Power Aware Job Scheduling in Multi-Processor System with Service Level Agreements Constraints

Congfeng Jiang, Jian Wan, Xindong You
Grid and Service Computing Technology Lab, Hangzhou Dianzi University, Hangzhou 310037, China
Email: cjiang@hdu.edu.cn

Yinghui Zhao
Department of Hydraulic Engineering and Agriculture, Zhejiang Tongji Vocational College of Science and Technology, Hangzhou, 311231, China
Email: zhaoyinghuihust@gmail.com

***Abstract*—Conventional hardware based per-component and system-wide power management methods can save more power consumptions if they are in assistance with software-level adaptation. Since the conventional coarse-grained methods are not adaptive to various fluctuating workload in real scenarios, the system performance can be deteriorated greatly if the objective is only to minimize the total power consumptions separately, despite of the violations of Service Level Agreements (SLAs). In this paper a fine-grained job-level power aware scheduling algorithm is proposed to minimize power consumption in multi-processor system with SLA constraints. Simulation results show that the proposed algorithm can save significant power consumptions while still providing SLAs guarantees and the performance degradation is acceptable. The results also show that fine-grained job-level power aware scheduling can achieve better power/performance balancing between multiple processors than coarse grained methods.**

***Index Terms*—power aware computing system, job scheduling, service level agreements, power estimation, workload characterization**

I. Introduction

With active deployment of large multi-core servers to support thousands of concurrent jobs, the computing ability of future multi processor platforms will depend on the increasing numbers of cores. When the scales of multi processor system, power consumption has become the most important design consideration and the major bottlenecks to system scalability since higher power consumption results in more heat dissipation, cooling costs and makes servers more prone to failures. Researchers have proposed various per-component energy management approaches and solutions to reduce power and energy hotspots, such as CPUs, memories, and hard disks. However, conventional hardware based per-component and system-wide power management methods can not save considerable power consumptions because they are coarse-grained and not adaptive to various fluctuating workload in real scenarios. Moreover, the system performances, for example, availability, responsiveness, and throughput, do not scale with the number of processors but the power consumption dose. Most unfortunately, the whole system performance can be deteriorated greatly if the objective is to minimize the total power consumptions separately, despite of the violations of Service Level Agreement (SLAs) requirements.

Virtualization offers management capabilities for service consolidation, isolation and power reductions. However, it is hard to coordinate SLAs requirements and power management decisions among multiple Virtual Machines (VMs).It is also desirable to schedule jobs among various VMs while still satisfying their SLAs requirements in response to changing data center conditions.

The literature of current power management schemes has mostly limited to mechanisms of DVS/DFS (Dynamic Voltage Scaling/ Dynamic Frequency Scaling) and not applicable for better performance such as load balancing and energy balancing. Due to the high density of service consolidation and the increasing number of users with heterogeneous requests, providing users with SLA guarantees and saving power consumption have become a crucial problem that needs to be addressed. In this paper a fine-grained job-level power aware scheduling algorithm is proposed to minimize power consumption in multi-processor system with SLA constraints.

The proposed algorithm responds to power supply constraint situations by using closed-loop policies to set a safe performance level and it consider the coordinated optimization of power consumption and SLA requirements together. It also attempts to address hotspots problem in multi-processor systems through job-

Corresponding author: Congfeng Jiang,email:cjiang@hdu.edu.cn redispathing and migrations. Simulation results show that the proposed algorithm can save significant power consumptions while still provides SLAs guarantees and the performance degradation is acceptable. The results also show that fine-grained job-level power aware scheduling can achieve better load balancing and energy balancing between multiple processors than coarse grained methods.

The remainder of this paper is organized as follows: In section 2 we analyze some related work on power aware computing system, and SLA constrained job scheduling in multi-processor systems. In section 3 we present a power aware job scheduling algorithm for multi-processor systems with SLA constraints. Then, in Section 4, we present simulation results and the effectiveness, practicality and performance of the proposed scheduling algorithm. We also compare the performance data with conventional power-unaware job scheduling algorithms or job scheduling algorithms without SLAs constraints in Section 4. In Section 5, we present experimental results in real-world application and discuss the relative performance and scalability of the proposed job scheduling algorithm. Finally, we summarize the work in Section 6.

II. Related Work

Power consumption has become a main concern for enterprise server system in order to achieve a high quality of service level in terms of, for example, availability and reliability. The adaptive power management has to ensure peak power safety but should also intelligently schedule individual jobs to computing resources to guarantee negotiated SLAs.

Since the overall system power consumption has strong relationship to processor resource usage, various excellent Dynamic Voltage Scaling/Dynamic Frequency Scaling (DVS/DFS) algorithms have been proposed to reduce the power consumption of processors [1, 2].

Memory is another source contribution for power consumptions in server systems. Existing techniques usually manage power for the main memory by passively monitoring the memory traffic and regulation. Some algorithms are proposed to predict when to power down which memory units and into which low-power state to transition [1,3,4].

In large scale data centers, server systems, or in the enterprise storage system, power consumption of hard disks is a critical issue where data-intensive applications exhaust disk storage extensively. Since energy consumptions directly affects hard disk drive reliability and system performance, reducing energy consumptions of disks can dramatically save overall enterprise IT costs [2, 5-9].

Mor Harchol-Balter et al [10] investigates the performance of task assignment policies for server farms, as the variability of job sizes (service demands) approaches infinity and they found that the Size-Interval-Task-Assignment policy (SITA), which assigns each server a unique size range, was inferior to the much simpler greedy policy, Least-Work-Left (LWL), for certain common job-size distributions, including many modal, hyper-exponential, and Pareto distributions.

The Service Level Agreement (SLA) is an electronic contract between a service user and a provider, and defines service quality like online time, response time, failure percentage, etc. From a general-purpose viewpoint, performance can be defined by SLA constraints for corresponding underlying workload heterogeneity [11].However, existing job scheduling algorithms are developed with consideration of overall system performance such as throughput, average response time, mostly ignoring power consumption and SLA guarantees. In this regard, we propose a novel scheduling strategies in this paper aimed at leveraging performance and power consumption for parallel applications running on multi-processor system.

In this paper, we propose a model for negotiating SLAs and a matchmaking algorithm based on service gains which is the ability to fulfill the service requestor requirements. In this model the SLAs negotiation is configured with the top-ranked service identified in the matchmaking phase. This model acts as a component of the Power and SLA-aware job scheduler that has the capability to predict the performance of the computing system it manages and allocate jobs in such a way that SLAs are satisfied.

