage node k, $B_{i,j}$ is the bandwidth between storage node i and computing node j, $d_{j\to i}^{\text{in}}$ is the input data from v_j to v_i, $d_{i\to k}^{\text{out}}$ is the output data from v_i to v_k, $Pred(v_i)$ and $Succ(v_i)$ are the predecessors and successors of v_i respectively.

It is clear that the first part of (4) is the energy consumption spent on obtaining input data before running v_i, and the second part is the energy consumption spent on storing output data after finishing v_i. Combining (3) and (4), we can obtain the total energy consumption under a given scheduling scheme shown as

$$E(G, M) = \sum\nolimits_{i=1}^{n} [E_c(v_i, M_{i,i',i''}) + E_d(v_i, M_{i,i',i''})]. \tag{5}$$

5 Scheduling Model and Algorithm

5.1 Scheduling Model

When scheduling a DAG application in conventional HPC systems, the earliest start time (EST) is the key metric for many scheduling algorithms. It is defined as the earliest start time of activity v_i if it is scheduled on processor p_j, which is calculated by

$$EST(v_i) = \begin{cases} 0, & \text{if } v_i = v_{\text{init}}, \\ \max\limits_{v_k \in Pred(v_i)} \{EFT(v_k) + w_{k,i}\}, & \text{otherwise}, \end{cases} \tag{6}$$

where $w_{k,i}$ is the cost of data transferring from v_k to v_i, $EFT(v_k)$ is the earliest finishing time of v_k. Therefore, a scheduling algorithm that aims to obtain optimal makespan is to minimize $EST(v_{\text{exit}}) + T_{\text{exec}}(v_{\text{exit}}, p_j)$, where $T_{\text{exec}}(v_i, p_j)$ is the execution time of v_i on processor p_j.

Various kinds of algorithms have proposed their own heuristics. For instance, HEFT[35] assigns each activity with an upward rank called $rank_u(v_i)$, which is defined as the longest distance from v_i to v_{exit} including the computation cost of v_i. Then, it sorts all activities in decreasing order of $rank_u(v_i)$ and assigns each activity with a processor which can minimize $EST(v_i, p_j) + T_{\text{exec}}(v_i, p_j)$.

In virtualized cloud systems, we can still adapt metric EST for workflow scheduling. However, some modifications of (6) should be made if the data-intensive feature is taken into account, which is shown as:

$$EFT(M_{i,i',i''}) = \begin{cases} 0, & \text{if } v_i = v_{\text{init}}, \\ \max\limits_{v_j \in Pred(v_i)} \Big\{EFT(M_{i,i',i''}) + \\ \quad \dfrac{d_{j'\to j''}^{\text{out}}}{B_{j',j''}} + \dfrac{d_{j''\to i'}^{\text{in}}}{B_{j'',i'}}\Big\}, & \text{otherwise}, \end{cases} \tag{7}$$

where $d_{j'\to j''}^{\text{out}}/B_{j',j''}$ is the cost of output data transferring from the computing node of v_j (one of the predecessors of v_i) to its intermediate storage node, $d_{j''\to i'}^{\text{in}}/B_{j'',i'}$ is the cost of input data transferring from v_j's intermediate storage node to the computing node allocated to v_i.

Clearly, the key difference between (6) and (7) is that the intermediate data generated by a data-intensive

②Amazon S3, http://aws.amazon.com/s3, Apr. 2013.

workflow should be accessed through independent storage nodes instead of being directly transferred between computing nodes. Although such a difference seems slight, it will have significant effects on the final performance of a scheduling scheme, including makespan and energy consumption. For example, considering a system with three computing nodes (c_1, c_2 and c_3) and two storage nodes (s_1 and s_2). Assuming s_1 is connected with the computing nodes through 100M network, while s_2 uses 1G network. To schedule a data-intensive workflow (shown in Fig.3), three VM instances ($VM_1 \sim VM_3$) are created by two different VM deployment schemes (DS_1 and DS_2), which are shown in Table 1. The only difference between the two schemes is that the first scheme VM_1 uses s_1 as its underlying storage node, while all VM instances use s_2 as their underlying storage node in the second scheme.

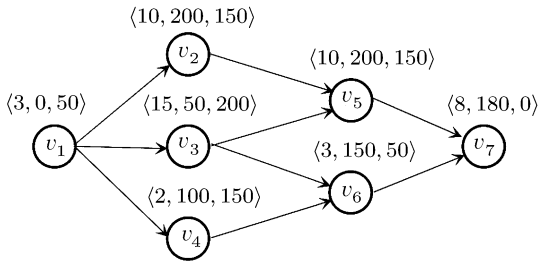


Fig.3. Example of data-intensive workflow.

Table 1. Two VM Deployment Schemes

VM Deploying Scheme	VM_1	VM_2	VM_3
DS_1	$\langle c_1, s_1\rangle$	$\langle c_2, s_2\rangle$	$\langle c_3, s_2\rangle$
DS_2	$\langle c_1, s_2\rangle$	$\langle c_2, s_2\rangle$	$\langle c_3, s_2\rangle$

When using HEFT algorithm scheduling this workflow under the two deployment schemes, we can have two different execution flowcharts as shown in Fig.4. It is clear that when a VM instance uses s_1 as its underlying storage node, then all activities assigned to this VM instance will suffer from longer data transferring time than other activities using s_2. It is noteworthy that both v_5 and v_6 require additional input data transferring when using DS_1 deployment scheme, while using DS_2 they do not. It is because that their predecessor v_3 is assigned to VM_1 whose storage node is s_1 under DS_1, while under DS_2 all VM instances use s_2 as their underlying storage nodes. More importantly, this will result in significant difference in energy consumption.

Motivated by the above observations, in this work we propose a novel heuristic, namely Minimized Energy Consumption in Deployment and Scheduling (MECDS), for data-intensive workflows in virtualized cloud systems. The MECDS mainly consists of two phases: VM deployment and workflow scheduling.

