EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments

Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, Liang Zhong
School of Computer Science and Engineering
Beihang University
Beijing, China
{libo, lijx, woty, liqin, zhongl}@act.buaa.edu.cn, huaijp@buaa.edu.cn

Abstract*—With the increasing prevalence of large scale cloud computing environments, how to place requested applications into available computing servers regarding to energy consumption has become an essential research problem, but existing application placement approaches are still not effective for live applications with dynamic characters. In this paper, we proposed a novel approach named EnaCloud, which enables application live placement dynamically with consideration of energy efficiency in a cloud platform. In EnaCloud, we use a Virtual Machine to encapsulate the application, which supports applications scheduling and live migration to minimize the number of running machines, so as to save energy. Specially, the application placement is abstracted as a bin packing problem, and an energy-aware heuristic algorithm is proposed to get an appropriate solution. In addition, an over-provision approach is presented to deal with the varying resource demands of applications. Our approach has been successfully implemented as useful components and fundamental services in the iVIC platform. Finally, we evaluate our approach by comprehensive experiments based on virtual machine monitor Xen and the results show that it is feasible.

Keywords: Cloud Computing, Virtual Machine, Energy Saving, Application Placement, Live Migration

I. INTRODUCTION

Recently, cloud computing has become a popular computing paradigm in which virtualized and scalable resources are provided as services over the Internet. Various cloud computing products and projects have been tremendously beneficial to network applications, such as Amazon EC2 [1], IBM Blue Cloud [2] etc. However, the keep running of the large scale of computing and data centers generally requires a large amount of energy, and energy consumption is a critical issue for IT organizations. For example, in 2006, data centers consumed about 4.5 billion kWh, equaling roughly 1.5% of the total U.S. electricity consumption, and trends show that power consumption keeps growing at 18% annually [22].

In fact, enormous energy has been wasted due to idle resources. A report [7] from NRDC pointed that servers sitting idle still use 69-97% of total energy even if power management function is enabled. In our evaluating experiments on a Dell PC with Core2 CPU, it consumes about 85*W* when sitting idle, almost half of the energy when sitting full-loaded. However, in a parallel and distributing computing environment, most of job scheduling research approaches [21] focus much on how to schedule independent or loosely-coupled tasks in a shared system. The objective is to balance the workload among servers, so as to maximize system throughput. But these studies have a lack of energy-saving considerations, and for the cloud computing environment with thousands of machines may cause huge energy waste.

Therefore, how to place the applications in a cloud platform to reduce energy consumption becomes an urgent problem. Many research works have proposed energy-saving computing methods, but there are some issues should be addressed for a cloud platform.

First, some studies [6][14][15] present some techniques such as voltage settings, processor speed adjustment and features such as turning off display, sleep mode etc. And they are only useful for PC or single computer. In particular, they cannot achieve the maximum energy optimization, since the energy saved by these techniques such as scaling down the CPU voltage is far less than powering off a computer. An energy-saving approach for the whole cloud platform is needed.

Second, a way named "workloads concentration" is used in large scale data centers to vacate some server nodes, and then power off them to save energy. But this approach depends on static configuration and setting previously. In an open cloud, applications (i.e. workloads) generally arrive and depart dynamically, and will break the "workloads concentration" state. For example, when an application finishes its job and departs, it will release the occupied resource; thereby the state of "workloads concentration" will be violated due to the idleness of resources.

978-0-7695-3840-2/09 $26.00 © 2009 IEEE
DOI 10.1109/CLOUD.2009.72

Third, many applications have varying resource demands, e.g., an application may request more resources during its running, or else the service quality of the applications will be decreased dramatically. But existing approaches need to shut down the applications and copy them to an idle server, it cannot support live application migration. Besides, the application may also release some resources, and then the server will be underutilized. Thus, it requires an approach supporting dynamic resource allocation and application live migration.

To address the above issues, we proposed a novel approach called *EnaCloud*, which can enable application live placement in consideration of energy efficiency and application dynamic characteristic in large scale of cloud computing environments.

The major contributions are summarized as follows:

- We use VM (Virtual Machine) to encapsulate the application and the VM live migration [8] techniques are utilized to support application live placement, thus the application can move to another physical server without interrupting the service.
- An energy-aware heuristic algorithm is proposed to get the application placement schemes regarding to the arrival, departure or resizing events of applications. Moreover, we introduce a resource provision method to optimize our approach to avoid the over frequent application migration due to resource resizing.
- An architecture of *EnaCloud* system is designed and implemented in the iVIC platform, which is a virtual computing environment developed for HaaS (Hardware as a Service) and SaaS (Software as a Service) applications. Some experimental studies show that the energy consumption can be effectively reduced through our approach.