Our research focuses on scheduling SLAs constrained parallel tasks and thus heuristics are applied to schedule parallel tasks to minimize power consumption and performance overheads. Unlike the existing scheduling algorithms that ignore all the power consumption of each task, the proposed algorithm schedules a task to proper processor if this scheduling can help in conserving power consumption. Our power-aware scheduling algorithm is conducive to balancing workloads and power consumptions of a set of SLAs constrained parallel tasks. We conducted extensive experiments using both synthetic benchmarks and real-world applications to compare our algorithms with two existing approaches. Experimental results based on simulated clusters demonstrate the effectiveness and practicality of the proposed scheduling algorithm.

III. Power Aware Job Scheduling with SLAs Constraints

The main idea of this paper is the intuition that in lower loaded periods, there is a potential to save power consumption by dynamically powering off part of or whole servers to address the actual computing demands. Under such lower-load conditions, an appropriate fine-grained job scheduling scheme can considerably reduce power consumption. In the meantime, under higher load condition, power aware scheduling can also schedule jobs properly to balance power consumption between various processors and avoid hotspots. Therefore, the proposed power aware job scheduling in this paper contains three parts: workload characterization and prediction, power consumption measuring and estimation, feedback control of power consumption through job scheduling with SLAs constraints. Fig.1 illustrates the control framework of the power and SLA-aware scheduling algorithm.

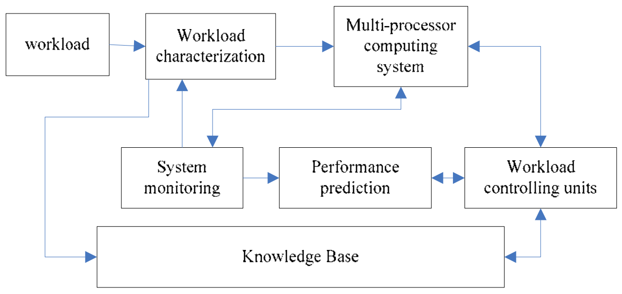

workload
Workload characterization
Multi-processor computing system
System monitoring
Performance prediction
Workload controlling units
Knowledge Base

Figure 1. Simplified system framework of feedback based controlling. The tasks of computing, sensing, and actuation are illustrated. The interaction among control tasks may affect the control performance.

Fig.1 shows that the controller will interact with different parts of the system since they are sharing resources such as CPU, memories, network, etc. Since the multi-processor computing system usually is connected by networks, deadlines of jobs may be missed. Moreover, the performance requirement of each control loop should be satisfied respectively.

According to the control theory, the framework of feedback based controlling in Fig.1 can be considered as a classical multi-input, multi-output control system. The controlling system can be defined as follows:

$$\dot{x}(t) = f[x(t),u(t),t], t \in [t_0,t_f] \quad (1)$$

Where:

x is the system state vectors, $x \in R^n$;

u is system controlling vectors and $u \in R^m$

In a deterministic state $x(t_0) = x_0$, Eq.1 has a unique solution, i.e. $x(t)$ given that u is preset and known.

Assume that the controlling system is a linear system, and the system can be defined as follows:

$$y(t) = \int_{t_0}^{t} G(t,\tau)u(\tau)d\tau \quad (2)$$

Where $G(t,\tau)$ is a $q \times p$ unit pulse response matrix:

$$G(t,\tau) = \begin{bmatrix} g_{11}(t,\tau) & g_{12}(t,\tau) & \dots & g_{1p}(t,\tau) \\ g_{21}(t,\tau) & g_{22}(t,\tau) & \dots & g_{2p}(t,\tau) \\ \vdots & \vdots & \vdots & \vdots \\ g_{q1}(t,\tau) & g_{q2}(t,\tau) & \dots & g_{qp}(t,\tau) \end{bmatrix} \quad (3)$$

The Service Level Agreements Constraints can be summarized as follows:

$$\psi[x(t_f),t_f] = 0 \quad (4)$$

Where $\psi \in R^r, r \le n; x(t_f) \in \psi(\cdot)$

Here we use a PID controller to keep the scheduling satisfying the SLA requirements and saving more power consumptions. This controller is specified by the following equations:

$$U = K_p \cdot e + K_i \cdot e + K_d \cdot \frac{du}{dt} \quad (5)$$

Where:

e is the control error and it is the difference between the measured control value, i.e., QoS gains and makespan of all the jobs. K_p, K_i, and K_d are the proportional, integral and dividal gains of the controller respectively. K_p produces a controller input proportional to the current control errors. K_i accumulates the control error and produces a control input based on historical control errors.

We will describe these parts in detail in the following.

A. Workload Characterization and Prediction

Workload characterization is an important technique to help understand the performance of system applications and demands. A good understanding of the workload of multi processor computing system should provide insights about user requests, program activities and help in improving the quality of the service provided to users. In a real computing system, workloads are varying dynamically due to various reasons such as user behaviors, usage modes of diverse applications, unforeseen load fluctuations, etc. In multi processor system, the workload of a processor dynamically changes in nature, because there is no effective and accurate way to predict when and where a task will arrive for service. Therefore, workload characterization acts as a crucial component of any performance analysis process and researchers often use processors runtime statistics, trace data of real workload and algorithms such as online learning, clustering, and fractal, to predict the workload of different applications in the future. However, since there are no specific workload characterization algorithms that can characterize every kind of workloads in different computing systems and applications, workload characterization and prediction schemes must be developed for specific individual application.

Depending on the type of application tasks, the workload characterization model could vary in complexity ranging from a simple specification of the task demands of the server resources, especially the processor cycles, to a detailed mathematical model capturing the flow of control during task execution. We assume that tasks use the hosts to execute some logic requiring a given amount of CPU time. Whenever a task is processed at a host, the actual execution time is compared with the current estimate of the task demand and if it is lower the estimate is updated. Thus, as the task progresses, the estimated task execution time is iteratively set to the more accurate observed execution time on the respective host. It is expected that during periods of lower workload intensity, the observed execution time will be close to the estimated task execution time. This approach would work provided that the time spent waiting is insignificant compared to the time spent executing the tasks. In this paper, we proposed a lightweight resource-oriented workload characterization scheme for multi processor system with parallel workloads which describes the consumption of system resources by the workload, such as length of processor queue, processor time, waiting time, disk operations, memory, etc. In resource-oriented workload characterization scheme, resource data can be collected easily by automated agents or third party packages which use the OS calls or APIs. The power consumption is measured by a multimeter and the results are transported to the dedicated computer through a USB connection.

