Energy-aware Parallel Task Scheduling in a Cluster

Lizhe Wang[1,2], Samee U. Khan[3†], Dan Chen[2†], Joanna Kołodziej[4], Rajiv Ranjan[5], Cheng-zhong Xu[6] and Albert Zomaya[7]

1 Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, P. R. China.
2 School of Computer, Chinese University of Geosciences, P.R.China.
3 Department of Electrical and Computer Engineering, North Dakota State University
4 Institute of Computer Science, Cracow University of Technology, Poland
5 ICT Centre, CSIRO, Australia
6 Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, P. R. China.
7 School of Information Technologies, The University of Sydney, Australia
†corresponding author: samee.khan@ndsu.edu and danjj43@gmail.com

Abstract

Reducing energy consumption for high end computing can bring various benefits such as, reduce operating costs, increase system reliability, and environment respect. This paper aims to develop scheduling heuristics and to present application experience for reducing power consumption of parallel tasks in a cluster with the Dynamic Voltage Frequency Scaling (DVFS) technique. In this paper, formal models are presented for precedence-constrained parallel tasks, DVFS enabled clusters, and energy consumption. This paper studies the slack time for non-critical jobs, extends their execution time and reduces the energy consumption without increasing the task's execution time as a whole. Additionally, Green Service Level Agreement is also considered in this paper. By increasing task execution time within an affordable limit, this paper develops scheduling heuristics to reduce energy consumption of a tasks execution and discusses the relationship between energy consumption and task execution time. Models and scheduling heuristics are examined with a simulation study. Test results justify the design and implementation of proposed energy aware scheduling heuristics in the paper.

Keywords: Cluster Computing, Green Computing, Task Scheduling

1. Introduction

Nowadays, high end computing facilities can consume a very large amount of power albeit they provide high performance computing solutions for scientific and engineering applications [38]. For example, operating a middle-sized data center (i.e., a university data center) demands 80000kW power [39]. It is estimated that computing resources consume around 0.5% of the world's total power usage [10], and if current demand continues, is projected to quadruple by 2020. Energy consumption for high performance facilities thus contributes to a significant electric bill. Additionally, high power consumption in general results in higher cooling costs. Furthermore, to allow computing facilities to operate on high power for a long time will lead to high temperature of computing systems, which further harms a system's reliability and availability. Therefore, reducing power consumption for high end computing becomes a critical research topic.

Modern processors are equipped with the Dynamic Voltage Frequency Scaling (DVFS) technique, which enables processors to be operated at multiple frequencies under different supply voltages. The DVFS technique thus gives opportunities to reduce the energy consumption of high performance computing by scaling processor supply voltages. Our research is devoted to developing scheduling heuristics which reduce energy consumption of parallel task execution by using the DVFS mechanism. A parallel task is a set of jobs with precedence constraints. Jobs in a parallel task may have some slack time for their execution due to their precedence constraints.

This paper makes a study on scheduling policies and application experiences to reduce power consumption of parallel tasks. Our first research issue is to minimize task execution time as well as reduce power consumption. The execution time of the non-critical jobs in a parallel task can be extended, thus giving an opportunity to scale down the supply voltages of processors. Based on the analysis of DVFS on non-critical jobs, we develop two power aware scheduling heuristics for parallel tasks, the Power Aware List-based Scheduling (PALS) algorithm and the Power Aware Task Clustering (PATC) algorithm.

Our second research objective is to make an study on energy and performance tradeoff for parallel task execution. The green Service Level Agreement (SLA) is introduced in this research. By negotiating with users via the green SLA, an energy-performance tradeoff algorithm is developed to reduce energy consumption with an affordable task execution time increase. We develop a simulation study on the proposed scheduling heuristics and make a performance evaluation.

We declare our contribution as follows:

- We develop formal models for parallel tasks and a power aware cluster and we also define the task scheduling issue.
- We develop two power scheduling heuristics for parallel

tasks: the PALS and the PATC.

- We present the green SLA use scenarios and propose a new scheduling heuristics for energy aware parallel task scheduling, which makes a study on the tradeoff between the energy consumption and task execution time (performance).
- We build a simulation study and performance evaluation on the proposed heuristics. Test results justify our design and implementation of energy aware heuristics.