The remainder of this paper is organized as follows. Section II introduces some concepts and problems statement, and Section III presents the energy-aware heuristic algorithm for application placement. We introduce implementation experience of *EnaCloud* in Section IV. The performance evaluation is given and analysed in Section V. We discuss related work about energy-saving in the area of computing system and virtual machine in Section VI. Finally, we conclude the paper in Section VII.

II. Problem Statement and System Model

A. Terminology and Assumption

The infrastructure of cloud computing environment [1][2] is usually composed of hundreds or even thousands of server nodes. The nodes can be categorized into two types – computing nodes and storage nodes. We assume that all the data and files are stored in storage nodes, which are running network file system. Each computing node consists of processor, memory, as well as network interfaces. For simplicity, we assume all the computing nodes are homogenous and the resource capacity of every server is 1 *unit*. The nodes are interconnected by high-speed LAN such as Infiniband. Each computing node runs a virtual machine monitor (VMM) and hosts one or more VMs. Each VM encapsulates an application or a component of the application. The application and the underlying operating system, which are encapsulated by a VM, are referred to as *workload.* We assume each workload has a predefined resource requirement when being submitted to our system. We use the term *open box* for a server node that is running VMs. The idle server node without running VMs is referred to as *close box*.

B. Problem Statement

In *EnaCloud*, the workloads are aggregated tightly so as to ensure the number of *open boxes* is minimal. In a cloud platform, the workloads always arrive or depart dynamically. **An Example**. When a new workload arrives, it should be assigned to an *open box* as far as possible, without opening a *close box*. As shown in Fig. 1(a), a new box should be opened, according to traditional application placement method, when a workload (0.5) arrives. But as shown in Fig. 1(b), if we firstly migrate the workload (0.3) from the first node to the second node, then the newcomer (0.5) can be inserted to the first node without opening a close box.

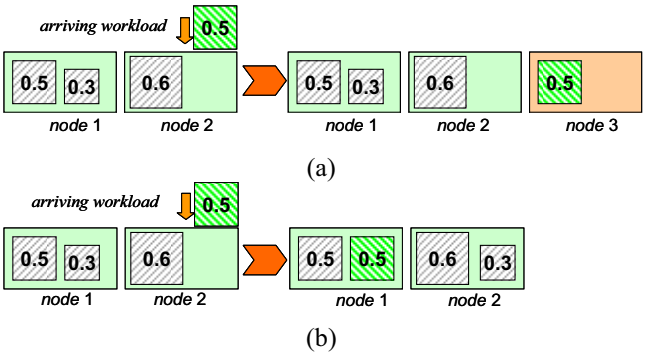


Figure 1. (a) Without migration, it requires three open boxes when inserting a new workload. (b) With migration, it remains two open boxes when inserting a new workload.

When one workload finishes its job, some of the other workloads should be remapped, so as to vacate an open box and hibernate it. The process should also be automated through live migration. In particular, many applications have varying resource demands, so we call this event as workload resizing. Workload resizing includes *workload inflation* and *workload deflation*. Workload inflation can affect the performance of the other workloads hosted on the same node. Workload deflation will release some resources and lead to resource idleness and energy waste. Therefore, the problem is how to remap workloads to the resource nodes through migration when a workload arrives, departs or resizes. The migration has two goals: (1) minimize the number of the open boxes; (2) minimize the times of migration.

C. System Model

Based on the above concepts and analysis, the system can be modeled as follows:

Given a resource pool ($node_1$, $node_2$, …, $node_n$) and a sequence of workloads ($workload_1$, $workload_2$, …, $workload_m$), there are three types of events which will trigger application migration: *workload arrival*, *workload departure* and *workload resizing*. So the input of system is a sequence of workload events: (…, $workload_i^A$, …, $workload_j^D$, …, $workload_k^R$, …). When an event occurs, an energy-aware heuristic algorithm will generate an application placement scheme (this scheme includes a series of the workloads insert and migration operations) to minimize the energy consumption. Then workloads are remapped to the resource nodes based on the scheme. The whole model is illustrated in Fig. 2.

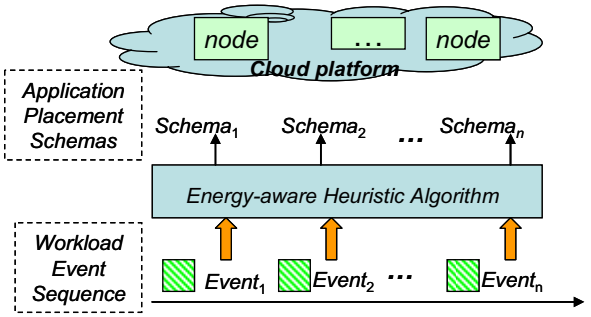


Figure 2. System Model

III. ENERGY-AWARE HEURISTIC ALGORITHM

In this Section, we present the design of our energy-aware heuristic algorithm. It is the "brain" of *EnaCloud*, aiming to guide the application placement including application migration to the most power efficient nodes, thus minimizing the total energy consumption of the cloud server pool.