We monitor the CPU and memory utilization during task execution and use it to split the measured observed time into time spent using the CPU and time spent waiting. This algorithm is conservative in that it starts with conservative estimates of the CPU cycles and refines them iteratively in the course of the execution. Therefore, in the beginning of the execution, its resource requirements would be overestimated possibly leading to rejecting or dissatisfying the performance such like throughput even though they could have been accepted without violating SLAs. This scheme can be used for tasks that run for the first time and there is no previous execution information available in the knowledge base. Once a new type of task set has been scheduled and processed, the estimates can be registered in the knowledge base and used as a starting point in future executions. Moreover, the estimates can be further refined as new tasks are executed.

The workload characteristics can be represented by observations in macro and micro scale. The former includes resource-oriented performance, such as response time, processor time, memory utilization, disk operations, etc. And the latter includes processor instructions execution time, status of registers, status of threads, hardware counters, etc. The workload can be characterized by interpreting these macro and micro performance tips and metrics. For example, there are 48 event counters and 18 performance counters, among which are three fixed performance counters in Intel® Pentium ™ IV processor, i.e., INSTR_RETIRED.ANY, Counter 4: CPU_CLK_UNHALTED.CORE, Counter 5: CPU_CLK_UNHALTED.REF. Therefore, the un-halted CPU time can be calculated as follows:

$$Time_{un-halted} = \frac{\text{CPU\_CLK\_UNHALTED.TOTAL\_CYCLES}}{\text{ProcessorFrequency} \times \text{NumberofCores}} \quad (6)$$

B. Power and SLAs Aware Job Scheduling

In this paper, a host or site refers to a PC, cluster, or a supercomputer and the term is used interchangeably. And the terms job, task, and application are used interchangeably to refer to a request made by a user to run a given application with QoS requirements or a given inputs. The following assumptions are made in this paper:

(1) The workload is heavy-tailed, as is characteristic of many empirically measured multi-processor computing system workloads.

(2) The applications have been divided into sub tasks and each sub task is independent. This assumption is commonly made in the job scheduling for heterogeneous distributed environment (e.g., [12-14]). The method we propose is applicable for tasks with no internal parallelism. Note that scheduling dependent jobs with sharing files, DAG (Directed Acyclic Graph) topologies or precedence constraints can be found in the literature but is out of the scope of this paper.

(3) The arrival rate of jobs is Poisson distributed.

(4) The estimates of expected task execution times on each machine in the multi processor computing system are known. The assumption that these estimated expected times are known is commonly made when studying scheduling algorithms [12, 14]. Approaches for doing this estimation include intrinsic and extrinsic factors method [15], analytic benchmarking [16], etc. For example, through code profiling, the execution time of a job can be calculated by the total clock period number required by it. Also note that the required execution time is in inverse proportional to the processor speed. Since the processor speed may be adjusted, the execution time required for the same task at the highest processor speed and the lowest is different. As the processor speed descends a half or the third, the task can be finished before its deadline yet, however, the energy consumption may be the quarter or the ninth of the original.

(5) The execution time of jobs on specific processor is proportional to the power consumption that the jobs consumed when the frequency or voltage is fixed.

(6) The execution sequence of tasks on a processor is FCFS (First-Come, First-Served).The host executes one task at a time and the task is not preemptive.

(7) Without loss of generality, we assume that the system components may fail and can be eventually recovered from failures. Both hardware and software failures obey the fail-stop failure mode.

(8) Job failures can occur online at any time and the total number of faulty hosts in a given multi processor computing system may never exceed a known percentage. And the sites failures are independent from each other.

(9) When the primary scheduler fails, there are backup schedulers to take over all the work of the primary scheduler.

(10)There is a job table maintained by a server and all the information of running jobs can be queried through this table.

Let $J = \{J_i \mid i = 1, 2, 3, ..., n\}$ denote jobs set, $J_i = (a_i, b_i, e_i, c_i, s_i, Q_i)$, M is set of processors, $M = \{M_j \mid j = 1, 2, 3, ..., m\}$, $M_j = (p_j, f_j, d_j, BW_j, PW_j)$.

Where:

a_i is arrival time of job J_i , b_i is starting time of job J_i , e_i is the average execution time of job J_i on all the processors, i.e. expected executed time on processor M_j where there is no other running jobs except for J_i , $e_i = (e_i^1, e_i^2, e_i^3, ..., e_i^m)$, c_{ij} is the expected completion time of job J_i on processor M_j , s_i is the size of data needed by job J_i (MB) , Q_i is the QoS values of job J_i .

As for hosts set M, p_j is the speed of processor M_j (MHz), f_j is the available memory capacity of host M_j (MB), d_j is the available disk space on host M_j (MB), BW_j is the bandwidth of host M_j (Mb/s), PW_j is the level of power consumption of host M_j, $PW_j = (PW_j^1, PW_j^2, PW_j^3, ..., PW_j^u)$, $\forall v \in [1,u], 0 < PW_j^v < 1$, where 0 stands for the lowest and 1 the highest.

Let Q denote a set of Quality of Service constraint $Q = \{Q_i \mid i = 1,2,...,n\}$, $Q_i = (T_i, R_i, S_i, A_i, P_i)$, where:

T_i (Timeliness) is timeliness requirement, i.e., the total expected completion time for running jobs, starting time of running jobs, and the deadline of running jobs;

R_i (Reliability) is reliability requirement, i.e., the probability of job completion;

S_i (Security) is security requirement. Since jobs may be executed at remote hosts in a distributed multi-processor system, various jobs and data need different security requirement, including confidentiality and integrity requirement;

A_i (Accuracy) is accuracy requirement. Due to the hardware limitations, some float computation may result in accumulated errors, which lead to the final non-accurate computing results. Therefore, jobs must be scheduled to hosts where they can be satisfied by the data accuracy requirement.

P_i (Priority) is priority requirement. In multiprocessor computing system, numerous jobs contents for limited computing resources. Therefore, jobs with higher priority values must be scheduled to dedicated hosts be the lower ones.

For simplicity, we use discrete values to modeling the Quality of Service constraints, i.e., the Quality of Service constraint is presented by several levels like very low, low, medium, high, and very high, not a specific number like 10% or 90% because in real computing system with user interaction a user only cares the interactive experience, not the specific performance numbers.

Let $ec^J = (ec_1, ec_2, ..., ec_m)$ denote the power consumption of m threads, and the matrix of n performance counters in m threads is $C = [c_{i,j}](1 \le i \le m, 1 \le j \le n)$.

