Magnet: A Novel Scheduling Policy for Power Reduction in Cluster with Virtual Machines

Liting Hu, Hai Jin, Xiaofei Liao, Xianjie Xiong, Haikun Liu

Services Computing Technology and System Lab

Cluster and Grid Computing Lab

School of Computer Science and Technology

Huazhong University of Science and Technology, Wuhan, 430074, China

hjin@hust.edu.cn

***Abstract*—The concept of *green computing* has attracted much attention recently in cluster computing. However, previous local approaches focused on saving the energy cost of the components in a single workstation without a global vision on the whole cluster, so it achieved undesirable power reduction effect. Other cluster-wide energy saving techniques could only be applied to homogeneous workstations and specific applications. This paper describes the design and implementation of a novel approach that uses live migration of virtual machines to transfer load among the nodes on a multilayer ring-based overlay. This scheme can reduce the power consumption greatly by regarding all the cluster nodes as a whole. Plus, it can be applied to both the homogeneous and heterogeneous servers. Experimental measurements show that the new method can reduce the power consumption by 74.8% over base at most with certain adjustably acceptable overhead. The effectiveness and performance insights are also analytically verified.**

I. INTRODUCTION

Green computing has been a hot topic in cluster computing for many years. Anecdotal evidence from data center operators [1][9] indicates that a significant fraction of the operation cost of these centers is due to power consumption and cooling. In addition, most power-generation technologies (such as nuclear and coal-based generation) are harmful to the environment.

Previous work on energy savings of the cluster can be divided into two groups, local techniques and cluster-wide techniques. Local techniques have focused on reducing power of the single workstation, by reducing the clock frequency, the supplied voltage or by saving interconnect components, including switches, *network interface cards* (NICs), and links [3][13]. However, it is a pity that most of them focus on the improvement of single node without a global vision of the whole system. On the contrary, cluster-wide techniques involve multiple servers. The basic idea of cluster-wide techniques is to aggregate the system load and then to determine the minimal set of servers which could handle the load. The energy savings can be obtained by turning off the redundant nodes. However, the cluster-wide approaches: 1) can only be applied to specific applications (e.g. Web service) in that its front-end has provided a good load balancing algorithm to redirect connection states of the back-end nodes; 2) focus solely on homogeneous systems yet real-life clusters are almost invariably heterogeneous; 3) fail to deal with run-time reallocation of load.

Significant progresses have been made on both sides, including local techniques and cluster-wide techniques. However, it seems to reach the bottleneck already. Fortunately, the emergence of the *virtual machine* (VM) gives us a new horizon and thus we can look upon the problem at a different angle. A virtual machine was originally defined by Popek and Goldberg as *an efficient*, *isolated duplicate of a real machine*. The VM used in our work is system VM (sometimes called *hardware VMs*), which allows the multiplexing of the underlying physical machine between different VMs, each running its own operating system.

The virtual machine exhibits the unique advantages as follows: first, it allows the separation of hardware and software and thus addresses the problem caused by heterogeneous computing platforms; second, live migration [5][10] of VMs allows the workload of a node to transfer to another node.

That is not to say, however, that we can make virtual machines randomly migrate among all nodes. In fact, the potential overhead caused by live migrations of VMs can not be ignored, which may have serious negative effect on cluster utilization, throughput and QoS issues. Therefore the challenge is how to design a migration strategy to effectively implement *green computing* and meanwhile influence little on the performance of the cluster.

Our *green computing* algorithm tends to turn off the redundant nodes to save the energy, provided that the system performance is guaranteed by the left nodes. The reason is that each node in our cluster consumes approximately 160 watts when idle and 280 watts when all resources are stretched to the maximum. It means that: 1) there is a difference in power consumption between an idle node and a fully utilized node; 2) the penalty for keeping a node powered on is high even if it is idle. Thus, turning a node off always saves power, even if its load has to be transferred to one or more other nodes.

We propose a policy, called *Magnet*, to implement *green computing* in the cluster with VMs. Magnet keeps track of all active nodes and organizes them into concentric, non-overlapping rings in terms of gradually decreasing workload,

978-1-4244-2640-9/08/$25.00 © 2008 IEEE

so it is easy to *squeeze* the existing running jobs which are widely distributed among lightweight nodes and then deliver them to a subset of current active nodes and it is also easy to *release* the overweighted nodes, thereby 1) turning off the redundant nodes to save energy when the system is in non-intensive computing state; 2) transferring violating jobs or big jobs to the free nodes when the system is in intensive computing state to obtain performance gains.

By conducting simulations from generated application workload with different intensities in the cluster with VMs, we show that our method can effectively reduce the power consumption by 67.1%, 72.0%, 69.3%, 72.8% and 74.8% at most for five application workload groups (light, moderate, normal, moderately intensive, and highly intensive job submission rates, respectively). Our method can increase the cluster utilization to 41.9%, 35.44%, 36.57%, 45.17% and 50.61%, respectively. Our method can also increase the quality-of-service for the heavyweight workload group. Plus, we show that the total migration overhead is acceptable and adjustable. The effectiveness and performance insights are also demonstrated through a theoretical analysis.

The rest of this paper is organized as follows. Section II discusses the related work. The Magnet policy is described in Section III. Section IV demonstrates the effectiveness of Magnet through theoretical analysis. Section V describes our simulation methodology. Section VI presents the performance evaluation. We conclude this work in Section VII.

II. Related Works

We divide previous work into two groups: local and cluster-wide technique. Local techniques are implemented independently by each server, whereas cluster-wide techniques involve multiple servers.

A. Local Techniques

Most of the local techniques aiming at reducing power consumption of a computing cluster focus on the improvement of the single node, by reducing the clock frequency, by reducing the supplied voltage or by saving interconnect components in computer. For instance, the DVS system dynamically reduces processors' supply voltages while guaranteeing proper operations. The DLS [12] project makes use of an appropriate adaptive routing algorithm to shut down links in a judicious way.

Elnozahy [16] proposed a new mechanism called *request batching*, in which the incoming requests are accumulated in memory by the network interface processor while the host processor of the server is kept in a low-power state. The host processor is awakened when an accumulated request has been pending for longer than a batching timeout.

However, such schemes do not achieve the maximal optimization as experimental results (SPECpower_ssj2008 [20]) confirm that the incremental energy savings from slowing down all CPUs (and scaling down their voltage) are far less than those from turning a machine off to reduce farm capacity by the same amount. Plus, request batching trades off system responsiveness to save energy, so it is not appropriate to trade or e-commerce server in that a very slow server will drive away customers.