The rest of this paper is organized as follows. Section 2 introduces background and related work. Then Section 3 discusses the models for parallel tasks, DVFS and compute clusters and Section 4 formally define the research issue of energy aware parallel task scheduling. Section 5 applies the DVFS technique on non-critical jobs of parallel tasks, which is the basis of the PALS and the PATC. We describe the scheduling heuristics of the PATC and the PALS in Section 6 and 7. Section 8 presents the Service Level Agreement with performance metrics of green computing and proposes the research issue of energy performance tradeoff for parallel task scheduling. Section 9 then presents the scheduling algorithm for the research issue proposed in 9. The complexity analysis for the proposed algorithms are presented in Section 10 and Section 11 describes a simulation study on the proposed scheduling heuristics. Finally this paper is summarized in Section 12.

2. Related Work

This section discusses background and related work of task scheduling, DVFS, and power aware cluster computing.

2.1. Parallel task scheduling

Task scheduling techniques in parallel and distributed systems have been studied in great detail with the aim of making use of these systems efficiently.

Task scheduling algorithms are typically classified into two subcategories: static scheduling algorithms and dynamic scheduling algorithms. In static task scheduling algorithms, the task assignment to resources is determined before applications are executed. Information about task execution cost and communication time is supposed to be known at compilation time. Static task scheduling algorithms normally are non-preemptive – a task is always running on the resource to which it is assigned [25]. Dynamic task scheduling algorithms normally schedule tasks to resources in the runtime to achieving load balance among PEs. are based on the redistribution [9, 41].

List scheduling algorithm is the most popular algorithm in the static scheduling [23, 31]. List based scheduling algorithms assign priorities to tasks and sort tasks into a list ordered in decreasing priority. Then tasks are scheduled based on the priorities. In this paper, we build a list based scheduling heuristic for parallel tasks – the PALS algorithm. The task execution information, such as task execution cost and communication cost, can be obtained by some profiling tools and compiler aides in advance.

The task graph clustering technique [20, 42] is an effective static scheduling heuristic for scheduling parallel tasks. Given a task graph, "clustering" is the process of mapping task graph nodes onto labeled clusters. All tasks of the same cluster are executed in the same processor. In traditional task scheduling heuristics, the process of clustering tasks is an optimization of reducing the makespan of the scheduled graph. In this paper, we proposed the PATC algorithm, whose process of clustering tasks is guided by reducing the total power consumption of the scheduled graph.

2.2. Energy reduction via DVFS techniques

Dynamic voltage and frequency scaling (DVFS) has been proven to be a feasible solution to reduce processor power consumption [17, 18]. By lowering processor clock frequency and supply voltage during some time slots, for example, idle or communication phases, large reductions in power consumption can be achieved with only modest performance losses. A DVFS-enabled cluster [38] is a compute cluster where compute nodes can run at multiple power/performance operating points. The DVFS techniques have been applied in the high performance computing fields, for example, in large data centers, to reduce power consumption and achieve high reliability and availability [14, 6, 7]. Popular DVFS-based software solutions for high end computing include:

- Scientific applications can be modeled with Directed Acyclic Graph (DAG) and the critical path is identified in for applications. Thus, it is possible to reduce energy consumption by leveling down the processor supply voltage during non-critical execution of tasks [34].
- Some work [13] builds online performance-driven runtime systems to automatically scale processor supply voltages.
- Some work applies DVFS during the communication phases of high performance computing, for example MPI [11, 24].
- In addition to parallel applications, virtual machine scheduling can also use DVFS [38].

Our research in this paper falls into the first category: scheduling DAGs on multiple processors in a cluster with DVFS techniques.

2.3. Power aware task scheduling

A lot of work has developed DVFS for task scheduling. For example, Yao et al [43] and Ali et al [29] discuss scheduling independent tasks with DVFS on a single processor, Wei et al [44] and Gruian et al [16] use DVFS to schedule dependent tasks on multiple processors, Martin et al [30] and Luo et al [27, 28] developed power aware task scheduling algorithm for real time systems. As our work is devoted to developing power aware scheduling algorithms for dependent tasks, we compare our work with related research in this topic.

Zhang et al [45], Martin et al [30], Schmitz [32], and Luo et al [28] schedule dependent tasks on real time, where the tasks normally are assigned with arrival time, deadline and max power consumption. In our research of energy aware high end computing, we don't have these restrictions on the tasks to be scheduled.