As we all know that a resource node will achieve the most energy efficiency when it is full-loaded. So our energy-aware algorithm tries to concentrate workloads to the minimal set of resource nodes. We abstract it as a classical bin packing problem. But the difference is that in our problem workload may arrive, depart or resize at any time. These three events will happen randomly, and the current one doesn't know who the successor is. Taking workload arrival event as an example, the new workload should be immediately assigned to a resource node, without the knowledge of subsequent workloads. So our algorithm will work in an event-driven manner and compute an application placement scheme each time when an event happens.

A. Basic Idea

The classical bin packing was one of the original NP-complete problems [9]. Similarly, our problem is equivalent to the classical bin packing, and we resort to a heuristic algorithm to produce an acceptable solution. Note that the classical bin packing algorithms like First-Fit-Decreasing (FFD) [10], Best-Fit and Worst-Fit may also be applicable here with some modifications. However, these algorithms will bring lots of resource gaps (For example, a node with two workloads [0.3, 0.4], and its gap is 1-0.3-0.4=0.3). Just as Fig. 1(a) shows, the new workload (0.5) had to be placed into a new bin without migration, while $node_1$ has a gap (0.2) and $node_2$ has a gap (0.4). This is because in these algorithms, a workload can only be placed once. In other words, if a workload has been put into a bin, it has no chance to be moved to another. Our basic idea is to narrow these resource gaps through migration, and get an approximate optimal solution - the minimal node usage. Next, we discuss our idea with three events.

Workload Arrival Event. When a workload arrives, the principle is inserting it into open boxes without opening a close box. The algorithm is based on a simple heuristic rule that the small workloads are more likely to be inserted into the gaps. So differing from FFD, each time a new workload arrives, the heuristic does not simply put the newcomer into the first node that can accommodate it, but tries to displace packed workloads smaller than the newcomer with the newcomer. The smaller workloads extruded from the node are then reinserted into the resource pool in the same manner. Take Fig. 1(b) as an example, when a new workload (0.5) arrives, the algorithm displace the smaller workload (0.3) in node1 with the newcomer (0.5), and then insert workload (0.3) into $node_2$, thus avoiding opening a new node. For the whole system, workload (0.3) migrates from $node_1$ to $node_2$, and a new workload (0.5) is placed to $node_1$.**Workload Departure Event** can be dealt with in a similar way. When a workload departs from one node, the other workloads on that node are popped and reinserted into the pool. If these workloads can be inserted to other open boxes, then the current node is closed. **Workload Resizing Event** is equivalent to a workload departure event plus a workload arrival event.

B. Algorithm Details

Our idea is based on the fact that smaller workloads are easier to be inserted into the gaps. The whole process involves replacing smaller workloads with bigger one and reinserts the smaller ones, thus leads to a lot of reinsertion operations. To reduce the number of reinsertions and lower the complexity of the algorithm, we divide the region of the workload size (0, 1] into 2M-2 subintervals according to a partition method from [11], and each subinterval represents a *level*.

$L_0 = ((M\text{-}1)/M, 1]$
$L_1 = ((M\text{-}2)/(M\text{-}1), (M\text{-}1)/M]$
…
$L_{M\text{-}1} = (1/3, 1/2]$
…
$L_{2M\text{-}4} = (1/M, 1/(M\text{-}1)]$
$L_{2M\text{-}3} = (0, 1/M]$

We define L_k is higher than L_{k+1}. A workload can only be replaced by the workload belonging to a higher level.

Workload Arrival. Based on the idea described in Section III(a), we propose a recursive algorithm (shown in Table I) to insert the new arrival workload into the nodes pool.