Therefore, in respect to the power prediction modeling, $\exists\ x^J = (x_1, x_2, ..., x_n)$, given that

$$\begin{cases} \min(\| C \cdot x - ec \|_2) \\ C \cdot x - ec \ge 0 \end{cases} \tag{7}$$

Due to the heterogeneity of multi-processor system, power consumption among multi-processors is heterogeneous. And the heterogeneity index of job set J is

$$H^J = \frac{1}{|J|} \sum_{J_i \in J} \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(\frac{1}{n} \sum_{j=1}^{n} \frac{e_i^j}{\min_{k=1}^{n}(e_i^k)} - \frac{e_i^j}{\min_{k=1}^{n}(e_i^k)}\right)^2} \tag{8}$$

Similarly, heterogeneity index of QoS is

$$H_J^M = \frac{1}{|T|} \sum_{T_i \in T} \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(\frac{1}{q} \sum_{j=1}^{q} Q_i^j - Q_i^j\right)^2} \tag{9}$$

Where q is the number of the length of the vector.

We define G_i^j is the gains of QoS of job J_i on host M_j, i.e.

$$G_i^j = \sum_{k=1}^{q} w_i^k \cdot g(Q_i^k, V_j^k) \tag{10}$$

Where w_i^k is the weight of different QoS requirements of job J_i, and $\sum_{k=1}^{q} w_i^k = 1$; $g(Q_i^k, V_j^k)$ is the kth gain of QoS requirements of job J_i:

$$g(Q_i^k, V_j^k) = \begin{cases} Q_j^k - V_i^k & \text{, when } V_j^k \ge Q_i^k \\ 0 & \text{, when } V_j^k < Q_i^k \end{cases} \tag{11}$$

Where V_j^k is the available QoS capacity of the corresponding host.

Intuitionally, the more the gains of QoS satisfactions, the more jobs can be executed with QoS guarantees. For specific users it is to maximize their gains of QoS satisfactions. However, from a system point of view, the situation is conversed. And the maximization of gains of QoS among various jobs leads to the QoS contention, performance degradation, and workload imbalance.

We define D_i as the available theoretical scheduling set of job J_i with QoS satisfactions, $\exists D_i = (D_i^j \mid G_i^j > 0)$

In order to avoid the QoS contention, the gains of QoS satisfactions must be minimized while still guarantying the QoS requirements. Assume that

$$\begin{aligned} OP_i &= (D_i^j \mid G_i^j > 0) \cap \min\left\{G_i^j\right\} \\ &= (D_i^j \mid G_i^j > 0) \cap \min\left\{\sum_{k=1}^{q} w_i^k \cdot g(Q_i^k, V_j^k)\right\} \end{aligned} \tag{12}$$

Then the objective function for power and QoS constrained scheduling for job set J_i is

$$S = \min\left\{\sum_{J_i \in J} OP_i\right\} \cap \min(\max\{_{T_i \in T}(c_i)\}) \cap \min(H^J) \cap \min(H_J^M) \tag{13}$$

Eq.13 is NP-hard and can be solved by heuristics scheduling. We propose a heuristics scheduling algorithm, PaSLA, to solve this problem of power aware job scheduling with SLA constraints. The PaSLA algorithm is triggered by a scheduling event. When the number of jobs in the job set becomes a fixed maximum number, like 5, we call this a scheduling event. A job is submitted to the primary scheduler and the backup scheduler respectively. A pseudo code of PaSLA is demonstrated in the following.

```
1. When scheduling event occurs {
2. for each task in J
3. Compute $Q_i$ , $G_i^j$
4. End for
5.Compute estimated power consumption of each job on specific processor through code profiling
6. for each task in J
7.If the tasks have non-negative QoS gains
8.If the tasks are left from the last scheduling set
9.Sort the tasks by QoS gains and power consumptions
10.Schedule the tasks with lower QoS gains first
11. If tasks have equal non-negative QoS gains
12. Schedule the task to processors with lower power consumptions
13. End if
14. Delete the task from J
15. Update job table
16. End if
17. If tasks have equal non-negative QoS gains
18. Schedule the task to processors with lower power consumptions
19. Delete the task from J
20. Update job table
21. End if
22. Schedule the task with lower non-negative QoS gains first to processors
23. Update job table
24.If cpu_queue of targeted processor is exceeded the maximum length
25.Insert the task into next scheduling tasks set
26. Update job table
27. End if
28. Else
29.Insert the task into next scheduling tasks set
30. Update job table
31. End if
32. End for
33.  }
```

Figure 2. The pseudo codes of PaSLA

When scheduling event occurs, PaSLA first computes attributes of jobs such as arrival times, expected execution time, expected completion time, QoS gains, power consumptions, etc. The scheduler first schedule the tasks left from the last scheduling. After that, the scheduler chooses a set of candidate processors with proper CPU length for job execution and orders the jobs for the job QoS gains and power consumptions. If the tasks have not non-negative QoS gains, the tasks will insert the tasks into the next job scheduling set for future scheduling and execution. Therefore, the complexity of this scheduling algorithm is O(n).

C. Power Balancing and Job Migration

Let $pw_{\max}$ denote the last available time of all processors, and the first is $pw_{\min}$, we define a power balance index as follows:

$$eb = \frac{pw_{\min}}{pw_{\max}} \tag{14}$$

eb can be any value is range [0,1]. If eb equals 1, it suggests that power is balanced among processors. And if eb equals 0 it suggests that power is imbalanced among processors and at least there if one processor which is not allocated any jobs. In the feedback control process, two thresholds, eb_l and eb_h , $eb_l < eb_h$ are selected. The scheduler sets eb to 0 before scheduling .Then jobs are scheduled until $eb > eb_h$ and then, queue aware scheduling is used until $eb < eb_l$ .Appropriate eb_l and eb_h can be got after several control cycles.

Job migration enables to move checkpointable and rerunnable jobs from one host to another. Through job checkpoint and restart, a migrated checkpointable job can be restarted on the new host from the point at which the job stopped on the original host. In this paper, the jobs must be migrated from the current host to another in the following cases:

(i) In the case of failures affecting hosts with pending jobs, if the host can't recover from the failures for a preset time, the jobs scheduled to it must be migrated.