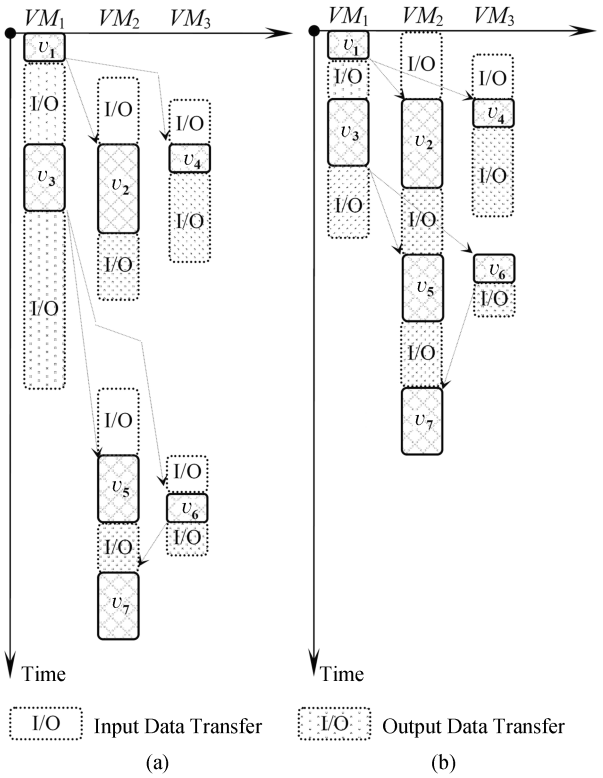


Fig.4. Workflow execution flowcharts under (a) DS_1. (b) DS_2.

In the phase of VM deployment, we select a storage node which can be allocated to a VM instance aiming to obtain minimal data accessing energy consumption for the current activities. To do this, we introduce a novel conception, called Minimal Data-Accessing Energy Path (MDEP), which is defined as the minimal total energy consumption from v_{init} to the current activity shown as:

$$MDEP(v_i) = \begin{cases} E_d(v_i, M_{i,i',i''}), & \text{if } v_i = v_{\text{init}}, \\ E_d(v_i, M_{i,i',i''}) + \\ \quad \min\limits_{v_j \in Pred(v_i)} \{\text{MDEP}(v_j)\}, & \text{otherwise.} \end{cases} \tag{8}$$

According to the definition of MDEP, if a storage node $s_{i''}$ can satisfy $\min\{MDEP(v_i)\}$ among all storage nodes, then the activity v_i should use it as the storage node. If a VM instance that uses $s_{i''}$ as the underlying storage node has already been created and deployed, then we can go on; otherwise a new VM instance should be created and deployed, which uses $s_{i''}$ as the underlying storage node. By repeating the above, we can complete the deployment of VM instances.

In the phase of workflow scheduling, we define the priorities of VM instances as:

$$rank(VM_i) = MDEP(v_{\text{exit}}) + E_c(v_{\text{exit}}, VM_i). \tag{9}$$

So, the rank of VM_i indicates the minimal data-accessing energy consumption of v_{exit} if it is assigned to VM_i. Here, we take into account the computing energy consumption $E_c(v_{\text{exit}}, VM_i)$ in case two VM instances are configured with the same storage node and different computing nodes. All VMs are sorted in the ascendant order of $rank(VM_i)$. As for the activity priority of activities, we directly use the classic b-level rank for the convenience of implementation, which is defined as

$$rank(v_i) = \max_{v_j \in Pred(v_i)} \{rank(v_j)\} + \frac{a_i}{r^i_{\text{cpu}} F_v}. \quad (10)$$

Based on the above scheduling model, the detailed implementation of the heuristic algorithm is shown as Algorithm 1.

Algorithm 1. MECDS

Input: original DAG: $G = (V, W)$, bandwidth matrix: $\boldsymbol{B}$, power model of all physical nodes.

Begin

1. $vset := \{\}$;
2. $k := 0$;
3. **for each** v_i **do**
4. Find an s_j that satisfies minimized $MDEP(v_i)$;
5. **if** exists a VM that uses s_i **then**
6. **continue**;
7. **end if**
8. $k := k + 1$;
9. Create VM_k and add it to *vset*;
10. **end for**
11. Sort all VMs in *vset* in the ascendant order of $rank(VM_i)$;
12. Compute $rank(v_i)$ for all activities by traversing G from v_{exit} to v_{init};
13. Sort the activities in a scheduling list by non-increasing order of $rank(v_i)$;
14. **while** there are unscheduled activities **do**
15. Select the first task v_i from the list for scheduling;
16. **for each** VM_j in $\{VM_1, \ldots, VM_k\}$ **do**
17. Compute $EST(M_{i,i',i''}) + T_{\text{exec}}(v_i)$;
18. **end for**
19. Assign v_i to $VM_{i''}$ that minimizes $EST(M_{i,i',i''}) + T_{\text{exec}}(v_i)$ by insert scheduling;
20. **if** $s_v < d_i^{\text{out}}$ **then**
21. Reconfigure the virtual disk of $VM_{i''}$ to meet the requirement of v_i;
22. **end if**
23. **end while**

End

In the MECDS implementation, step 3 $\sim$ step 10 are to create and development VM instances for the target workflow; step 11 $\sim$ step 23 are to schedule workflow onto the set of VM instances. In step 21, a VM reconfiguration operation is performed because we cannot know the exact virtual disk size at the time of VM creation (in step 9).

5.2 Analysis of MECDS Algorithm

We first analyze the time complexity of the MECDS algorithm. Then, the major properties of MECDS will be presented theoretically.

Theorem 1. *The time complexity of MECDS is $O(n \times s + n \times \log(n) + n \times \log(s))$, where n is the number of activities, s is the number of storage nodes.*

Proof. There are two main phases in the implementation of MECDS. The first phase is VM configuration and deployment (as shown in step 3 $\sim$ step 10). In this phase, MECDS traverses all activities in the target workflow. In each loop, a storage node s_i that satisfies $\min\{MDEP(v_i)\}$ should be selected out as a candidate. As shown in (8) that $MDEP(v_i)$ is not a linear function of any parameter of a storage node, binary-searching algorithm is not applicable here. So, it takes $O(s)$ time to do this work in the worst case, and the average time-complexity is $O((s-1)/2)$, where s is the number of storage nodes. Therefore, the total average time-complexity of the first phase is $O(n \times (s-1)/2)$, where n is the number of activities in the workflow.