TABLE I. INSERT PROCEDURE

Procedure: *Insert*
Input: *x,* size of the arrival workload
Output: a placement scheme
1. **if** *level*(*x*)=2*M*-3 or *level*(*x*)=*0*
2. insert *x* using First-Fit
3. return the destination node of *x*
4. **foreach** node *v* in pool
5. **foreach** workload *w* in node *v*
6. filter out *w* where *level*(*w*)<*level*(*x*)
7. place *x* to *v** using Best-Fit
8. sort each workload *w** in *v** where *level*(*w**)<*level*(*x*) to { w_1*, … , w_n* } in ascending order
9. **for** i = 1 **to** n
10. **if** *v** can accommodate *x*
11. **break**
12. pop w_i* from *v** and *Insert* (w_i*)

The input of the *Insert* procedure is the size of the new arrival workload. The output is a workload placement scheme such as:

[$workload_1(node_1, node_2)$, $workload_3(node_2, node_5)$, $workload_8(null, node_1)$]

Here, $workload_1(node_1, node_2)$ denotes migrating $workload_1$ from $node_1$ to $node_2$. Next, we'll give a brief description of the *Insert* procedure. If $x \in L_{2M-3}$ or $x \in L_0$, it will be directly inserted into the pool based on the First-Fit algorithm, when a new workload *x* arrives. If *x* belongs to the other levels, the algorithm temporarily filters out workloads whose levels are lower than L_x. In other words, workloads whose level is equal or greater than *x* will be temporarily ignored. Then the algorithm inserts *x* based on Best-Fit, the result is that *x* is placed into node *v**. At the same time, some workloads on *v**, whose levels are lower than L_x, are extruded in ascending order. It means that the lower-level workloads are popped earlier than higher-level workloads. The extrusion process continues until the node can accommodate *x*. The popped workloads are reinserted according to the *Insert* procedure. In the whole procedure, the new workload is inserted appropriately and some old workloads are relocated, and a placement scheme is generated.

Workload Departure. When a workload finishes its work and departs from node *x*, the algorithm reinserts the other workloads on *x*. Each reinsertion may produce a placement scheme. The *Pop* procedure returns the Union of the placement schemes. The process of *Pop* is shown in Table II.

TABLE II. POP PROCEDURE

Procedure: *Pop*
Input: the node *x* that the workload departs from
Output: migration scheme
1. **foreach** workload *w* in node *v*
2. pop *w* and invoke *Y*=*Insert*(*w*)
3. Return $\cup Y_i$

Workload Resizing. The procedure of *Resize* is rather straightforward. It can be transformed to a *Pop* and an *Insert* Procedure. The detail of *Resize* procedure is shown in Table III.

TABLE III. RESIZE PROCEDURE

Procedure: *Resize*
Input: old size *x* of workload, new size *y* of workload
Output: migration scheme
1. *X*=*Pop*(x)
2. *Y*=*Insert*(y)
3. Return $X \cup Y$

C. Over-provision

In our basic algorithm, the *Resize* procedure which involves one *Pop* and one *Insert* Procedure may incur more migration overhead. Especially for real-world applications, the resource demands of applications may change frequently, which will result in lots of *resizing* events.

To reduce the *resizing* overhead, we introduce an over-provision method as an enhancement to our algorithm. The over-provision method means allocating more resources to the workloads than they actually needed according to some expected events.

We define an *over-provision ratio* α ($0 <= a <= 1$) for the percentage of extra resources needed to be allocated for a workload. When a new workload *x* arrives, our system first allocates *size'(x)*= $(1+\alpha) \times size(x)$ resources for it. The *size*(*x*) is a predefined resource demand of *x*. Our algorithm insert *x* according to $(1+\alpha) \times size(x)$. Since the *Resource Provision Manager* dynamically adjusts the resource allocation, when *x* resizes from *size*(*x*) to *size'(x)*, the algorithm first check to see if *size*'(*x*) is between $(1-\alpha) \times size(x)$ and $(1+\alpha) \times size(x)$. If yes, the resizing event will be ignored; if not, the size of *x* is updated to *size'(x)*, and the allocation to *x* is adjusted to $(1+a) \times size'(x)$. For example, the over-provision ratio is set to 0.2. When a new workload (0.5) arrives, it firstly allocates (0.6). The algorithm inserts (0.6) into the pool. If the actual size of the workload changes to 0.55, since 0.55 is between 0.5 and 0.6, the *resizing* event will be ignored by the algorithm. But if the actual size of the workload changes to 0.65, since 0.65 is not between 0.5 and 0.6, the *resizing* event will be handled by the algorithm.

From the above analysis, we can see, though over-provision may cause some resource wastes, it can reduce the number of resizing operation. The over-provision ratio is an adjustable value. Through adjusting the over-provision ratio, the algorithm can balance the relations between energy efficiency and migration overhead. To further reduce the number of resizing operations and ensure that a small burst does not trigger needless migration, a resizing event is invoked only if thresholds are exceeded for a sustained time.

IV. System Implementation

A. System Architecture

In this section, we propose the *EnaCloud* framework for automating the workload concentration process in cloud computing environments. The system architecture is illustrated in Fig. 3.