(ii) If there are many hosts whose processors are far lower utilized, the jobs must be migrated to hosts in that the hosts running jobs may keep a moderate utilization level and to keep some hosts idle to save more power consumptions. For the CMOS based processor, decreasing the supply voltage can reduce the power consumption greatly. Dynamic Voltage Scaling (DVS) technology is mainly proposed for reducing the power consumption of processor by adjusting the processor speed and supply voltage. In this paper, the multi processor computing system with adjustable supply voltage processors, which uses special instructions for the voltage control and adjusts its voltage discretely, is considered. Moreover, the operating system can use the instructions of the voltage control, such that the supply voltage can be altered when a task is scheduled or migrated.

(iii) If the jobs remain idle for a certain period of time, i.e., waiting for a long time, they are eligible for migration if there are hosts that match the jobs.

(iv)If a local user submits a job that requests a particular platform or specific software that does not exist in the local computing system, it must be migrated to a different site that does have a computing resource with that platform. However, we do not consider this resource availability problem.

In the first case, we use a server to monitor the heartbeat status of hosts through notification mechanism. In the former section we have mentioned that all the jobs are scheduled to the primary scheduler. There is another backup scheduler to backup all the job replications in case of host failures. Every host running a job will inform the execution status of the job to the primary scheduler periodically. If the primary scheduler receives a job completion message from the host, it will drop the job in the job set. In all other cases, the primary scheduler scans the job table periodically to see if some hosts didn't send job execution report. If an execution failure is detected, an alternative host will be selected.

In the second and third case, the scheduler migrates a running job to a different set of processors by performing the following actions:

Step1: Stops the job if it is running

Step2: Checkpoints the job if the job is checkpointable

Step3: Kills the job on the current host

Step4: Restarts or reruns the job on the first available host, inserting all pending jobs into the next scheduling task set.

Fig. 3 shows the pseudo codes of the power balancing and job migration algorithm.

```
1. Repeat {
2. If any host is failed
3. Insert the current running jobs into the task set for future scheduling
4. Update job table
5. End if
6. For each host in M
7. If it is under utilized
8. Stop the job if it is running
9. Checkpoint the job if the job is checkpointable
10. Kill the job on the current host
11. If there exist hosts that match the QoS requirements of migrated jobs
12. Restart the job on the matched hosts
13. Else
14. Insert all pending jobs into the next scheduling task set.
15. Update job table
16. End if
17. End if
18. End for
19.   }
```

Figure 3. Power balancing and job migration algorithm

First, when a host fails, the primary scheduler will insert the current running jobs into the task set for future scheduling and mark the failed host failed avoiding further job allocations.

If any host is under utilized, then the primary scheduler reschedules the replication to hosts that match the QoS requirements. If a host is found, a replication will be migrated to the host and both the backup scheduler and the job table will be updated accordingly. However, if no host is available, insert the task into the beginning of the next tasks set.

IV. Simulation Results and Performance Analysis

We use simulations and real workload experiments to study the performance of PaSLA scheduling algorithm. The common approach to study the performance of PaSLA scheduling algorithm is to compare it with a non-power-aware or non-SLA-aware scheduling algorithm. Thus we studied and compared the performance of the simple and frequently used heuristics such as EDF (Earliest Deadline First), Min-min [12], Max-Min, QoS guided Min-Min[17], Sufferage[18], and MCT (Minimum Completion Time) [18], with PaSLA by testing various scenarios.

To evaluate the PaSLA scheduling algorithm, we use the following metrics:

(1)*Makespan*: the total running time of all jobs;

(2)*Average waiting time*: the average waiting time spent by a job in the grid.

(3)*Scheduling success rate*: the percentage of jobs successfully completed in the system;

(4)*Power consumption*: the power consumed by the jobs.

(5)*Average violating rate of SLA*: the percentage of SLAs violation when scheduling user jobs out of total jobs.

(6)*Average migration rate:* the percentage of migrated jobs out of total scheduled jobs.

The scheduling steps of Min-min algorithm is described as follows:

Step 1: For each task in the task set, the machine that gives the task its minimum completion time (first Min) is determined (ignoring other unmapped tasks).

Step 2: Among all task-machine pairs found in Step 1, the pair that has the minimum completion time (second Min) is determined.

Step 3: The task selected in Step 2 is removed from the task set and is mapped to the paired machine.

Step 4: The ready time of the machine on which the task is mapped is updated.

Steps 1-4 are repeated until all tasks have been mapped.

The scheduling steps of Sufferage algorithm is described as follows:

Step 1: For each task in the task set, the machine mj that gives the earliest completion time is found.

Step 2: For each task in the task set, the Sufferage value is calculated. (Sufferage value = second earliest completion time minus earliest completion time).

Step 3: If machine m_j is unassigned then assign t_k to machine m_j, delete t_k from L, and mark m_j as assigned. Otherwise, if the sufferage value of the task (t_i) already assigned to m_j is less than the sufferage value of task t_k then unassigned t_i, add t_i back to L, assign t_k to machine m_j, and remove t_k from the task set.

Step 4: The ready times for all machines are updated.

The scheduling steps of MCT algorithm is described as follows:

Step 1: The first task in the task set is mapped to its minimum completion time machine (machine ready time plus estimated computation time of the task on that machine).

Step 2: The task selected in step 1 is removed from the task list.

Step 3: The ready time of the machine on which the task is mapped is updated.

Step 4: Steps 1-3 are repeated until all the tasks have been mapped.

A. Simulation Setup and Parameters Settings

Feedback control based scheduling algorithm is capable of modifying its own scheduling decision and program behavior through time, depending on the execution characteristics. Here, the objective of PaSLA is to schedule jobs to processors while guarantying the SLA requirements and minimizing the total power consumptions of the computing system, preserving its simplicity and low overhead. We test this algorithm and describe its behavior under a number of workloads. Simulations based on MATLAB ™ and MATLAB ™ Simulink ™ includes an analysis of the performance sensibility with the variation of the control parameters and its application in a multi processor computing system. In our simulations, we construct synthetic workloads using jobs with varying arrival rate and execution time. Moreover, in order to evaluate the robustness of our algorithm, we allow the jobs to follow different distributions, such as Gaussian, Possion, Uniform, Weibull, and heavy tailed distribution. We generate a wide range of workloads by varying the number of jobs and their execution times in our simulations. Specifically, we conduct over 1000 sets of experiments, and the number of jobs in each experiment is selected according to a specific distribution. The relative performance of an algorithm in each experiment is compared by normalizing its original values. Table 1 lists the key simulation parameters of one simulation.