In the second phase, according to (9) MECDS first sorts all VM instances in the ascendant order of $rank(VM_i)$ (step 11), which will take $O(\log|vset|)$ time to do this, where $|vset|$ is the number of VM instances. In steps 12 $\sim$ 13, the ranks of all activities should be calculated according to (10) and sorted in non-increasing order, which will take $O(n + n \times \log(n))$ time to do this. In the following while-loop, the algorithm traverses the unscheduled list and assigns them to suitable VM instances. It takes $O(n \times \log|vset|)$ time to do this.

Summarizing the above analysis, we can know that the total time-complexity of MECDS is $O(n \times s + \log|vset| + n + n \times \log(n) + n \times \log|vset|)$. As $1 \leqslant |vset| \leqslant s$, it means $O(1) \leqslant O(\log|vset|) \leqslant O(\log(s))$. Therefore, we can simply note the time complexity of MECDS as $O(n \times s + n \times \log(n) + n \times \log(s))$. □

Theorem 2. *If $\forall v_i \in V$ satisfying the following condition*

$$E_c(v_i, M_{i,i',i''}) \ll E_d(v_i, M_{i,i',i''}),$$

then, the MECDS algorithm can obtain the scheduling scheme M with minimized $E(G, M)$.

Proof. The condition means that for all activities the energy spent on computing tasks is ignorable. By (5) we can have

$$E(G, M) \approx \sum_{i=1}^{n} E_d(v_i, M_{i,i',i''}).$$

According to (4), $E_d(v_i, M_{i,i',i''})$ only depends on the selection of its underlying storage node for a given DAG. Meanwhile, step 4 in the MECDS algorithm will select those storage nodes that satisfy minimized $MDEP(v_i)$ for the following VM configuration. By the definition of $MDEP(v_i)$, we can know that for $\forall v_i \in V$ there exists at least one VM instance that can minimize $E_d(v_i, M_{i,i',i''})$. Taken v_{init} as an example, the definitions of $MDEP(v_{\text{init}})$ and $E_d(v_i, M_{i,i',i''})$ are identical, so the VM instance that can satisfy $\min\{E_d(v_i, M_{i,i',i''})\}$ is the one selected out in step 4. If there are several VMs satisfying $\min\{E_d(v_i, M_{i,i',i''}))$, we only need to select one that satisfies the precedent-constraint of the DAG. This is done by step 16 $\sim$ step 19 in MECDS, which uses inserting strategy to schedule the current activity. □

Theorem 2 indicates that the MECDS algorithm is effective to reduce the energy consumption of data-intensive workflow, whose data-accessing energy consumption contributes most to the total energy consumption.

Theorem 3. *If all the storage nodes have the identical power model, and the condition in Theorem* 2 *is still satisfied, the MECDS algorithm can obtain the scheduling scheme M with minimized makespan.*

Proof. By (4), we can know that $E_d(v_i, M_{i,i',i''})$ is dependent on the storage node's power model $P^i_{\text{disk}}(t)$. If all the storage nodes have the identical power model, for $\forall v_i \in V$, $E_d(v_i, M_{i,i',i''})$ will be strictly linear to the intermediate data transferring time $(d^{\text{in}}_{j\to i}/B_{i',j''} + d^{\text{in}}_{i\to k}/B_{i',j''})$. According to the analysis and conclusion of Theorem 2, the MECDS can minimize $E_d(v_i, M_{i,i',i''})$ for all activities. Therefore, it can minimize the intermediate data transferring time, which in turn minimizes the makespan. □

Theorem 3 indicates that the MECDS algorithm can improve the execution performance for data-intensive workflows. It is noteworthy that the conditions assumed in Theorem 3 are sufficient instead of necessary.

6 Experiments and Evaluation

We conduct two groups of experiments to investigate the performance of the proposed algorithm. The first group of experiments is performed on an extended version of CloudSim[36] by using synthetical DAG workloads. The second group of experiments is conducted on a real cloud testbed and uses a real world workflow as workload. To compare the performance of MECDS algorithm with others, we adapt four other existing scheduling algorithms including HEFT[35], MMF-DVFS[24], ECS+idle[25], and EADAGS[23].

6.1 Simulative Experiments

In simulative experiments, we mainly focus on the effects of data-intensive characteristic on workflow energy consumption and execution performance (makespan). The flowchart of the target workflow is shown in Fig.5, which has two features: firstly, it has massive paralleling subtasks which generate a large volume of intermediate data; secondly, its level-structure is quiet distinguishing, which enables us to exploit the potential of a cloud system. Such a workflow paradigm can be seen in many data-intensive applications, such as MapReduce applications[37] and parameter sweep applications[38].

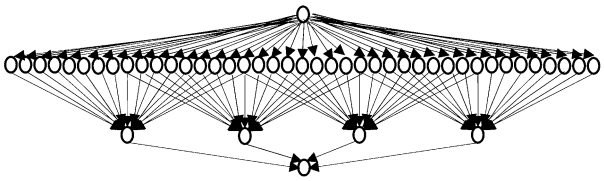


Fig.5. Flowchart of simulative workflow.

For the convenience of conducting experiments, we set all the computing nodes and the storage nodes fully connected with various bandwidths randomly distributed from 10 MB to 1 GB. We use the communication computation ratio (CCR) metric as the indicator of data-intensive degree. A higher CCR value means that the workflow is more data-intensive. The experiment is conducted six times, each with a different CCR value ranging from 0.1 to 10.0 gradually. To distinguish the energy spent by different components (i.e., vCPU, vMem, vDisk), we separately record them in all cases, and the experimental results are shown in Fig.6(a)$\sim$Fig.6(f).