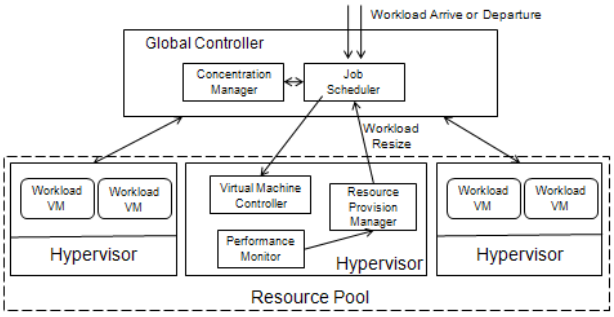


Figure 3. EnaCloud Architecture

In *EnaCloud*, there is one globally central node, which runs the *Concentration Manager* and *Job Scheduler*. The *Job Scheduler* receives the workload arrival (e.g. a user submit an application to run), departure and resizing events (e.g. allocating more resource for the workload), and deliver them to the *Concentration Manager*. The *Concentration Manager* will generate an application placement scheme composed of a series of insertion and migration operations based on these events, and then send this scheme back to *Job Scheduler*. The J*ob Scheduler* decomposes the placement scheme to a set of insertion and migration commands, and then dispatches them to the *Virtual Machine Controller*.

The *Virtual Machine Controller, Performance Monitor* and R*esource Provision Manager* are deployed in each resource node. The *Virtual Machine Controller* receives commands from *Job Scheduler*, and invoke the VM management interface of the hypervisor to execute the commands delivered from *Job Scheduler* such as VM start, stop or migrate. The *Resource Provision Manager* dynamically adjusts resource allocation (VM Resizing) to the workload VM based on the performance statistics periodically collected by the *Performance Monitor*. Each adjustment generates a *resizing* event, and it will be submitted to the *Job Scheduler*.

B. Implementation Experience

iVIC[1] is a virtual computing environment developed for HaaS and SaaS applications. iVIC enables users to dynamically create and manage various kinds of virtual resources such as Virtual Machines, Virtual Clusters and Virtual Networks. It also can deliver on-demand streaming applications to a client in a real-time manner without on-premise installation.

The *EnaCloud* framework has been implemented in iVIC system with Python. The energy-aware heuristic algorithm is implemented in the *Concentration Manager*. The communication between the global node and resource nodes is via SOAP message. We choose Xen as the hypervisor, and implement *Resource Provision Manager* based on the Xen `credit scheduler` in non-working conserving mode, which provides strong performance isolation among VMs. And the *Resource Provision Manager* uses Xen interfaces to dynamically adjust `cap` parameters of the scheduler to change the resource allocation of VM during runtime, based on an adaptive provision method mentioned in [13]. The exact command is: `xm sched-credit -d domain -w Weight -c Cap.`

We implement the *Virtual Machine Controller* using Xen's Python management API to start, stop and migrate the VM. By querying `xentop`, the Performance Monitor can obtain real-time performance statistics of each workload VM, which includes CPU and Memory usage.

V. Experimental Study

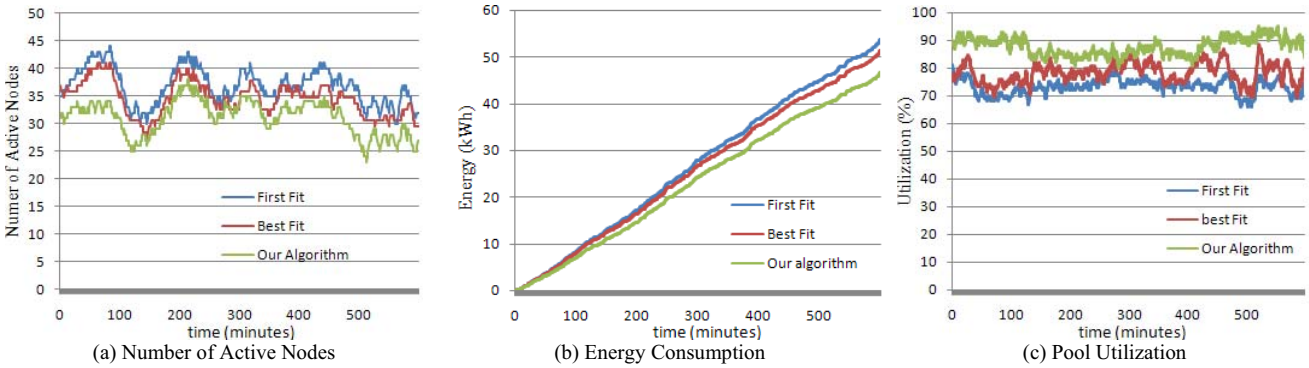
A. Experiment Setup

To evaluate the effectiveness of our approach in a real cloud setting, we conducted a series of experiments in our iVIC environment. Our experiment environment is based on an iVIC cloud pool consisting of sixty servers with Intel Core2 Duo 3.0 GHz, 4G RAM, Linux 2.6.18 operation system, Xen 3.0.3 virtual machine monitor, and gigabit Ethernet connection. We use a Pentium-D PC to run the *Concentration Manager* and *Job Scheduler*. Some other desktop machines are used as clients to send requests to server application. A Voltech PM3000 AC*E* power analyzer is used to measure the transient power and total energy consumption of the server pool.