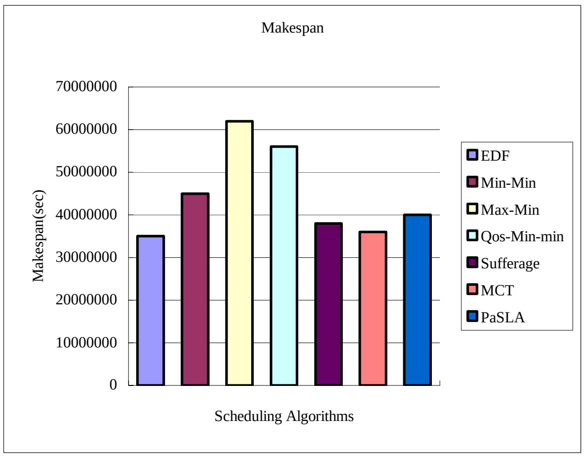
TABLE I.
REPRESENTATIVE SIMULATION PARAMETERS AND SETTINGS

Number of jobs	10,000,000
Number of processors	16
Site processing speed	8 nodes with 2.4GHz and 8 nodes with 1.8GHz
Job arrival rate	Poisson distributed in [0.3, 0.9]
Job execution time	Normal distributed in [0.1, 1000] sec
Sites failure rate	Poisson distribution with failure rate λ_{fail}

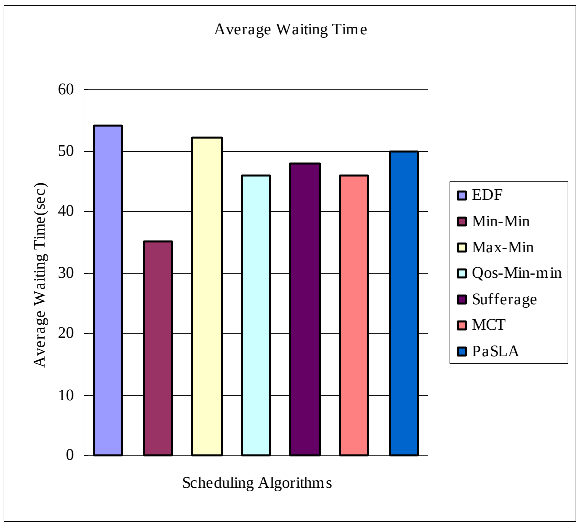
B. Simulation Results and Analysis

We assume that the hosts fail in a Poisson distribution with failure rate λ_{fail} and the jobs arrive in the queue in a Poisson stream with rate λ_{task}. Without loss of generality, every task set consist two kinds of jobs, i.e., jobs with SLA constraints and jobs without SLA constraints. We compare the relative performance such as makespan, average waiting time, scheduling success rate, power consumption, average violating rate of SLA, average migration rate. The simulation results are shown in Figure 4. All the data in the figures are mean values of 20 simulation results.

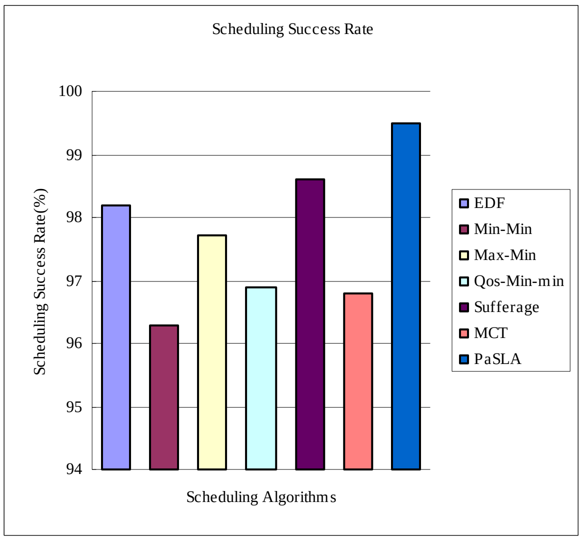
In Figure 4(a), the makespan order of the scheduling algorithms from maximum to minimum is: (1) Max-Min, (2) QoS-Min-min, (3) Min-Min, (4) PaSLA, (5) Sufferage, (6) MCT, and (7) EDF. The makespan of EDF is the smallest because of its smallest computation consumption. PaSLA dynamically schedules jobs to computing sites according to the real time power consumption and SLA constraints. Thus the makespan of PaSLA is relatively large.



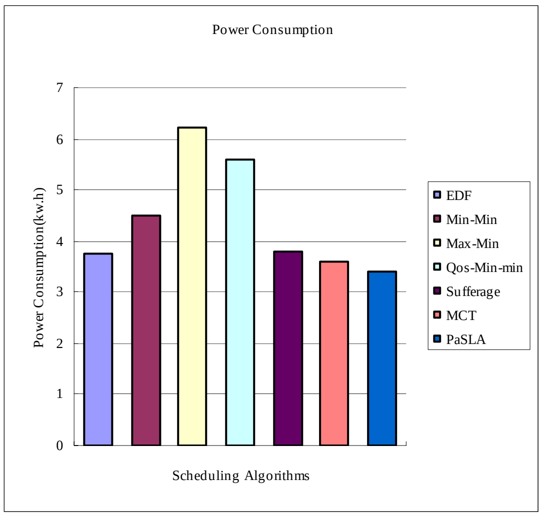
4(a) Makespan



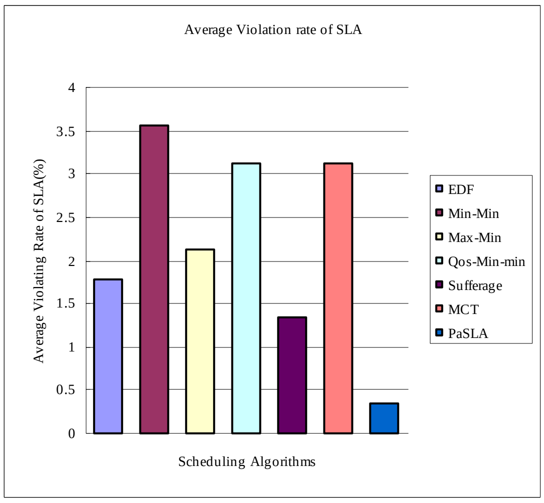
4(b) Average Waiting Time



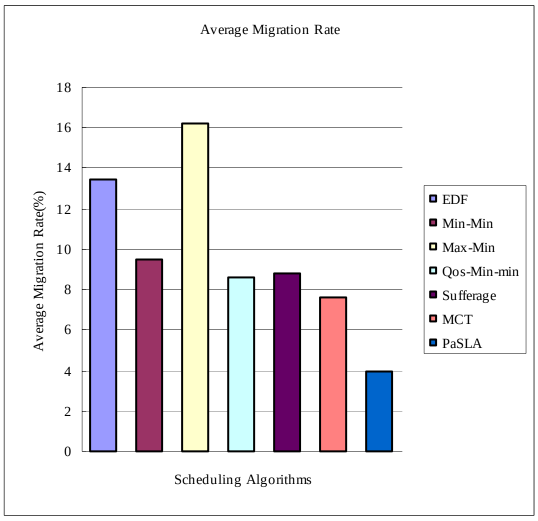
4(c) Scheduling Success Rate



4(d) Power Consumption



4(e) Average Violation Rate of SLAs



4(f) Average Migration Rate
Figure 4. Simulation results and relative performance