B. Cluster-wide Techniques

Pinheiro [17] and Chase [4] concurrently proposed similar strategies for managing energy in the context of front-end server clusters. The basic idea of such approaches is to leverage the aggregate system load and then determine the minimal set of servers which could handle the load. Finally, energy savings can be obtained by turning some machines on or off. Elnozahy et al. [15] evaluated different combinations of cluster reconfiguration and dynamic voltage scaling for clusters in which the base power is relatively low, including *Independent Voltage Scaling* (IVS), *Coordinated Voltage Scaling* (CVS), *Vary-On Vary-Off* (VOVO), *Combined Policy* (VOVO-IVS), and *Coordinated Policy* (VOVO-CVS).

Such approaches have drawbacks including that (1) it is only applicable to specific applications (e.g. Web service) in which the front-end has already provided a good load balancing algorithm and the connection states of the back-end node can be migrated across different nodes. In that sense, the given load could be concentrated on a subset of all nodes in a balance way, yet the strategy fails on most application servers; (2) they focus solely on homogeneous systems. However, real-life clusters are almost invariably heterogeneous in terms of their operation systems, the performance, capacity and power consumption of their hardware components; (3) most of these approaches fail to deal with run-time reallocation of the load.

III. MAGNET Design

To achieve good performance in the cluster, Magnet faces the following challenges: (1) how to design a framework to run in the *green computing* way and meanwhile to influence little on the performance; (2) how to decrease the overhead caused by frequent live migrations of VMs; (3) how to maintain the service continuity and stability (decreasing the impact of interruption caused by the crash down of unknown node). In this section, we discuss the situations which lead to bad performance to the cluster computing first and then introduce the design of Magnet.

Generally the performance of the cluster, such as throughput, average job slowdown and QoS issues are likely to be influenced by two problems, called as *inner job blocking problem* and *outer job blocking problem*. The former is caused by certain violating jobs with seriously fluctuant resource requirements which lead to node thrashing. Previous studies [22][24] focused on balancing the number of jobs/tasks among the workstations, but the CPU or memory requirement should be informed in advance. The latter happens due to the coming of a big job with remarkable working set requirement which can not be satisfied by the current active workstations, resulting in the blocked working flow of rest of the jobs. Towards *outer job blocking problem*, existing schemes like backfilling scheduling [19] and gang scheduling must consider the size of node needed by a job or estimated runtimes [21].

The analysis above indicates us that if only we could conserve the energy globally and meanwhile flexibly address the job blocking problems, we can attain a win-win situation, saving energy and improving performance.

A. Overview

As illustrated in Fig. 1, the basic workflow of Magnet is as follows. First, the multilayer ring-based overlay is constructed and new jobs arrive continuously and are submitted to the service center. Second, a detector is employed to supervise the *evil* states of the computing cluster which can be categorized into four kinds: (1) *saving energy state* caused by the arrived lightweight working flow which persists for a long time; (2) *inner job blocking state* caused by node thrashing; (3) *outer job blocking state* caused by violating or big jobs; (4) *fault tolerance state* caused by certain nodes crashing down due to physical or software malfunctions; Third, a controller is employed to choose corresponding strategies to response the *evil* states.

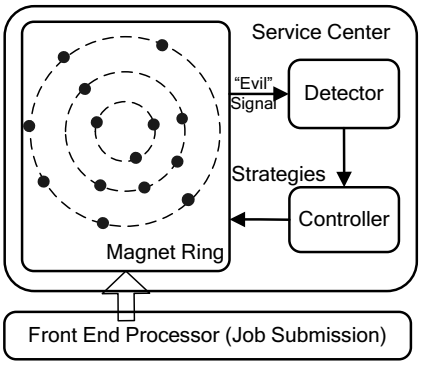

Fig. 1. The system diagram of Magnet

B. Multilayer Ring Based Overlay

Let *degrad* be the maximum acceptable weight a node could suffer *without* performance degradation. *degrad* can be given by the user expectation, the QoS requirement of the application and the like. Under different situations, *degrad* is different and we can not set a constant value. Let *Max*(*vm*) be the maximum number of VM containers a node could have. *Max*(*vm*) is also uncertain as it changes in accordance with different configurations of the physical machine. Let *violate* be the weight a node could suffer which leads to unacceptable performance degradation. For example, let *violate* be 89%, then the node which consumes the resource ratio exceeding 89% will be regarded as violating node and should be released.

In the first operation of Magnet, each node boots up one VM and new jobs are submitted to these VMs. Then every VM, with a unique ID, consumes resources of the physical nodes at different rates (from 0% to 100%). Magnet keeps track of all active nodes, and organizes them into concentric, non-overlapping rings in accordance with decreasing resource consumed ratios. Magnet maintains *3-layer* rings, the members of each ring suffer the workload that span the range [0, *degrad*/2), [*degrad*/2, *degrad*), [*degrad*, 100%). Magnet deals with the members of the outer ring by squeezing them to the secondary ring. VMs on the secondary ring are not recommended to merge together as the sum of their resource consumption rates exceed the performance upper bound (*degrad*).

In the second step, the *leader* takes the responsibility of maintaining a stable Magnet ring-based overlay for the reason that the memory and CPU demand of jobs may change dynamically and the execution time may not know in advance. The leader is the node who suffers the maximum load among its ring members of the same layer.

Within each ring, the leader periodically updates his logical links with his members. At regular intervals, the leader checks whether the workload of his members is less than the threshold of his layer. If not, he removes the node which does not belong to his layer to the alternative appropriate layer.

Finally, Magnet system addresses the resilience issue of Magnet ring-based overlay through the introduction of *co-leaders*. Each leader recruits the co-leader at the time being elected.

To detect unannounced departures, Magnet relies on heartbeats exchanged among leaders and their crews. Unreported nodes are given a fixed time interval before being considered to be dead. If the failure node happens to be the leader, the members of the leader's ring regards co-leader as the replacement leader. In that sense, co-leader improves the resilience of the Magnet ring-based overlay by avoiding dependencies on single nodes.

C. Squeeze Measure and Release Measure

In this section, we analyze the overhead caused by the migrations of VMs and then introduce the *squeeze* and *release* migration measures which decrease the overhead to the maximum extent.