As shown in Fig.6(a), when CCR is in low level, we notice that processors contribute most to the total energy consumption regardless of the used scheduling algorithm. In this case, HEFT performs the worst among all the five algorithms, while ECS+idle obtains about 47% energy saving compared with the former. The reason is that ECS+idle is specially designed for computation-intensive applications, which applies DVFS mechanism to reduce the processor energy consumption. As mentioned in Section 2, both MMF-DVFS and EADAGS use DVFS for scheduling DAG applications, so we can see that their processor related energy consumptions are also very low just like ECS+idle. The difference between those three DVFS-based algorithms is that MMF-DVFS seems more unstable than the other two, since the deviation of its vCPU energy

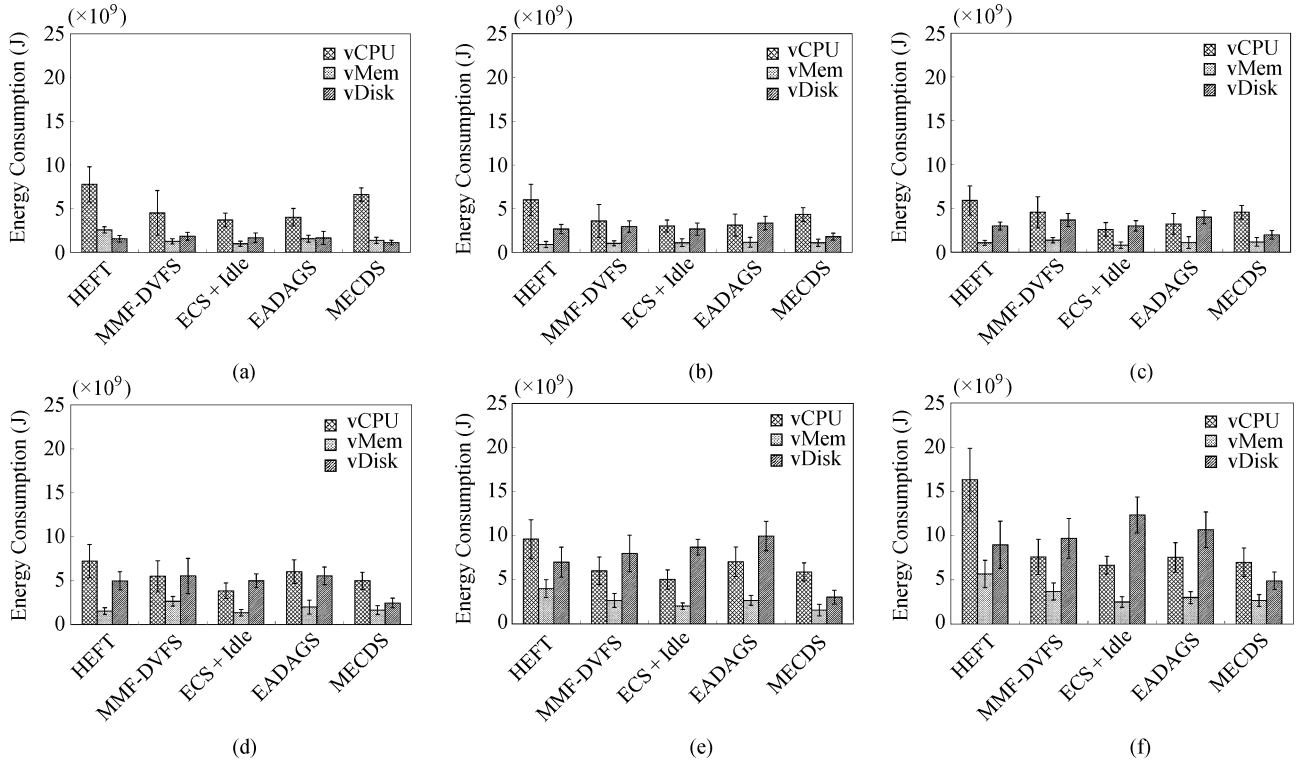


Fig.6. Energy consumption and distribution for different CCRs. (a) CCR = 0.1. (b) CCR = 0.5. (c) CCR = 1.0. (d) CCR = 2.0. (e) CCR = 5.0. (f) CCR = 10.0.

consumption is very high. The reason is that MMF-DVFS uses coarse-grained DVFS adjusting strategy which considers energy consumption minimization as the only objective, while ECS+idle and EADAGS use fine-grained DVFS adjusting strategy which tries to make a tradeoff between makespan and energy consumption. For example, ECS+idle algorithm recursively adjusts the working frequency according to a relative superiority (RS) metric. As to the MECDS, its vCPU energy consumption is slightly lower (about 9%) than that of HEFT, but significantly higher than other three algorithms. It is because that MECDS does not use DVFS mechanism to reduce the processor energy consumption.

When CCR value is in moderate level (as shown in Fig.6(b)∼Fig.6(d)), the vCPU energy consumptions of all algorithms are reduced at first and then are increased. Taking ECS+idle algorithm as an example, its vCPU energy consumption is reduced by about 33% when CCR increases from 0.1 to 1.0, and then is increased by about 24% when we further increases CCR to 2.0. To explain this, we show the mean makespan of all experiments in Fig.7.

It is clear that the mean makespan has the same trend just like energy consumption. The main reason is that the total energy consumption of an application

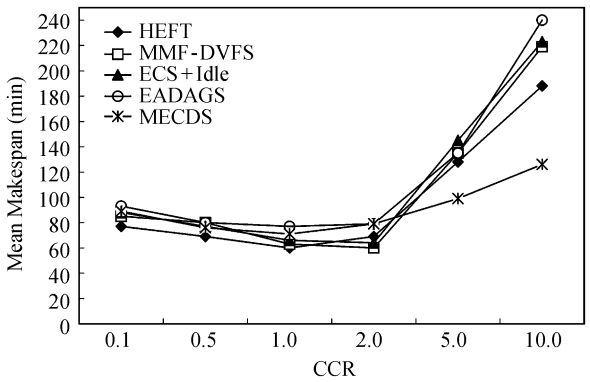


Fig.7. Mean makespan for different CCR values.

is linearly proportional to its execution time. Therefore, longer makespan often results in higher energy consumption. So, the important point is that how the CCR metric affects the execution time. In the simulative experiments, when adjusting CCR in low level, we reduce the size of computing tasks so as to increase the CCR value. While the CCR is in moderate level, we increase the size of input/output data to increase the CCR value. Therefore, we can see that the makespan is reduced at first and then is increased, which in turn results in corresponding change of energy consumption.