Workloads Generation: We use three types of workloads to simulate the diversity of applications in a cloud.

- *Web Server and Database Server*: We choose Apache Web server and MySQL server to simulate applications continuously running for a long time, and `SPECWeb 2005`[2] tool is used to performance evaluation.
- *Compute-intensive Applications*: We choose three Bioinformatics applications: *Blast* (Sequence Alignment Parallel Tool), *Siclone* (Gene Finding

[1] http://www.ivic.org.cn
[2] http://www.spec/org/web2005/



(a) Number of Active Nodes (b) Energy Consumption (c) Pool Utilization

Figure 4. Energy efficiency of First Fit, Best Fit and Our algorithm

Tool), and *CAP3* (Gene Sequence Assembly Tool) to simulate applications which consume a lot of CPU cycles. We also use Spec CPU 2006 [12] to generate different levels of CPU and Memory loads.

- *Common Applications*: We choose some popular software including C compiler to simulate applications which require less computing power.

In *EnaCloud*, each application is encapsulated into a VM, and it will start automatically together with the VM. At the same time, the execution time of these applications is different, and we just choose them to simulate various workload lifetime. For example, application of *Web server and Database Server* are long-stay workloads, and *Common Applications* e.g., kernel compilation are comparatively short-time workloads.

During the experiments, we randomly create a workloads set with above applications, and submit them to the cloud pool with a given rate. When the workload VM runs out of its time, it will be stopped immediately by Virtual Machine Controller.

To better simulate the behaviors of a cloud computing environment, we launch the workloads according to a Poisson distribution with $\lambda = 1/\overline{T}_{interval}$ where $\overline{T}_{interval}$ denotes the average interval between the arrival of any two workloads. Through adjusting the value of $\overline{T}_{interval}$, the overall system load can be controlled.

For our algorithm, we set M=4, so (0,1] is divided into 6 subintervals: L_0= (3/4, 1], L_1= (2/3, 3/4], L_2= (1/2, 2/3], L_3= (1/3, 1/2], L_4= (1/4, 1/3], L_5= (0, 1/4].

B. Experiment Results

Experiment Group 1: The purpose of this experiment is to evaluate the effectiveness of our energy-saving approach. We submit 1,000 workloads with $\lambda = 1/300$ ($\overline{T}_{interval}$ = 5min), which means workload is submitted every five minutes on average. To evaluate the performance of our algorithm, we compared it with *First Fit* and *Best Fit* in terms of the number of active server nodes, energy consumption, and pool utilization.

The results are shown as Fig. 4. Fig. 4(a) shows our algorithm uses less active nodes to run the workloads than *First Fit* and *Best Fit* algorithm. At the 84th minute, our algorithm only uses 34 active nodes, saving 10 and 6 nodes compared with *First Fit* and *Best Fit* respectively. The reason is our algorithm exploits live migration to further concentration workloads, so it can ensure a tightly concentrated state at any time.

Fig. 4(b) shows the total energy consumption for the three algorithms. Compared to *First Fit* and *Best Fit* algorithm, our algorithm exhibits much more energy saving and saves about 10% and 13% energy respectively. This result shows that our algorithm successfully saves energy through reducing the active nodes used by the workloads.

Fig. 4(c) shows that our algorithm can maintains the pool utilization at 90% on average, which is close to the full capacity of nodes. As higher utilization indicates low resource wasting, it shows the energy efficiency of our algorithm from another aspect. We can also see that *Best Fit* is better than *First Fit* to a limited extent. We have tried to optimize the current algorithm, but the improvement is very limited, since our algorithm has already achieved a high utilization, there is not much space to further concentrate.

Experiment Group 2: The purpose of this experiment is to study the energy cost of application live migration. The results in Table IV show that the energy consumption of application migration almost increases linearly with the increasing number of memory usage of the workloads. For instance, when migrating a workload with 512MB memory, it is 1Kwh (3600000 Joule) energy consumption for almost 4597 times workload migration. This result shows that a small additional energy cost for a limited number of migration to optimize the whole system.