VM migration is the key in the energy saving method of aggregating and redistributing the system load in the cluster consisting of VMs. The most optimal effect can be achieved by (1) calculating the overall load of the system and being divided by the capacity of a single physical machine to get the number of nodes which can handle the overall load; (2) transferring the scattered load to the calculated minimal set of servers by a sequence of live migrations. The challenge is how to obtain the optimal effect meanwhile decrease the overhead caused by VM migrations.

Overhead. Previous work [5][23] have conducted a series of experiments to measure the overhead for migrating a number of running VMs from one physical host to another. The results show that, the overhead of VM migrations is reflected in two aspects, the time cost for all the migrations and the throughput loss of the competitive VMs on the target node.

Let r be a fixed remote execution cost in second; B be the bandwidth; D be the amount of data in bits (OS image) to be transferred in the job migration; N be the times of all VM migrations. The time cost for all the migrations can be given by:

$$Overhead(Time(s)) = (r + \frac{D}{B}) \times N \quad (1)$$

Second, the throughput loss is caused by the VMs' competition for the shared cache although the VMs have been already isolated in terms of CPU and memory. It has been shown that page faults frequently occur in some heavily loaded nodes but a few memory accesses or no memory accesses are requested on some lightly loaded nodes or idle nodes [2]. Therefore, in order to decrease the total throughput loss, it is recommended to merge lightly loaded VMs together on the physical host and avoid transferring one heavily loaded VM to the host inside which there is another heavily loaded VM running.

Squeeze Measure. The above analysis gives us the directions for minimizing the overhead: the less *D*, *N*, and probability of allocating heavily loaded VMs together, the less overhead will be achieved. While in *energy saving* state or *outer job blocking* state, Magnet will take *squeeze* measure to migrate a sequence of VMs on the outer layer ring, which is similar to the process of constructing an optimal tree. The difference lies in that the process will stop if the sum of load of VMs on a physical host exceeds the upper bound load of their layer (*degrad*/2). Fig.2(a) presents the detail *squeeze* steps. Let the number inside the VM be the amount of load in terms of a percentage and 33% be the upper bound of the cluster *P1* consisting of outer layer ring nodes (*degrad*=66%). The physical host which contains 4 VMs (4%, 6%, 10%, and 26%) will be removed to the secondary layer ring. Each step of *squeeze* measure merges two lightest loaded VMs together and thus guarantees the minimal *D* and minimal probability of the mergence of heavily loaded VMs. Moreover, the process of constructing the optimal tree leads a relatively smaller value of *N*.

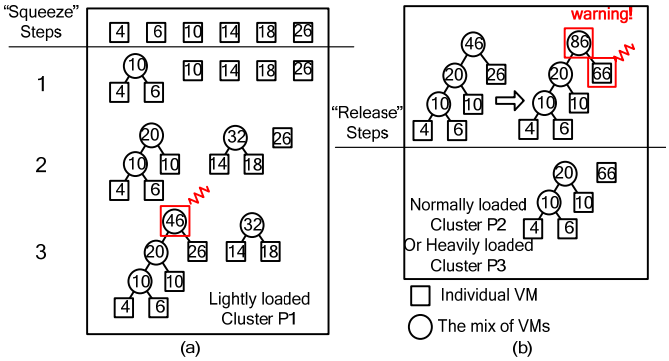

Fig. 2. (a) The *squeeze* steps in detail (b) The *release* steps in detail

Release Measure. While in *inner job blocking state*, Magnet will take *release* measure to the VM whose host machine consumes the resource ratio exceeding the upper bound (*violate*). *Release* measure is the inverse process of *Squeeze* measure and the disjointed *violating* VMs will be transferred to the free nodes on the lightly loaded nodes of the outer layer ring. Fig.2(b) presents the detail *release* steps, in which the VM (26%) mutates to be a violating VM (66%) due to some unknown reasons, e.g. its jobs' fluctuant demand of a large memory space. Consequently, the physical machine of the VM (66%) becomes a *violating* node (suppose *violate* is 80%) and then it is released.

D. The Rationale of Our Solutions

In this section, we separately discuss our solutions to the four common *evil* system states and the rationale behinds them.

In *saving energy state,* Magnet virtually reconfigures the cluster system to further utilize resources by taking *squeeze* measure on outer layer nodes (see Fig.3). At next step, Magnet switches the high-power state of the released nodes to the low-power state. You may question why we do not submit these lightweight jobs to certain nodes from the very beginning and thus save trouble and overhead of all these migrations. Actually, a situation is ignored, that jobs on cluster may change dynamically, heavily loaded nodes could become lightly loaded when its jobs are completed or terminated that we do not know in advance. Therefore, saving energy is not so much a predetermined strategy as a feedback-driven process.

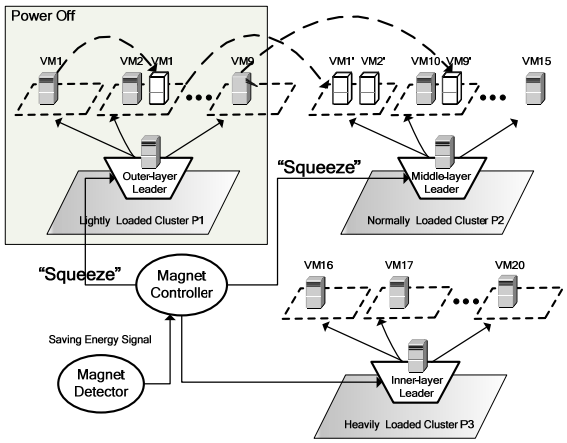

Fig. 3. Magnet reconfiguration in saving energy state

In *inner job blocking state*, it is necessary to make the violating job migrate to the node which can provide enough memory space or CPU resource. More importantly, experiments have shown that a large job is likely to be a large job with long lifetime [6][11]. In the sense, the candidate node should not be a heavyweight node as it is likely to be the same for a long time and thus causes another *inner job blocking* state.

Magnet takes *release* measure on the violating VMs. Magnet maintains a multilayer ring-based overlay among which the lightweight workload nodes are organized on the outer layer ring, so it is easy to find the candidate nodes by requesting the corresponding outer layer leader for the list of his members and then choosing one from them, as illustrated in Fig.4.

In *outer job blocking state,* a new job is coming, which demands large memory space and CPU resource. Unfortunately, the available space of each individual node is not large enough to serve it and thus the following submissions to the workstation will be blocked. However, if

the idle memory spaces of all individual nodes can be accumulated, then the sum may fit the large job. Magnet takes *squeeze* measure on outer layer nodes to release the resources.