According to the results shown in Fig.6 and Fig.7, CCR=2.0 seems to be the turning point for both makespan and energy consumption. When CCR value is bigger than 2.0, vDisk energy consumption is dramatically increased (as shown in Fig.6(e)∼Fig.6(f)). Therefore, we can know that the efficiency of storage nodes has become the performance bottleneck when CCR increases to a high level, not only for the energy consumption but also for the makespan. In this situation, MECDS significantly outperforms other algorithms in terms of both metrics. As shown in the implementation of MECDS, the phase of VM deployment plays an important role in saving data accessing energy. In this phase, we use MDEP as the heuristic for VM configuration. It is noteworthy that we try to create VM instances as less as possible, and let each VM instance be configured with a different storage node. In this way, several parallel tasks can be mapped onto the same VM instance so as to maximize the utilization of high-performance storage nodes.

Although DVFS mechanism is effective for saving processor energy consumption, its energy savings still cannot compensate the energy wastage caused by prolonged execution time. More specifically, many processors still consume a lot of energy when they are waiting for the completion of massive data accessing, even their working frequency has been switched to the lowest level. For example, the vCPU energy consumption of ECS+idle increases about 74% when the CCR is increased from 2.0 to 10.0. We can easily guess that a great deal of vCPU energy is wasted on I/O waiting. Unfortunately, CloudSim does not provide any facilities for differentiating the wasted energy from the total energy. So, we have to evaluate that in the real platform experiment, which will be presented in the next subsection.

6.2 Experiments on Practical Workflow

The real world experiments are conducted on the cloud platform deployed in our HP high-performance network center. The platform consists of 20 computing nodes ($c_1 \sim c_{20}$) and 7 storage nodes ($s_1 \sim s_7$) as underlying physical resources, which are virtualized by using XCP③. To take into account the heterogeneity, we adopt various kinds of equipments made by different vendors, and the detailed parameters of these nodes are shown in Table 2 and Table 3. During the experiments, we use the Oprofile toolkit④ to log the energy consumption related events, and adopt the VM power model proposed in [39] to measure the per-VM energy consumption.

The target application we select is the well-known INVMOD workflow[40], which is designed to study the effects of climatic changes on the water balance. The basic framework of INVMOD workflow is shown in Fig.8, which mainly consists of two parallel processing levels. Each task in the two-level parallel processing (iWasimRunA and iWasimRunB) will transfer its input data from storage nodes, and the intermediate data should be stored temporarily.

Unlike the simulative experiment, we cannot figure out the CCR value of the INVMOD workflow before its execution, because the size of transferred data is dynamically changed. To solve this problem, we use the pre-defined iteration counter (noted as n) to represent the size of the INVMOD, which is gradually increased from 10 to 50. According to our estimation after experi-

Table 2. Hardware Parameters of Computing Nodes

Node ID	Processor Architecture	CPU Speed (GHz)	Core Voltage (V)	Power Consumption (W)	Memory (GB)
$c_1 \sim c_8$	Intel Xeon E5606 Quad-core	4×2.13	$0.75 \sim 1.35$	Idle: > 280 Peak: 450	16
$c_9 \sim c_{20}$	AMD A4-3400 Dual-core	2×2.7	$0.91 \sim 1.41$	Idle: > 120 Peak: 200	2

Table 3. Hardware Parameters of Storage Nodes

Node ID	Storage Framework	Total Capacity (GB)	Total Power Consumption (W)	Interface Speed (MB)	Disk Cache (MB)
$s_1 \sim s_2$	IBM Ultrastar RAID	5 400	Idle: > 550 Peak: 1280	$800 \sim 1\,200$	32
$s_3 \sim s_7$	WDC iSCSI Sever	250	Idle: > 180 Peak: 250	$200 \sim 350$	8

③XCP, http://www.xen.org/download/xcp/index_1.1.0.html, Apr. 2013.

④Oprofile toolkit, http://oprofile.sourceforge.net, Apr. 2013.

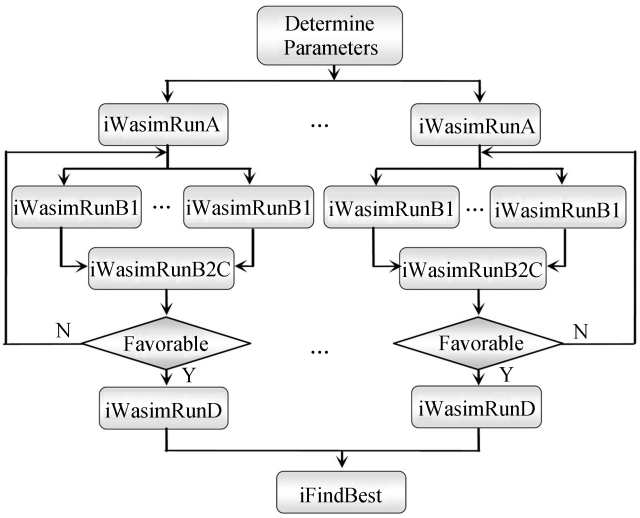


Fig.8. Framework of INVMOD application.

ments, the CCR value is about $5.0 \sim 7.0$ when iteration counter $n = 10$, and it is increased to about $43 \sim 60$ when $n = 50$. As a result, we can see from Fig.9 that vDisk energy consumption is significantly higher than other kinds of energy consumption.

When $n = 10$ and $n = 20$, HEFT and MMF-DVFS seem to perform worst in all the five algorithms in term of total energy consumption. The difference between them is the energy consumption distribution. More specifically, HEFT costs more energy on vCPU, while MMF-DVFS costs more energy on vDisk. The reason is the same as mentioned in the simulative experiments. However, we find that the vDisk energy consumption of MMF-DVFS and ECS+idle increase more quickly than HEFT and EADAGS when $n > 20$. This is different from our simulative results.