In Figure 4(b), the average waiting time order of the scheduling algorithms from maximum to minimum is: (1) EDF, (2) Max-Min, (3) PaSLA, (4) Sufferage, (5) QoS-Min-min and MCT, and (6) Min-Min. EDF has the longest average waiting time because it executes tasks without global information such as waiting times of other tasks. Consequently, EDF makes a significant increase of total execution time and makes the average waiting time longest eventually.

In our simulations, a task will be dropped if it couldn't be finished successfully after ten times. Thus, the scheduling success rate can't reach to 100%.In Figure 5(c), PaSLA has the highest scheduling success rate in a failure-prone multi-processor environment. PaSLA reschedules the tasks whose demand couldn't be satisfied on the current time when next scheduling event occurs. Thus, PaSLA increases the scheduling success rate significantly.

In Figure 4(d), the power consumption order of the scheduling algorithms from highest to lowest is: (1) Max-Min, (2) Qos-Min-min, (3) Min-Min, (4) Sufferage, (5) EDF, (6) MCT, and (7) PaSLA. PaSLA has the lowest power consumption because it takes into account the real time power consumption when scheduling tasks.

Since PaSLA also optimizes the SLA requirements and satisfactions while scheduling tasks, it has the lowest average violation rate of SLAs and the lowest average migration rate through Fig.4(e) and Fig.4(f).

The results in Figure 4 show that no single algorithm achieves the highest performance for all metrics. However, PaSLA exhibits relatively better performance with highest success rate, moderate level of makespan and average waiting time, lowest power consumption, average violation rate of SLAs and the lowest average migration rate due to its power aware and SLA aware scheme. We observe from Fig.5 that substantial performance improvements can be obtained. As the experimental results finally turn out, the performances of our algorithm is fairly insensitive to the distributions we choose.

V. Conclusions and Future Work

Power management is a key problem for IT management and operation in multi processor system, especially for Data Centers and cloud computing environments. Recent processor support for dynamic frequency and voltage scaling (DVS) makes it possible for software to affect power consumption by varying execution frequency and supply voltage. However, DVS and DFS are not enough for processor power reductions if they are issued only by entering a sleep mode. Therefore, it is feasible to coordinate hardware level and software and job level power management to save more power consumption. Moreover, fine-grained job-level power aware scheduling can achieve application specific performance SLA guarantees than system wide or per-component power management.

In this paper we analytically derive functions for real-time estimation of system power consumption using performance counter data on real hardware. We also developed our job scheduling model based on feedback control theory and characterize workload through data gathered from micro benchmarks and real time user applications to capture possible application behavior. The job scheduling model is independent of implementation platforms and therefore feasible for future applications on multi-processor systems. Power consumption is reduced and the SLA constraints are guaranteed by the proposed power-aware and SLA-aware job scheduler.

We study several job scheduling algorithms for different workload characteristics. An infrastructure for investigating scheduling performance is implemented. A fine-grained job-level power aware scheduling algorithm is proposed to minimize power consumption in multi-processor system with SLA constraints. Performance such as Makespan, Average waiting time, Scheduling success rate, Power consumption, Average violating rate of SLA, Average migration rate are assessed for different job scheduling algorithms. Measurements provides a quantitative assessment of the potential of energy savings for power and SLA aware job scheduling algorithms as opposed to conventional job scheduling algorithms that disregard power consumption and SLA constraints.. Simulations results show that the proposed algorithm can save significant power consumptions while still providing SLAs guarantees and the performance degradation is acceptable. The results also show that fine-grained job-level power aware scheduling can achieve better power/performance balancing between multiple processors than coarse grained methods.

In order to satisfy the SLA requirements of user applications, the status of the multiple sites and processors must be monitored and the performance data should be recorded. However, in a multi processor system, the collection of system performance data, the coordinated optimization of power consumption and SLA requirements will add on a large amount of computation and communication overhead. Thus, efficient system-wide and per-component monitoring and discovering technologies must be developed. Moreover, estimating power consumption is critical for OS process scheduling and for software/hardware developers. However, obtaining processor and system power consumption is non-trivial and using real platform simulators is time consuming and prone to error.

However, in a real multi processor system such as a data center, asking the users to fully specify their SLA requirements quantitively is an unreasonable burden. For example, user only need to specify a power consumption level such as low, middle, or high when executing jobs rather than the numerical values. Therefore, how to evaluate the qualitative and quantitative effects is a key factor that impacts the coordinated optimization of power consumption and SLA requirements heavily [19, 20]. However, global energy consumption mode is different in different systems with specific performance-oriented applications. Moreover, energy consumption mode is also different for different platforms with different performance constraints and SLA requirements and SLA satisfactions. It is an open problem to reduce energy consumption, while still meeting performance demands, system loads and reliability.

ACKNOWLEDGMENTS

The funding support of this work by State Key Development Program of Basic Research of China (Grant No. 2007CB310900), Natural Science Fund of China (NSFC, Grant No. 60873023, 60973029), Science and Technology Research and Development Program of Zhejiang Province, China (Grant No. 2009C31033),and Hangzhou Dianzi University Startup Fund (Grant No. KYS 055608058, KYS055608033) are greatly appreciated.

REFERENCES

[1] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar, “Critical power slope: understanding the runtime effects of frequency scaling,” In Proceedings of the 16th Annual ACM International Conference on Supercomputing (ICS'02),New York:ACM,2002,pp.35-44, doi:10.1145/514191.514200.

[2] W. Kim, M. Gupta, G. Wei, D. Brooks, “System level analysis of fast, per-core DVS/DFS using on-chip switching regulators,” In Proceedings of 14th International Symposium on High-Performance Computer Architecture (HPCA-14), Los Alamitos :IEEE Computer Society, 2008,pp.123-134, doi:10.1109/HPCA.2008.4658633.