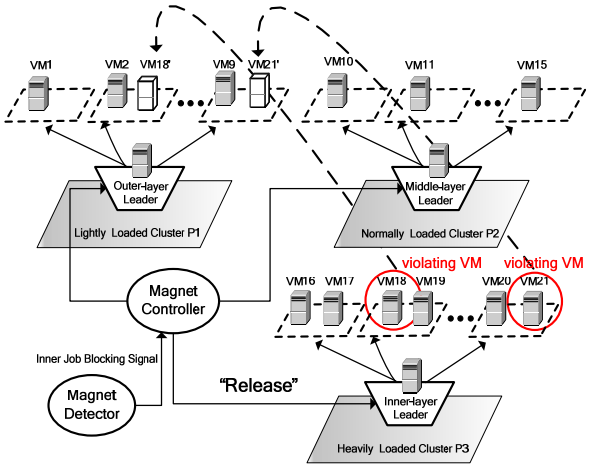

Fig. 4. Magnet reconfiguration in inner job blocking state

More importantly, Magnet can provide both the centralized and distributed job scheduling approaches in the cluster. Previous FEP emphasizes a lot on the characteristics of jobs by adjusting its scheduling strategies in accordance with job types, such as rigid jobs and moldable jobs. Undoubtedly, the centralized management has many advantages. However, when the scalability is restricted or high fault tolerance is required, distributed management is an alternative scheme. Magnet can transfer the responsibilities of the FEP to the workstations in cluster, making the workstations themselves decide which node to run certain task. In other words, the computing cluster acts like a *black box* which is transparent to the user and the FEP.

IV. Analysis

A. Energy Saving Modelling

Let V be the set of all workstations, t be the given interval for Magnet reconfiguration, $S_M(t) \subset V$ be the set of the active workstations during t, ε be the average watt of electricity consumed by one workstation per second, $\Delta E = \left(|V| - |S_M|\right) \times \varepsilon \times t$ representing the energy saved during t.

Assuming that node N_i follows Poisson distribution requiring K_i percentage of resource ($0<K_i<100$) with the mean rate λ_i (intensity of workload) [24], hence the probability density of resource consumption of N_i can be calculated as $P(X_i = k_i) = e^{-\lambda_i} \lambda_i^{k_i} / k_i!$.

As the sum of N independent Poisson distributions still follows Poisson distribution, at time t, the sum of workload on all the workstations follows Poisson distribution. Suppose that the *mean* workload of the outer layer ring, the secondary ring and the inner layer ring is $\overline{load_1} \in [0, k_1)$, $\overline{load_2} \in [k_1, k_2)$ and $\overline{load_3} \in [k_2, 100]$, respectively.

Assuming the percentage of workstations suffering $\overline{load_1}$, $\overline{load_2}$, $\overline{load_3}$ is P_1, P_2 and P_3, respectively, so there are $|V|P_1$, $|V|P_2$, and $|V|P_3$ workstations on the outer ring, the secondary ring and the inner ring, where:

$$P_1 = \int_0^{k1} \frac{e^{-\lambda} \lambda^r}{r!} dr,\ P_2 = \int_{k1}^{k2} \frac{e^{-\lambda} \lambda^r}{r!} dr,\ P_3 = \int_{k2}^{100} \frac{e^{-\lambda} \lambda^r}{r!} dr$$

$P_1+P_2+P_3=1$

Before reconfiguration: $|S_M| = |V|$

After reconfiguration: The load on the outer ring and the secondary ring will be partially *squeezed* or *transferred*. Magnet only deals with the members of the outer ring by squeezing them to the secondary ring, then:

$$|S_M| = |V| P_1 \frac{\overline{Load_1}}{Load_2} + |V| P_2 + |V| P_3, \text{ and then}$$

$$|V| - |S_M| = |V| - |V| P_1 \frac{\overline{Load_1}}{Load_2} + |V| P_2 + |V| P_3 = |V| P_1 (1 - \frac{\overline{Load_1}}{Load_2})$$

Thus, the saved energy can be given by

$$\Delta E = \left(|V| - |S_M|\right) \times \varepsilon \times t = |V| P_1 (1 - \frac{\overline{Load_1}}{Load_2}) \times \varepsilon \times t$$

$$= |V| \left(\int_0^{k1} \frac{e^{-\lambda} \lambda^r}{r!} dr \right) (1 - \frac{\overline{Load_1}}{Load_2}) \times \varepsilon \times t \qquad (2)$$

We have $\overline{load_1} < \overline{load_2}$, so $\Delta E > 0$.

The above model gives conditions for the Magnet reconfiguration to reduce the total electricity. A key condition for performance gains is from k_1, λ and variance between $\overline{load_1}$ and $\overline{load_2}$. The more k_1, the less λ and the larger difference between $\overline{load_1}$ and $\overline{load_2}$, the more energy will be saved.

B. Quality of Service Modelling

The total execution time of job i in a workload for i=1, 2,... n, $t_{exe}(i)$ is expressed as $t_{exe}(i)=t_{cpu}(i)+t_{page}(i)+t_{que}(i)$, where $t_{cpu}(i)$, $t_{page}(i)$, and $t_{que}(i)$ are the CPU service time, the paging time for page faults, the queuing time waiting in a job queue.

$$T_{exe} = \sum_{i=1}^{n} t_{cpu}(i) + \sum_{i=1}^{n} t_{page}(i) + \sum_{i=1}^{n} t_{que}(i) = T_{cpu} + T_{page} + T_{que}$$

After Magnet reconfiguration, with the equation (1), we have

$$\hat{T}_{exe} = \hat{T}_{cpu} + \hat{T}_{page} + \hat{T}_{que} + \hat{T}_{mig} = \hat{T}_{cpu} + \hat{T}_{page} + \hat{T}_{que} + (r + \frac{D}{B}) \times N$$

The jobs demand identical CPU service on both cluster environments, so that $T_{cpu} = \hat{T}_{cpu}$.

For *inner job blocking* problem, the paging time reduction ($T_{page} - \hat{T}_{page}$) can be achieved by making jobs with large memory demands migrate to the nodes of the outer ring which has enough resources. The *inner job blocking* problem happens during the running process of the system, so that $\hat{T}_{que} = T_{que}$.