TABLE IV. ENERGY FOR APPLICATION MIGRATION

Memory (*MB*)	128	256	512	1024
Energy (*Joule*)	202	399	783	1524

Experiment Group 3: The purpose of this experiment is to investigate the impact on over-provision ratio for the energy saving and migration times. Over-provision ratio α is a metric to measure how much our algorithm can tolerate the varying of workload resource demand.

As shown in Fig. 5 (we select a time period of 100 minutes from the total 24 hours), it saves about 250W power when α=0.1, and about 350W power over α=0.25 and α=0.3 on average. But for α=0.1, it exhibits more jitters than the others. This is because in low over-provision ratio, our algorithm will need many times of migrations due to frequently resizing, and we also observe a significant increase in frequency of nodes power-on and power-off. For α=0.25 and α=0.3, the two result curves appear to almost overlap.

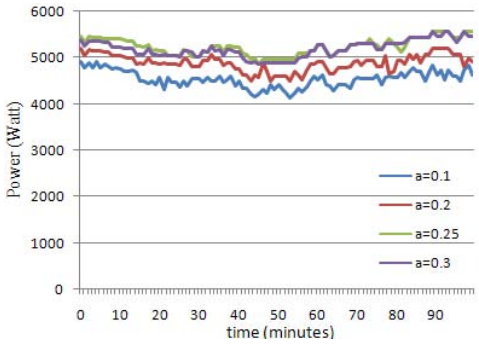


Figure 5. Power consumption for different α

Moreover, we study the impact on migration times for different over-provision ratio. From Table V, we can see that for α=0.1, it will bring about 1.7 times of migration per event and 5.8 times of migration per minute. As for our system, these two values are acceptable. As for α=0.2, 0.25, 0.3, each brings small migration overhead.

We also inspect other over-provision ratios. When α=0.4, the pool utilization drops below 70%, and leads to a lot of resource waste. For α=0.05, it leads to excessive number of migrations and greatly affects the application's performance. So generally the reasonable range of α is [0.1, 0.3]. And this is an important direction for configure our system in a real cloud platform.

TABLE V. MIGRATION TIMES FOR DIFFERENT OVER-PROVISION RATIO

Over-provision ratio / Migration times	α=0.1	α=0.2	α=0.25	α=0.3
Per event	1.7	1.0	0.6	0.5
Per minute	5.8	3.3	1.9	1.7

Last, we study the effect of workload submission rate. Experiment results show that if $\overline{T}_{interval}$ is too low (for example, less than 30 seconds), our algorithm will incur lots of migration. In this situation, our algorithm will degrade to *Best Fit* to avoid excessive migration.

VI. RELATED WORK

Energy efficiency has become one of the most active topics in large scale of data center or cluster computing environment today. As evidence, consider that processor and chipset vendors are marketing products on "performance per watt", instead of just processor clock frequency and benchmark performance. Early studies [6][14][15] mainly focus on energy consumption reducing for PC or single server node. Operating systems, such as Windows, already provide a rich set of energy saving features including the ability to turn off the display and automatically put the system to sleep when the user is not interacting with the computer. Moreover, some techniques can dynamically vary the voltage settings and rotational speeds of processors and disks based on the processor and I/O demands for computers. However, they cannot achieve the maximum optimization, since the energy saved by these techniques such as scaling down the CPU voltage is far less than powering off a computer.

Some research work [3][4][5][16][17] implement energy-efficiency policy in a front-end load balancer to distribute the requests to backend servers, so as to minimize the energy consumption without violating application's QoS requirements. And these approaches are only useful to server hosting environments, in which physical servers are shared among competing server applications like Web server. But in cloud computing environments, a wide range of applications will be hosted. Our approach uses VM to encapsulate applications, and leverages live migration feature to achieve the energy efficiency placement, thus it is more appropriate to a cloud platform with various applications.

Manget [20] presents a multilayer ring-based overlay architecture and also uses VM live migration to transfer load among the server nodes. But the main difference between Manget and our work is that Manget adopts a periodically reconfiguration method, the interval of reconfiguration can greatly affect the performance. If the interval is too short, it will incur a lot of migrations; if the interval is too long, it will result in energy waste. Our approach employs an event-driven way to dynamically adapt to the workloads changing events, so can avoid such problem.

In addition, some energy models have been proposed and are used to guide the application placement. For example, mantis [18] presents a linear power model for full-system power analysis and modeling. PMapper [19] proposes a power-aware placement algorithm based a power model and a migration cost model. While these models generally based on some assumptions, and are not applicable to a real computing environment. Our approach does not explicitly depend on a power model, but provides an approach during the phase of application placement to minimize the number of the running nodes, so can be integrated into many computing environments.