[3] H. Huang, C. Lefurgy, T. Keller, and K. G. Shin, “Improving energy efficiency by making DRAM less randomly accessed,” In Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED’05), New York:ACM,2005,pp. 393-398, doi:10.1145/1077603.1077696.

[4] I. Hur, C. Lin, “A comprehensive approach to DRAM power management,” In Proceedings of 14th International Symposium on High-Performance Computer Architecture (HPCA-14), Los Alamitos :IEEE Computer Society, 2008,pp. 305-316, doi:10.1109/HPCA.2008.4658648.

[5] W. Jiang, C. Hu, Y. Zhou and A. Kanevsky, “Are disks the dominant contributor for storage failures? A comprehensive study of storage subsystem failure characteristics,” In Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08), Berkeley: USENIX, 2008, pp.111-125.

[6] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton and J. Wilkes, “Hibernator: helping disk arrays sleep through the winter,” ACM SIGOPS Operating Systems Review ,vol. 39 , pp. 177-190, December 2005, doi: 10.1145/ 1095809. 1095828.

[7] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, “DRPM: dynamic speed control for power management in server class disks,” In Proceedings of 30th Annual International Symposium on Computer Architecture (ISCA-30), Los Alamitos: IEEE Computer Society,2003,pp.169-179,doi:10.1109/ISCA.2003.1206998.

[8] C. Weddle, M. Oldham, J. Qian, and A. Wang, P. Reiher, G. Kuenning, “PARAID: a gear-shifting power-aware RAID, ”In Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07), Berkeley: USENIX, 2007,pp.245-260.

[9] L. Cai and Y.-H. Lu, “Joint power management of memory and disk,” In Proceedings of the Conference on Design, Automation and Test in Europe (DATE’05), Washington: IEEE Computer Society, 2005, pp.86-91, doi:10.1109/DATE.2005.192.

[10] Mor Harchol-Balter, A. Scheller-Wolf , A. Young, “Surprising results on task assignment in server farms with high-variability workloads,” In Proceedings of the 11th

international joint conference on Measurement and modeling of computer systems (SIGMETRICS'09) , New York:ACM,2009,pp.287-298,doi:10.1145/1555349. 1555383.

[11] Y. Guo, Solihin,L. Zhao, and R. Iyer, “A framework for providing service level agreements in chip multi-Processors, ”Proc. the 40th Annual IEEE/ACM Symposium on Microarchitecture (MICRO-2007), Washington: IEEE Computer Society ,2007,pp.343-355, doi:10.1109/MICRO.2007.6.

[12] T.D Braun, D. Hensgen, R. Freund, H. J. Siegel, “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, ” Journal of Parallel and Distributed Computing, vol. 61,pp.810-837,June 2001, doi:10.1006 /jpdc.2000.1714.

[13] S. Song, K. Hwang, and Y. Kwok, “Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling,” IEEE Transactions on Computers, vol. 55, pp.703-719, June 2006, doi:10.1109/TC.2006.89.

[14] J. Kim, S. Shivle, H.J. Siegel, A. Maciejewski, “Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment, ” Journal of Parallel and Distributed Computing, vol. 67,pp. 154-169, February 2007,doi: 10.1016/j.jpdc.2006.06.005.

[15] K. Jong and K.G. Shin, “Execution time analysis of communicating tasks in distributed systems,” IEEE Transactions on Computers, vol. 45, pp.572-579, May 1996, doi:10.1109/12.509908.

[16] M. A. Iverson. Ozguner and L. Potter, “Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment,” IEEE Transactions on Computers, vol. 48, pp.1374-1379, December 1999, 10.1109/12.817403.

[17] X. He, X. Sun, and G. von Laszewski, “QoS guided min-min heuristic for grid task scheduling,” Journal of Computer Science and Technology, vol. 18, pp.442-451, July 2003, doi: 10.1007/BF02948918.

[18] L.D. Briceño, M. Oltikar, H.J. Siegel, and A. A. Maciejewski, “Study of an iterative technique to minimize completion times of non-makespan machines,” In Proceedings of 2007 IEEE International Parallel and Distributed Processing Symposium (IPDPS’07), Los Alamitos: IEEE Computer Society, 2007, pp.1-14, doi:10.1109/IPDPS.2007.370325.

[19] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, X. Zhu, “No Power Struggles: Coordinated Multi-level Power Management for the Data Center,” In Proceedings of 2008 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XIII), New York:ACM,2008, pp.48-59,doi: 10.1145/1353535.1346289.

[20] R. Nathuji, A. Somani, K. Schwan, and Y. Joshi, “CoolIT: Coordinating Facility and IT Management for Efficient Datacenters,” In Proceedings of 2008 USENIX Workshop on Power Aware Computing and Systems (HotPower08), 2008.

Congfeng Jiang is a lecturer of School of Computer Science and Technology, Hangzhou Dianzi University, China. He is with the Grid and Service Computing Lab in Hangzhou Dianzi University.

Before joining Hangzhou Dianzi University, he was a PhD candidate in Huazhong University of Science and Technology from 2002 to 2007. He received his PhD degree in 2007. His current research areas include power aware computing system, virtualization, grid computing, etc.

Dr. Jiang is a member of China Computer Federation (CCF), IEEE and IEEE Computer Society.

Jian Wan is the director of Grid and Service Computing Lab in Hangzhou Dianzi University and he is the dean of School of Computer Science and Technology, Hangzhou Dianzi University, China.

Prof. Wan received his PhD degree in 1996 from Zhejiang University, China. His research areas include parallel and distributed computing system, virtualization, grid computing, etc.

Prof. Wan is a member of China Computer Federation (CCF).

Xindong You is a lecturer of School of Computer Science and Technology, Hangzhou Dianzi University, China. She is with the Grid and Service Computing Lab in Hangzhou Dianzi University.

Before joining Hangzhou Dianzi University, she was a PhD candidate in Northeastern University from 2002 to 2007. She received her PhD degree in 2007. Her current research areas include virtualization, distributed computing, etc.

Yinghui Zhao is a lecturer of Department of Hydraulic Engineering and Agriculture, Zhejiang Tongji Vocational College of Science and Technology, Hangzhou, China.

Before joining Zhejiang Tongji Vocational College of Science and Technology, she was an engineer in China Southern Power Grid Company. She received her Master degree from Huazhong University of Science and Technology in 2007. Her current research areas include remote sensing images processing, geographical information system (GIS), etc.