In that sense,

$$\Delta T = T_{exe} - \hat{T}_{exe} = (T_{page} - \hat{T}_{page}) - \hat{T}_{mig} = (T_{page} - \hat{T}_{page}) - (r + \frac{D}{B}) \times N \quad (3)$$

For *outer job blocking* problem, we can conclude that $T_{que} > \hat{T}_{que}$ as Magnet helps keep the workflow smooth. It is under the assumption that the total resource can suffice the jobs, so $T_{page} = \hat{T}_{page}$. Then we have:

$$\Delta T = T_{exe} - \hat{T}_{exe} = (T_{que} - \hat{T}_{que}) - \hat{T}_{mig} = (T_{que} - \hat{T}_{que}) - (r + \frac{D}{B}) \times N \quad (4)$$

The above model gives conditions for the Magnet reconfiguration to reduce the total execution time of jobs. The equation (3) and (4) tell us that the more Magnet reconfigures the nodes to maintain the stable overlay, the more difference of T_{page} and $\hat{T}_{page}$ or T_{que} and $\hat{T}_{que}$ will be obtained. However, the migration times N will increase too, so ΔT is not always positive. Certainly, less D (data amount) and less B (bandwidth) will lead to smaller time cost for migrations.

V. Experimental environment

A. A Simulated Cluster with VMs

We have simulated a cluster with 64 homogeneous hosts, each of which has an AMD Athlon 3500+ processor and 1GB DDR RAM. Storage is accessed via iSCSI protocol from a NetApp F840 *network attached storage* server (NAS). Moreover, each host has an Intel Pro/1000 NIC to transfer the images of the VMs with 1000 Mbps network bandwidth. We used Xen 3.10 as the virtual machine monitor on each host in all cases and the host kernel for XenLinux is a modified version of Linux 2.6.18.

In the simulation of the cluster, the virtual machine is configured to use 512MB of RAM, the memory page size is 4KB, page fault service time is 10ms, and the context switch time is 0.1ms. The remote submission/execution cost, r, is 0.01s for 1000 Mbps network. Each host maintains a dynamically changed load index file which contains CPU, memory, and I/O load status information. Magnet periodically collects the load information among the workstations.

B. Application Workload

In order to effectively conduct Magnet policy with unknown CPU or memory demands, we need to select different benchmark programs which are representing different types of jobs and then we mix them together to generate the application workload at different submission rates.

The large scientific and system programs we use are from [7], which are representative CPU-intensive, memory-intensive, and/or I/O-active jobs: *bit-reversals* (bit-r), *merge-sort* (m-sort), *matrix multiplication* (m-m), *a trace-driven simulation* (t-sim), *partitioning meshes* (metis), *cell-projection volume rendering for a sphere* (r-sphere), and *cell-projection volume rendering for flow of an aircraft wing* (r-wing). Chen [7] have measured the execution performance of each program and monitored their memory performance in a dedicated computing environment. Table I [7] presents the results of all the seven programs, where the *data size* is the number of entries of the input data, the *working set* gives a range of the memory space demand during the execution, the *lifetime* is the total execution time of each program.

TABLE I
Execution Performance and Memory Related Data of the Seven Application Programs

Programs	Data Size	Working Set (MB)	Lifetime (s)
bit-r	2^{23}	64.22	192.26
m-sort	2^{23}	64.27	82.76
m-m	$1{,}700^2$	66.37	4902.29
t-sim	31,061	4.64	41.63
metis	1M-4M	1.37-4.30	124.41
r-sphere	150,000	36.84-39.66	318.64
r-wing	500,000	19.53-23.39	72.28

TABLE II
Benchmark Result Summary

Performance			Power
Target Load	**Actual Load**	**ssj_ops**	**Average Power**
100%	99.2%	40,852	336
90%	89.1%	36,677	308
80%	80.7%	33,235	288
70%	69.0%	28,398	263
60%	58.7%	24,157	241
50%	49.8%	20,512	225
40%	39.5%	16,281	207
30%	30.0%	12,337	194
20%	20.0%	8,237	181
10%	10.1%	4,142	170
Active Idle		0	159

SPECpower_ssj2008 [20] shows the relationship between the workload and the power consumptions (Table II), which have been formulized by us to calculate the power consumption under certain target load. The results are applied as input file.

The application workload consisting of different types of jobs is randomly submitted to the cluster. Each job has a header item recording the submission time, the job ID, and its lifetime measure in the dedicated environment. Following the header item, the execution activities of the jobs are recorded in a time interval of every 100*ms* including CPU cycles, the memory allocation demand and the details of its VM migrations including its VM container ID, the source and destination node, the start and the end time. Thus, the power consumption can be calculated by the closely monitored CPU and memory utilization rates with Table II. Meanwhile, the total execution time, the average slowdown and the time cost of migrations can be given by the logs.

C. Job Submission Rate Generations

In order to implement our policy across a broad range of workload intensities, we have conducted our experiment at different submission rates respectively. Similar to [7], we

have also generated the job submission rates by the lognormal function:

$$R_{in}(t) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(\ln t-\mu)^2}{2\sigma^2}} & t > 0 \\ 0 & t \le 0 \end{cases} \quad (5)$$

where $R_{in}(t)$ is the lognormal arrival rate function, t is the time duration for job submissions in a unit of seconds, and the values of μ and σ adjust the degree of the submission rate. The lognormal job submission rate has been observed in several practical studies [6][18]. Five application workload groups with different arrival rates are illustrated in Table III, where *APP-1*, *APP-2*, *APP-3*, *APP-4*, *APP-5* represents light, moderate, normal, moderately intensive and highly intensive submission rate, respectively, *Submission Duration* is the time duration for job submissions in a unit of seconds.

TABLE III
JOB SUBMISSION RATES OF THE APPLICATION WORKLOAD

Application Workload	σ	μ	Amount of Jobs	Submission Duration (s)
APP-1	3.9	3.9	318	5,499
APP-2	4.1	4.1	450	5,503
APP-3	4.3	4.3	565	5,497
APP-4	4.5	4.5	707	5,510
APP-5	4.6	4.6	993	5,498

D. Migration Cost Estimation

The approach we propose is a continual optimization approach, where we dynamically make the VMs migrate from one physical server to another in order to minimize the total power consumption. The migration process between the two hosts involved the following stages: pre-migration, reservation, iterative pre-copy, stop-and-copy, commitment and activation [21], which requires creation of a checkpoint on secondary storage and retrieval of the VM image on the target server, so applications can continue running during the migration. However, the performance of applications may be influenced in the transition because of cache misses (hardware caches are not migrated) and potential application quiescence. Thus, it is necessary to estimate the time cost for one-time migration of the VM on which the seven benchmark programs (Table II) runs and study the parameters that affect the cost for the further *MT/ET* (section VI) calculation.