To find out the reason, we investigate the logs of scheduling procedure during our experiments. The logs indicate that HEFT algorithm tends to assign tasks onto those VM instances configured with higher computing capability (i.e., $c_1 \sim c_8$), while MMF-DVFS and ECS+idle tend to map tasks onto the VMs configured with more power-efficient computing nodes. In common sense, MMF-DVFS and ECS+idle should perform more energy-efficiently. However, the structural characteristic plays a more important role in this case. As shown in Fig.8, when $n = 10$ it means that the loop will be executed 10 times at most. So, increasing n value will significantly prolong the execution time of solving sub-problems. More importantly, each loop has another parallel branch, in which all the tasks require transferring data. Simply speaking, the performance bottleneck of running INVMOD is the execution of this inside loop. When n is increasing, this bottleneck becomes more and more significant. As HEFT always tends to use powerful computing resources, which enables it to reduce the overall makespan of INVMOD and in turn reduces the energy consumption. On the contrary, MMF-DVFS and ECS+idle do not notice this application-specific feature. Although they can reduce

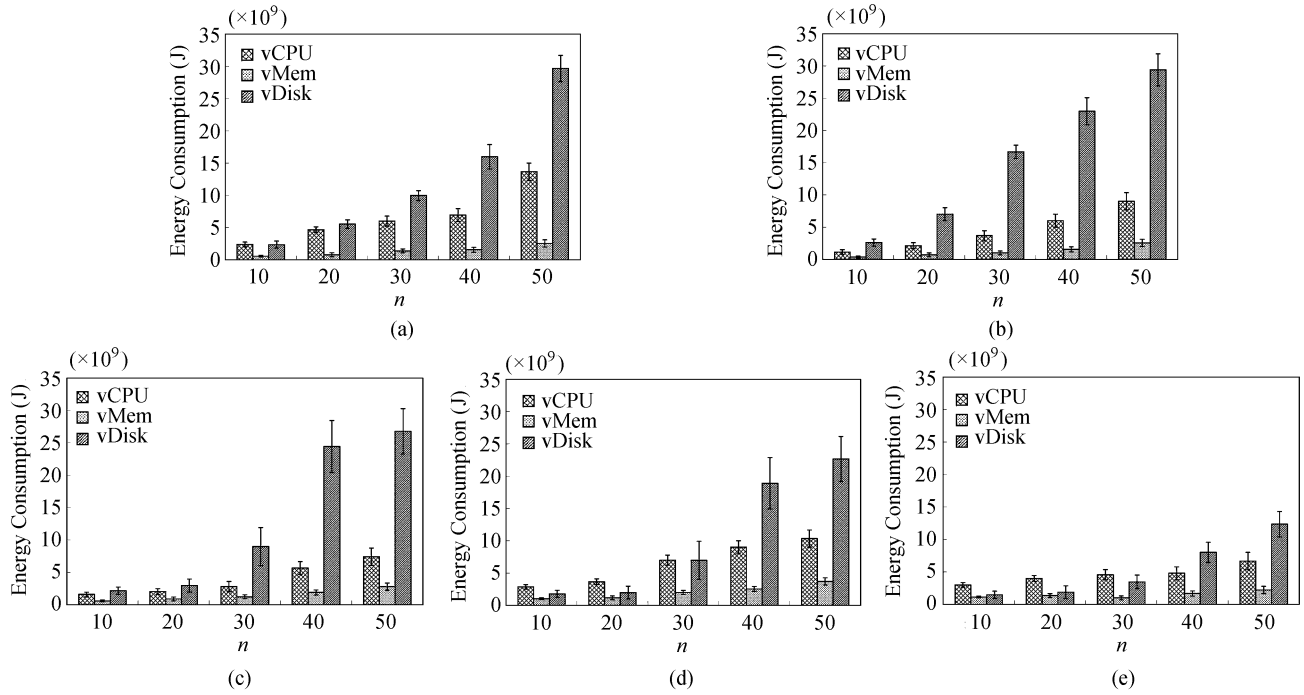


Fig.9. Comparison of energy consumption and distribution with different scheduling algorithms. (a) HEFT. (b) MMF-DVFS. (c) ECS+idle. (d) EADAGS. (e) MECDS.

the vCPU energy consumption, it still cannot compensate the energy wastage spent by idle storage nodes.

Like HEFT, our MECDS also is effective to improve the execution efficiency of the inside loop. However, its strategy is to improve the data transferring efficiency of the parallel branch, which is quite different from HEFT. More importantly, this strategy seems more robust than HEFT. As shown in Fig.9(a), when $n = 50$ HEFT and MMF-DVFS perform the worst in term of vDisk energy consumption. As mentioned in Subsection 6.1, the overall energy consumption metric may be confusing, since we cannot tell how much energy is effectively used and how much is wasted. To further investigate the energy-efficiency, we introduce three metrics here:

• *Effective Computing Energy Consumption* (*ECEC*): total energy spent on processor and memory, subtracting the energy spent at idle state.

• *Effective Data Accessing Energy Consumption* (*EDAEC*): total energy spent by all storage nodes, subtracting the energy spent at idle state.

• *Ineffective Energy Consumption* (*IEEC*): subtracting both ECEC and EDAEC from the total energy consumption.

In this experiment, we record all the three metrics in all cases. For the limitation of space, we only present the experiment results when $n = 50$, since it is most representative for the data-intensive topic. The results are shown in Fig.10, and all the metrics are converted into percentage form for clear representation.

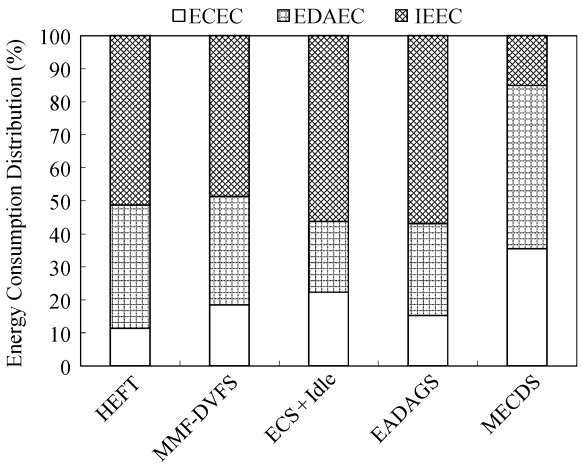


Fig.10. Energy-efficiency metrics for different scheduling algorithms ($n = 50$).

We can see that all the IEEC measurements of the other four algorithms are more than 48%, which means that about half of the energy is wasted when running INVMOD application on our cloud platform. It is clear that MECDS outperforms other algorithms because of its high effective energy consumption for both computing nodes and storage nodes. It is noteworthy that MECDS is not aiming at optimizing the computing related energy consumption. So, its high ECEC measurement comes from the reduced execution time, which in turn reduces the total computing related energy consumption. Among the other four algorithms, HEFT has the lowest ECEC and the highest EDAEC. It is because the high-performance computing nodes tend to waste more energy if they are kept in low utilization for a long time. Relatively, its EDAEC measurement is increased as a result.