VII. CONCLUSION

Cloud computing refers to the trend that computing power is becoming a utility, generated remotely and delivered as a service over the Internet. How to provide an energy-efficiency application placement schema for a cloud platform has become a critical problem. Our work is an essential supporting for such applications to get economical

running. In our study, we proposed *EnaCloud* which is an energy-saving application live placement approach for large scale of cloud platform. An energy-aware heuristic algorithm is proposed to choose an appropriate schema for dynamic application placement. Moreover, an over-provision approach is presented to deal with frequent resource resizing issue. We have implemented our approach based on Xen VMM and our experience shows initial evidence that it is a viable solution to save the energy for a cloud platform.

Our ongoing work is to develop an improved algorithm with consideration of multi-factor (including CPU, Memory etc.), and also give more practical evaluations for real applications in our iVIC platform.

ACKNOWLEDGMENT

This work is partially supported by grants from China 863 High-tech Program (Project No. 2007AA01Z120, 2009AA01Z419), and National Natural Science Funds for China (Project No. 60525209, 60731160632). We would also like to thank members in Network Computing Research team in Institute of Advanced Computing Technology of Beihang University for their helpful suggestions.

REFERENCES

[1] The Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/

[2] IBM Blue Cloud. http://www.ibm.com/ibm/cloud/

[3] R. Doyle, “Energy Management for Server Clusters”, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, p.165, May 20-22, 2001.

[4] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, Ronald P. Doyle, “Managing energy and server resources in hosting centers”, Proceedings of the eighteenth ACM SOSP, October 21-24, 2001, Canada.

[5] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. “Energy conservation in heterogeneous server clusters”. Proceeding of ACM PPoPP, 2005.

[6] Tibor Horvath , Tarek Abdelzaher , Kevin Skadron , Xue Liu, “Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control”, IEEE Transactions on Computers, v.56, p.444-458, 2007.

[7] Natural Resources Defense Council “Recommendations for Tier I ENERGY STAR Computer Specification”,. http://www.energystar.gov/ia/partners/prod_development/revisions/downloads/computer/RecommendationsTierICompSpecs.pdf

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A.Warfield, “Live Migration of Virtual Machines”, Proceedings of 2nd Symposium on Networked Systems Design and Implementation, USENIX, 2005.

[9] Miller, R. E., Thatcher, J. W. , “Reducibility Among Combinatorial Problems”, In Complexity of Computer Computations, pp. 85-103, New York, 1972.

[10] Yue, M., “A simple proof of the inequality FFD(L) ≤ (11/9)opt(l)+1, for all L, for the FFD bin-packing algorithm”, Acta Mathematicae Applicatae Sinica , pp. 321–331, 1991.

[11] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. Journal of the ACM, 32(3):562-572, July 1985.

[12] SPEC CPU2006, next-generation, industry-standardized, CPU-intensive benchmark suite. http://www.spec.org/cpu2006/

[13] Wang, Z., Zhu, X., Singhal, S., “Utilization and SLO-based control for dynamic sizing of resource partitions”, In Proceeding of 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM), October 2005.

[14] Xiaotao Liu , Prashant Shenoy , Weibo Gong, “A time series-based approach for power management in mobile processors and disks”, Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video, June 16-18, 2004.

[15] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole, “A Feedback-driven Proportion Allocator for Real-Rate Scheduling”, In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI), pages 145–158, February 1999.

[16] Rajamani, K., Lefurgy, C., “On evaluating request-distribution schemes for saving energy in server clusters.”, In Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software, 2003.

[17] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems". Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP), September 2001.

[18] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan, “Full-system power analysis and modeling for server environments”, In 2nd WS Modeling, Benchmarking & Simul., pages 158--168, Boston, MA, June 2006.

[19] Akshat Verma, Puneet Ahuja, Anindya Neogi, “pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems”, Middleware 2008: 243-264.

[20] Liting Hu, Hai Jin, Xiaofei Liao, Xianjie Xiong, Haikun Liu. “Magnet: A novel scheduling policy for power reduction in cluster with virtual machines”, In Proceedings of the 2008 IEEE International Conference on Cluster Computing, September, 2008, Tsukuba, Japan.

[21] Fu, S. and Xu, C. 2004. Migration Decision for Hybrid Mobility in Reconfigurable Distributed Virtual Machines. In Proceedings of the 2004 international Conference on Parallel Processing (August 15 - 18, 2004). ICPP. IEEE Computer Society, Washington, DC, 335-342.

[22] Report to Congress on Server and Data Center Energy Efficiency,U.S. Environmental Protection Agency ENERGY STAR Program, http://www.energystar.gov/ia/partners/Prod_development/downloads August 2, 2007.