Generally speaking, the time cost for the one-time VM migration contains the shutdown delay, the migration duration and the startup delay. As the VM is shut down after being migrated, the shutdown delay is irrelevant to the execution time of jobs running on the VM; As the booting of a new VM is informed in advance, the startup delay is irrelevant to execution time of jobs running on the VM neither. Therefore, we disregard the shutdown delay and startup delay of the 512MB VM in our simulation.

We have measured the migration cost of a VM with 512MB of RAM under different background load, as illustrated in Fig.5. The physical machine is a 3.6GHz Pentium PC with 1GB main memory and a swap space of 1GB, running Linux version 2.6.9. The background load is the mix of the seven application programs as mentioned above.

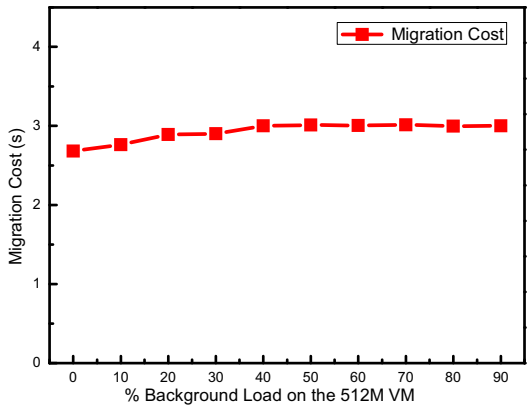

Fig. 5. Migration cost in second of a 512MB VM at a time while under different background load

We observe that the cost of migration is independent of the background load and depends only on the VM characteristics. However, it is based on the premise that the network is idle. Once the task execution environment is communication intensive, the bandwidth will be partially occupied and thus the result is not likely the same.

VI. PERFORMANCE EVALUATION

A. Measured Metrics and Reconfiguration Parameters

To better evaluate the performance of Magnet, we use the metrics as follows: (1) *Power savings over base* is defined as the ratio of the saved electricity to the total electricity during the entire lifetime of the application workload in percentage terms. (2) *Cluster utilization* is defined as the average ratio between the amounts of consumed memory volume to all memory space of active workstations (rule out the shut down nodes) during the entire lifetime of the application workload in percentage terms. (3) *Total execution time* is defined as the sum of the total CPU service time, the total paging time for page faults, the total queuing time waiting in a job queue and the total migration time. (4) *Job slowdown* is defined as the average ratio between its wall-clock execution time and its CPU execution time of all nodes. Plus, we also use the metric (5) *MT/ET* which is defined as the average ratio between the cumulative Magnet reconfiguration time and its total execution time to evaluate the *overhead* of Magnet.

For our experiment, we refer to the time interval between reconfigurations as the *elapse* parameter. Let *degrad=80%* and *Max*(*vm*)=3 .The workload of each layer ranges from 0~40%, 40%~80%, 80%~100%, respectively, of the base resource. To guarantee the QoS of the tasks, the number of the active nodes should not be less than one third of the number of the shut down nodes and for our experiment, the threshold is *eight*. The threshold changes according to the different QoS requirements of the services.

Finally, towards each metric, we compare the following policies:

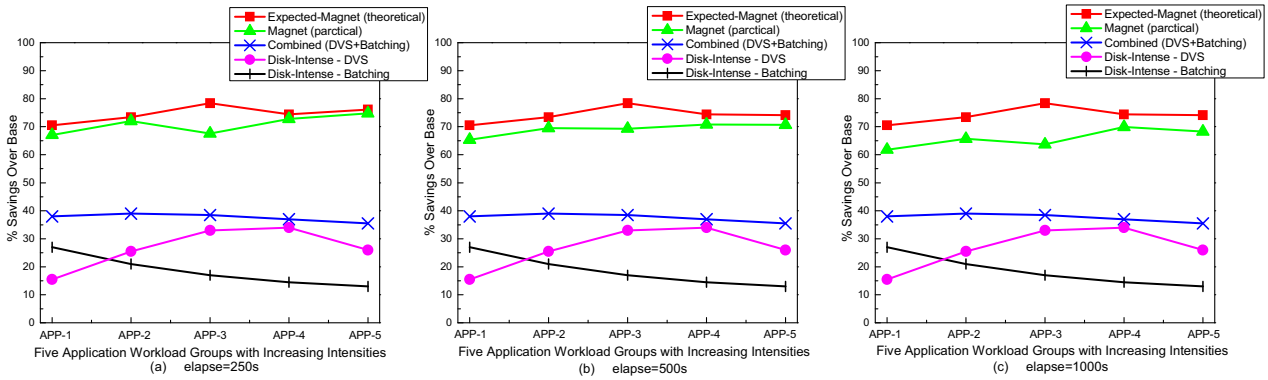

Fig. 6. Energy savings for Magnet, Expected-Magnet and current methods (a) *elapse* = 250 seconds (b) *elapse*=500 seconds (c) *elapse*=1000 seconds

- Basic method without any virtual reconfiguration (Base)
- Current methods (Combined DVS and Batching, Disk Intense Batching and Disk Intense DVS [16])
- Magnet method with expected results deduced by mathematical analysis (Expected-Magnet)
- Magnet method with practical results (Magnet)

The Expected-Magnet results can be calculated with equation (2), $\Delta E = |V|\left(\int_0^{k1}\frac{e^{-\lambda}\lambda^r}{r!}dr\right)(1-\frac{\overline{Load_1}}{Load_2})\times\varepsilon\times t$, where $|V|$ =64, $\frac{\overline{Load_1}}{Load_2}$ =(1+0.4)/(1+0.8). Since the selected application workload tends to be lightweight, $\int_0^{k1}\frac{e^{-\lambda}\lambda^r}{r!}dr$ is set to 1, and ε is set to 170 watts (average power from SPECpower_ssj2008).

B. Improving Power Consumption

Fig.6 presents the energy savings (in percentage) for the Magnet policy over five application workload groups with an increasingly *elapse*, 250s, 500s and 1000s. Compared to Combined (DVS+Batching), Disk-intensive-DVS and Disk-intensive-Batching methods, Magnet policy exhibits much more power savings.

From Fig.6(a), it can be seen that Magnet method, under the reconfiguration *elapse* of 250s, significantly reduced power consumptions. The figure shows that power consumptions are reduced by 67.09%, 72.02%, 67.55%, 72.77% and 74.81% for light, moderate, normal, moderately intensive and highly intensive job submissions, respectively (APP-1, 2, 3, 4, 5).