Among the five algorithms, only EADAGS and MECDS are designed for data-intensive workflows. When $n = 10$, 20 and 30, the performance difference between the two algorithms is not very distinguishing. When n is bigger than 30, the vDisk energy consumption of EADAGS increases significantly. As noted in [23], EADAGS also uses DVFS mechanism for energy conservation. However, it does not directly aim at reducing the processor related energy consumption. On the contrary, it uses metric CCR as its objective function which is dynamically adjusted through using DVFS mechanism. This strategy is difficult to analyze theoretically. So, we record its summing power of all storage nodes in real-time fashion, and compare it with MECDS.

As shown in Fig.11, the first peak of data accessing incurs at the beginning of transferring input data for the tasks (iWasimRunB). In this phase, the two algorithms perform very likely. The significant difference incurs when some of the sub-problems have been solved and the intermediate data should be stored. When using MECDS, we find that only a few storage nodes are power-active in this phase. More important, they are connected with computing nodes through high-speed bandwidth. So MECDS can complete the intermediate

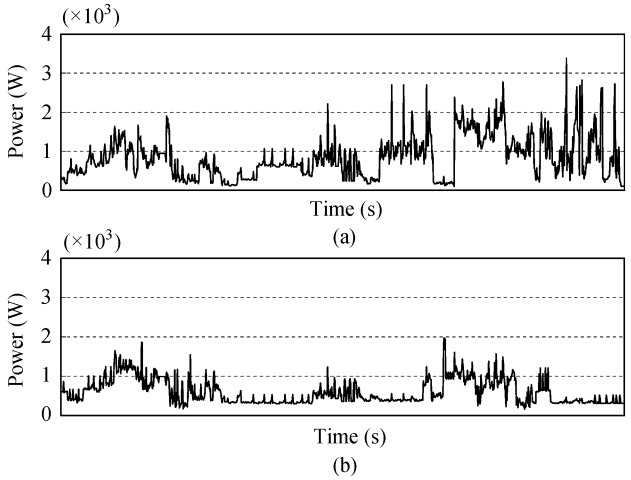


Fig.11. Real-time summing power of all storage nodes (sample interval = 10 seconds). (a) EADAGS. (b) MECDS.

data transferring in shorter period and less energy consumption. When using EADAGS, many low-bandwidth storage nodes are used for those data-intensive tasks. This strategy leads the storage power to frequently reach a peak whether for intermediate data transferring or for the final results collection, as shown at the right-hand of Fig.11(a). In fact, the makespan of MECDS is shorter than that of the EADAGS by about 21%.

7 Conclusions and Future Work

To address the issue of energy-aware scheduling for data-intensive workflow applications in virtualized platforms, this work presented a novel heuristic for VM deployment and task scheduling. Experimental results based on both synthetic and real world workloads show that the proposed algorithm is more robust than other DVFS-based algorithms, especially when the system is in the presence of intensive data-accessing requests. In the future, we plan to incorporate some adaptive mechanisms into MECDS, such as workload-aware and load-balance mechanism, configurable strategy for VM deployment, and energy-aware VM migration mechanism. Furthermore, we are planning to design a more energy-efficient VM scheduler in VM hypervisor level. In this way, we hope to further reduce the overheads caused by intensive I/O requests when scheduling data-intensive workflows in cloud systems.

References

[1] Sun D W, Chang G R, Gao S, Jin L Z, Wang X W. Modeling a dynamic data replication strategy to increase system availability in cloud computing environments. *Journal of Computer Science and Technology*, 2012, 27(2): 256-272.

[2] Sedaghat M, Hernández F, Elmroth E. Unifying cloud management: Towards overall governance of business level objectives. In *Proc. the 11th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, May 2011, pp.591-597.

[3] Iosup A, Yigitbasi N, Epema D. On the performance variability of production cloud services. *In Proc. the 11th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, May 2011, pp.104-113.

[4] Mahadevan P, Banerjee S, Sharma P, Shah A, Ranganathan P. On energy efficiency for enterprise and data center networks. *IEEE Communications Magazine*, 2011, 49(8): 94-100.

[5] Goth G. Data center operators face energy irony. *IEEE Internet Computing*, 2010, 14(2): 7-10.

[6] Wang J, Feng L, Xue W, Song Z. A survey on energy-efficient data management. *SIGMOD Record*, 2011, 40(2): 17-23.

[7] Figueiredo J, Maciel P, Callou G, Tavares E, Sousa E, Silva B. Estimating reliability importance and total cost of acquisition for data center power infrastructures. *In Proc. the IEEE Int. Conf. Systems, Man, and Cybernetics*, Oct. 2011, pp.421-426.

[8] Li J X, Li B, Wo T Y, Hu C M, Huai J P, Liu L, Lam K P. CyberGuarder: A virtualization security assurance architecture for green cloud computing. *Future Generation Computer Systems*, 2012, 28(2): 379-390.

[9] Garg S K, Yeob C S, Anandasivamc A, Buyyaa R. Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. *Journal of Parallel Distributed Computing*, 2011, 71(6): 732-749.

[10] Juve G, Deelman E, Berriman G B, Berman B P, Maechling P. An evaluation of the cost and performance of scientific workflows on Amazon EC2. *Journal of Grid Computing*, 2012, 10(1): 5-21.

[11] Yuan D, Yang Y, Liu X, Zhang G, Chen J. A data dependency based strategy for intermediate data storage in scientific cloud workflow systems. *Concurrency and Computation: Practice and Experience*, 2012, 24(9): 956-976.

[12] Tolosana-Calasanza R, Bañares J A, Pham C, Rana O F. Enforcing QoS in scientific workflow systems enacted over cloud infrastructures. *Journal of Computer and System Sciences*, 2012, 78(5): 1300-1315.

[13] Sotomayor B, Montero R S, Llorente I M, Foster I. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 2009, 13(5): 14-22.

[14] Chapman C, Emmerich W, Márquez F G, Clayman S, Galis A. Software architecture definition for on-demand cloud provisioning. *Cluster Computing*, 2012, 15(2): 79-100.