From Fig.6(b) and 6(c), regarding an increasingly Magnet reconfiguration *elapse* (500s and 1000s), the power consumption are reduced by 67.36%, 69.53%, 69.28%, 70.79%, 70.67% and 61.81%, 63.69%, 63.68%, 69.90%, 68.28%. Note that when *elapse* is 1000s, Magnet performs worse than that of 250s and 500s in the energy saving. This suggests that Magnet performs better while its ring-based overlay is maintained more frequently, for the reason that the less interval time between Magnet reconfigurations, the more redundant workstations can switch to shut down state. Meanwhile, from Fig.6(a), 6(b) and 6(c), we can see that the less *elapse* is, the closer the practical Magnet results come to the theoretical Magnet results.

C. Improving Cluster Utilization

We have also observed the average total consumed memory volumes during the lifetime of job executions in each workload group. Fig.7 presents the comparative average cluster utilization during lifetimes of five workload groups using Magnet scheme and basic scheme.

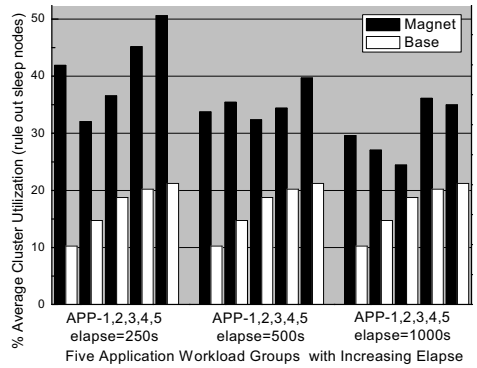

Fig. 7. The average cluster utilizations of the five application workload groups scheduled by Magnet scheme and the basic scheme (Base) with increasing *elapse*s

Compared to the original cluster utilization (10.24%, 14.73%, 18.75%, 20.20% and 21.20% for workload APP-1, 2, 3, 4, 5 respectively), our method can increase the average cluster utilization significantly. When *elapse* is 250s, the average cluster utilization mounts up to 41.89%, 32.04%, 36.57%, 45.17% and 50.61%; when *elapse* is 500s, it mounts up to 33.72%, 35.44%, 32.36%, 34.40% and 39.68%; when *elapse* is 1000s, it mounts up to 29.59%, 27.05%, 24.42%, 36.10% and 34.97%, respectively.

The increasing of the average cluster utilization is caused mainly by the decrease number of the idle nodes. By means of turning off the idle nodes, the overall workload can be *squeezed* to a subset of active workstations and thus increase the throughput. Compared bars of different *elapse* parameters (250s, 500s and 1000s), it is clear that less value

of *elapse* parameter leads to further utilization of active workstations.

D. Improving Quality of Service

Fig.8 and Fig.9 present the comparative *total execution time* and *job slowdown* during lifetimes of five workload groups using Magnet scheme and basic scheme with respect to an increasing value of *elapse* parameter, from 250s to 1000s. From Fig.8**,** it can be seen that when *elapse* is 250s, the total execution time is reduced by *-37.10%*, *-17.83%*, *-27.66%*, 3.47%, and 4.96% for workload APP-1, 2, 3, 4, 5, respectively; when *elapse* is 500s, it is reduced by *-40.72%*, *-21.65%*, *-5.62%*, 0.30%, and 5.10%; when *elapse* is 1000s, it is reduced by *-50.0%*, *-10.51%*, 15.52%, 3.98%, and *-9.43%*.

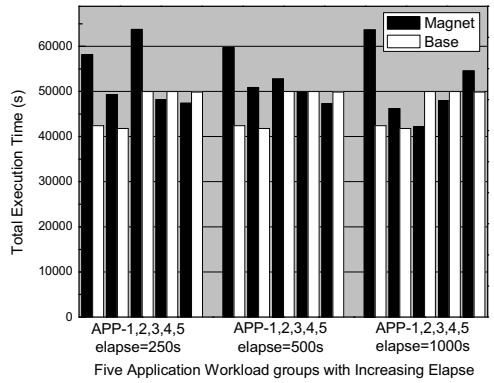

Fig. 8. The total execution times of the five application workload groups scheduled by Magnet scheme and the basic scheme (Base) with increasing *elapse*s

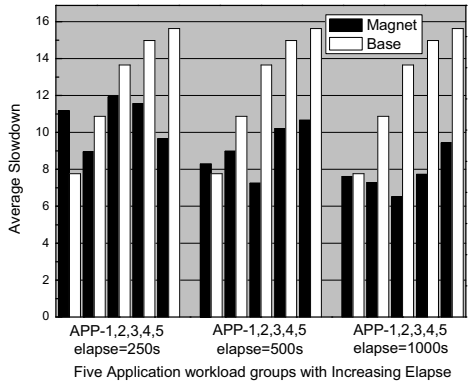

Fig. 9. The average slowdowns of the five application workload groups scheduled by Magnet scheme and the basic scheme (Base) with increasing *elapses*

Note that the result is not positive for light job submissions, moderate job submissions and normal job submissions (APP-1, 2, 3). It is not surprising since job blocking problems happen scarcely under light workload and thus the time increased by live migrations outweighs the time reduced by addressing job blocking problems.

Fig.9 shows that Magnet generally decreases the average job slowdowns of workload APP-1, 2, 3, 4, 5. When *elapse* is 250s, we are able to reduce the average job slowdown by *-43.82%*, 17.61%, 15.35%, 22.85%, and 38.21%; when *elapse* is 500s, it is reduced by *-6.74%*, 17.39%, 46.93%, 31.84%, and 31.79%; when *elapse* is 1000s, it is reduced by 2.00%, 32.98%, 52.28%, 48.4%, and 39.62%. Note that the result is not positive for light job submissions when *elapse* is 250s and 500s, the reason is more likely that frequent migrations lead to longer waiting times yet address much less job blocking problems for lightweight working flow. Therefore, the time increased by live migrations outweighs the time reduced by addressing job blocking problems.

E. Overhead Analysis

As undue Magnet reconfiguration will cause noticeable overhead, it is important to make sure that the QoS is not sacrificed excessively in favor of power and energy savings.

Finally, we test the *MT/ET* (Magnet reconfiguration time/Total execution time) and the average cumulative migration times during the entire lifetimes of the five different working flows (see Fig.10) while increasing the interval between reconfigurations (*elapse*) gradually. *MT* can be estimated as the product of the migration cost in second and the total times of migrations. It shows that the increasing of *elapse* leads to the decreasing of *MT/ET*, indicating that although smaller value of *elapse* parameter achieves better performance on energy saving, it is at the expense of more overhead on the total execution time. However, considering the benefits (more energy savings and cluster utilization) carried by high frequency (see Fig.6, Fig.7), it seems that the most optimal approach is a balanced one that an appropriate value of *elapse* parameter should be chosen.

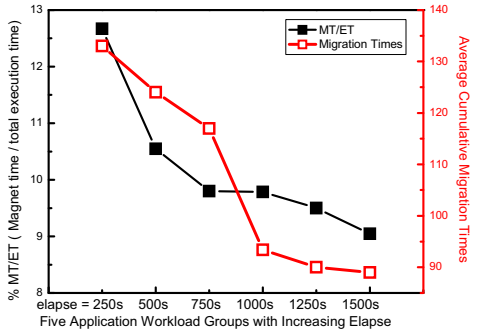

Fig. 10. *MT/ET* in percentage terms and the average cumulative migration times during the entire lifetimes for APP-1, 2,3,4,5 with increasing *elapse*s

VII. CONCLUSION

This paper aims at providing effective strategies to reduce the power consumption and meanwhile influence little on the performance. The contributions can be described as follows: 1) our scheme addresses the limitations caused by heterogeneous computing platforms; 2) an adaptive Magnet approach is proposed to obtain significant energy savings by taking the advantage of live migration of VMs; 3) through the theoretical analysis, we propose the *squeeze* and *release* measures to guide the live migrations aiming at the minimal overhead. Experimental results show that the method have positive impact on the average job slowdown and minor

negative impact on the total execution time. Particularly, the overhead is adjustable by changing the parameter *elapse*.

In the future, we will try to optimize the power reduction effect by exploring more intelligent schemes according to the characteristics of jobs such as CPU intensive, memory intensive or I/O intensive and the like. We will also analyse the strategies of the migration of multiple VMs, e.g. parallel migration and serial migration, to further reduce the impact of VM migration on the system performance. We are hopeful that our theoretical work will be complemented by empirical research that can shed light on the practicality of our provable novel scheduler.

ACKNOWLEDGMENT

This work is supported by National 973 Basic Research Program of China under grant No.2007CB310900, Hubei Natural Science Foundation under grant No.2007ABD009, the Ministry of Education and Intel Joint Information Technology special research fund under grant No.MOE-INTEL-08-06, the research fund supported by HP Labs China.

REFERENCES

[1] APC-American Power Conversion, *Determining Total Cost of Ownership for Data Center and Network Room Infrastructure*. ftp://www.apcmedia.com/salestools/CMRP-5T9PQG_R2_EN.pdf, December 2003.

[2] A. Acharya and S. Setia, "Availability and Utility of Idle Memory in Workstation Clusters", *Proc. ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems*, May 1999, pp.35-46.

[3] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A Dynamic Voltage Scaled Microprocessor System", *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2000, pp.294-295.

[4] J. Chase, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle, "Managing Energy and Server Resources in Hosting Centers", *Proceedings of the 18th Symposium on Operating Systems Principles*, October 2001.

[5] 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 (NSDI'05)*, USENIX, 2005.

[6] S. Chen, L. Xiao, and X. Zhang, "Adaptive and virtual reconfigurations for effective dynamic job scheduling in cluster systems", *Proceedings of the 22nd International Conference on Dist. Comp. Systems*, March 2002.

[7] S. Chen, L. Xiao, and X. Zhang, "Dynamic load sharing with unknown memory demands in clusters", *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS'2001)*, April 2001, pp.109-118.

[8] G. Ciardo, J. Muppala, and K. S. Trivedi, "SPNP: stochastic Petri net package", *Proc Petri NETS and Performance Models*, Kyoto, Japan, 1989, pp.142-151.

[9] M. Hopkins, "The Onsite Energy Generation Option", *The Data Center Journal*, http://datacenterjournal.com/News/Article.asp?article_id=66, February 2004.

[10] W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High Performance Virtual Machine Migration with RDMA over Modern Interconnects", *Proceedings of IEEE International Conference on Cluster Computing (Cluster'07)*, September 2007.

[11] S. Jiang and X. Zhang, "TPF: a system thrashing protection facility in Linux", *Software: Practice and Experience*, Vol.32, No.3, 2002, pp.295-318.

[12] E. J. Kim, K. H. Yum, G. M. Link, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, M. Yousif, and C. R. Das, "Energy optimization techniques in cluster interconnects", *Proceedings of ISLPED*, ACM, 2003, pp.459-464.

[13] E. J. Kim, K. H. Yum, G. M. Link, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, M. Yousif and C. R. Das, "A holistic approach to designing energy-efficient cluster interconnects," *IEEE Trans. Computers*, Vol.54, No.6, 2005.

[14] D. Lifka, "The ANL/IBM SP scheduling system", *Job Scheduling Strategies for Parallel Processing*, *Lect. Notes Comput. Sci.,* Vol.949. Springer-Verlag, 1995, pp.295-303.

[15] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters", *Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, February 2002.

[16] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers", *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[17] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, "Dynamic Cluster Reconfiguration for Power and Performance", *Compilers and Operating Systems for Low Power*, Kluwer Academic Publishers, August 2003.

[18] M. S. Squillante, D. D. Yao, and L. Zhang, "Analysis of job arrival patterns and parallel scheduling performance", *Performance Evaluation*, Vol.36-37, 1999, pp.137-163.

[19] E. Shmueli and D. G. Feitelson, "Backfilling with look ahead to optimize the performance of parallel job scheduling", *Job Scheduling Strategies for Parallel Processing*, *Lect. Notes Comput. Sci.*, Vol.2862, Springer-Verlag, 2003, pp.228-251.

[20] Standard Performance Evaluation Corporation, http://www.spec.org.

[21] Y. Wiseman and D. G. Feitelson, "Paired gang scheduling", *IEEE Trans. Parallel & Distributed Sys.*, Vol.14, No.6, June 2003, pp.581-592.

[22] L. Xiao, X. Zhang, and S. A. Kubricht, "Incorporating job migration and network RAM to share cluster memory resources", *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC-9)*, August 2000, pp.71-78.

[23] M. Zhao and R. J. Figueiredo, "Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources", *Proceedings of 2nd International Workshop on Virtualization Technologies in Distributed Computing (VTDC)*, November 2007.

[24] X. Zhang, Y. Qu, and L. Xiao, "Improving distributed workload performance by sharing both CPU and memory resources", *Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'00)*, April 2000, pp.233-241.