[15] Kirschnick J, Alcaraz-Calero J M, Goldsack P, Farrell A, Guijarro J, Loughran S, Edwards N, Wilcock L. Towards an architecture for deploying elastic services in the cloud. *Software: Practice and Experience*, 2012, 42(4): 395-408.

[16] Cherkasova L, Gupta D, Vahdat A. Comparison of the three CPU schedulers in Xen. *ACM SIGMETRICS Performance Evaluation Review*, 2007, 35(2): 42-51.

[17] Krishnan B, Amur H, Gavrilovska A, Schwan K. VM power metering: Feasibility and challenges. *ACM SIGMETRICS Performance Evaluation Review*, 2010, 38(3): 56-60.

[18] Kang H, Chen Y, Wong J L, Radu S, Wu J. Enhancement of Xen's scheduler for MapReduce workloads. In *Proc. the 20th Int. Symp. High Performance Distributed Computing*, June 2011, pp.251-262.

[19] Kim H, Lim H, Jeong J, Jo H, Lee J. Task-aware virtual machine scheduling for I/O performance. In *Proc. the 2009 ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution*, March 2009, pp.101-110.

[20] Abbasi Z, Varsamopoulos G, Gupta S K S. TACOMA: Server and workload management in Internet data centers considering cooling-computing power trade-off and energy proportionality. *ACM Transactions on Architecture and Code Optimization*, 2012, 9(2): Article No.11.

[21] Fang W, Liang X, Sun Y, Vasilakos A V. Network element scheduling for achieving energy-aware data center networks. *International Journal of Computers Communications and Control*, 2012, 7(2):241-251.

[22] Benoit A, Goud P R, Robert Y. Performance and energy optimization of concurrent pipelined applications. In *Proc. the 24th IEEE Int. Symp. Parallel and Distributed Processing*, Apr 2010, pp.1-12.

[23] Baskiyar S, Abdel-Kader R. Energy aware DAG scheduling on heterogeneous systems. *Cluster Computing*, 2010, 13(4): 373-383.

[24] Rizvandi N B, Taheri J, Zomaya A Y, Lee Y C. Linear combinations of DVFs-enabled processor frequencies to modify the energy-aware scheduling algorithms. In *Proc. the 10th IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing*, May 2010, pp.388-397.

[25] Lee Y C, Zomaya A Y. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems*, 2011, 22(8): 1374-1381.

[26] Mezmaza M, Melab N, Kessaci Y, Lee Y C, Talbi E G, Zomaya A Y, Tuyttens D. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems.

Journal of Parallel and Distributed Computing, 2011, 71(11): 1497-1508.

[27] Zhu D, Melhem R, Childers B R. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 2003, 14(7): 686-700.

[28] Zong Z, Briggs M, Connor N, Xiao Q. An energy-efficient framework for large-scale parallel storage systems. In *Proc. the 21st IEEE Int. Symp. Parallel and Distributed Processing*, Mar. 2007, pp.1-7.

[29] Manzanares A, Bellam K, Qin X. A prefetching scheme for energy conservation in parallel disk systems. In *Proc. the 22nd IEEE Int. Symp. Parallel and Distributed Processing*, Apr. 2008, pp.1-5.

[30] Bohra A, Chaudhary V. Vmeter: Power modelling for virtualized clouds. In *Proc. the 24th IEEE Int. Symp. Parallel and Distributed Processing*, Apr. 2010, pp.1-8.

[31] Cho S, Melhem R G. On the interplay of parallelization, program performance, and energy consumption. *IEEE Transactions on Parallel and Distributed Systems*, 2010, 21(3): 342-353.

[32] Kim K H, Beloglazov A, Buyya R. Power-aware provisioning of virtual machines for real-time cloud services. *Concurrency and Computation: Practice and Experience*, 2011, 23(13):1491-1505.

[33] Speitkamp B, Bichler M. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on Services Computing*, 2010, 3(4): 266-278.

[34] Hupfeld F, Cortes T, Kolbeck B, Stender J, Focht E, Hess M, Malo J, Martí J, Cesario E. The XtreemFS architecture — A case for object-based file systems in grids. *Concurrency and Computation: Practice and Experience*, 2008, 20(17): 2049-2060.

[35] Topcuoglu H, Hariri S, Wu M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 2002, 13(3): 260-274.

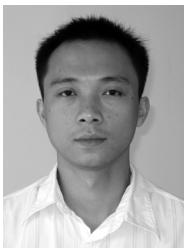
[36] Calheiros R N, Ranjan R, Beloglazov A, De Rose C A F, Buyya R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 2011, 41(1): 23-50.

[37] Berlinska J, Drozdowski M. Scheduling divisible MapReduce computations. *Journal of Parallel and Distributed Computing*, 2011, 71(3): 450-459.

[38] Kiss T, Greenwell P, Heindl H, Terstyánszky G, Weingarten N. Parameter sweep workflows for modelling carbohydrate recognition. *Journal of Grid Computing*, 2010, 8(4): 587-601.

[39] Kansal A, Zhao F, Liu J, Kothari N, Bhattacharya A A. Virtual machine power metering and provisioning. In *Proc. the 1st ACM Symp. Cloud Computing*, June 2010, pp.39-50.

[40] Theiner D, Wieczorek M. Reduction of calibration time of distributed hydrological models by use of grid computing and nonlinear optimisation algorithms. In *Proc. the 7th Int. Conf. Hydroinformatics*, Sept. 2006.



Peng Xiao received the M.S degree in computer science from Xiamen University, China. Now, he is currently a Ph.D. candidate in Central South University, Changsha and a lecturer in Hunan Institute of Engineering, Xiangtan. His research interests include cloud computing, resource virtualization technology, and HPC energy-efficiency management.

Zhi-Gang Hu received his M.S. and Ph.D. degrees in computer science from Central South University, Changsha. Now he is a professor in Central South University. His current research interests include parallel and distributed computing, cloud computing, grid computing, green HPC policy and management, and power optimization in embedded systems.

Yan-Ping Zhang received her M.S. degree in computer science from Central South University, Changsha. Now she is a Ph.D. candidate in Technical University of Munich, Germany. Her research interests include cloud-based large-scale application, virtual machine power model, workflow scheduling model and algorithm.