Zong et al [47] employ the similar DAG model and resource model with us and developed energy-aware duplication scheduling algorithms. This work however did not use DVFS technique to reduce power consumption, therefore their implementation certainly has some room to further reduce energy consumption if DVFS technology is employed when scheduling parallel tasks. Gruian et al [16] propose a list based low energy scheduling algorithm – LEneS. It smartly introduces enhanced task-graphs (ETG) and energy gain in the list based scheduling. Martin et al [30] develop a hybrid global/local search optimization framework for DVFS with simulated heating. LPHM [3] is a low power scheduling of DAGs to minimize task execution time. LPHM combines the heterogeneous earliest finish time with the DVFS technique. Zong et al [46] develops two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters: EAD and PEBD. Lee et al [22] propose an energy-conscious scheduling (ECS) heuristic for parallel tasks on heterogeneous computing systems. Kimura et al [21] use the same idea of extending task execution time by reclaiming slack times for non-critical jobs.

Costa et al present a multi-facet approach to reduce energy consumption in clouds and grids with users decisions consideration [8] and SLA-aware management for management Cloud resources [5], which enjoy similar ideas of user-defined and SLA-based energy management.

Compared to the above related research, our PALS algorithm not only considers minimizing the energy consumption in the scheduling algorithm, but also uses the concept of slack time for jobs in power Gantt chart to discuss the trade off between energy consumption and scheduling length. The PALS algorithm also concerns reducing voltages during the communication phases between parallel jobs. None of above research work discusses this aspect.

We propose a novel power aware scheduling algorithm based on task clustering – the PATC algorithm. The PATC algorithm merges tasks by zeroing communication links aiming to reducing power consumption, which is different scheduling philosophy from the list based heuristics.

3. System Model

This section provides the formal description for a DVFS-enabled cluster, parallel tasks, and performance models, which are employed as basis of the formal problem definition in Section 4 and the scheduling algorithms in Section 6 and 7.

3.1. DVFS Model

A DVFS-enabled processor can be operated on a set of supply voltages V and a set of processor frequencies F.

$$V = \bigcup_{1 \leq m \leq M} \{v_m\} \tag{1}$$

$$F = \bigcup_{1 \leq m \leq M} \{f_m\} \tag{2}$$

where,
v_m is the m-th processor operating voltage;
f_m is the m-th processor operating frequency;
$v_{min} = v_1 \leq v_2 \leq ... \leq v_M = v_{max}$;
$f_{min} = f_1 \leq f_2 \leq ... \leq f_M = f_{max}$;
$1 \leq m \leq M$, M is the total number of processor operating points.

3.2. Energy Model

The energy consumption of modern processor for job execution, ξ, can be divided into two parts, dynamic energy consumption $\xi_{dynamic}$, and static energy consumption ξ_{static} [19]. Static power consumption arises from running, bias and leakage currents. Dynamic power consumption arises from the charging and discharging of the circuit node capacitances found on the output of every logic gate.

$$\xi = \xi_{dynamic} + \xi_{static} \tag{3}$$

According to [12], the dynamic power consumption $P_{dynamic}$ is computed as follows:

$$P_{dynamic} = A \times C \times v^2 \times f \tag{4}$$

Where,
A is the percentage of active logic gates, which are charged dynamically;
C is the total capacitance load;
v is the supply voltage;
f is the processor frequency.

Then, we have:

$$\xi_{dynamic} = \sum_{\triangle t} P_{dynamic} \times \triangle t \tag{5}$$

where,
$P_{dynamic}$ is the dynamic power;
$\triangle t$ is a time period.

ξ_{static} is normally proportional to $E_{dynamic}$ [24]:

$$\xi_{static} \propto \xi_{dynamic} \tag{6}$$

Therefore the whole power consumption could be estimated as follows:

$$\xi \propto \xi_{dyanmic} \tag{7}$$

In conclusion, we have the performance model:

$$\xi = \sum_{\triangle t} (\delta \times v^2 \times f \times \triangle t) \tag{8}$$

Where,
δ is a constant determined by the PE.
v is the processor operating voltage during $\triangle t$;
f is the processor operating frequency during $\triangle t$;
$\triangle t$ is a time period.

3.3. Resource Model

A compute cluster normally contains multiple compute nodes, which are formally termed as Processing Elements (PEs) in a context of parallel computing. This paper makes a study on homogeneous clusters: all PEs of the cluster have the same processing speed or provide identical processing performance in term of MIPS (Million Instruction Per Second). A homogeneous cluster, C, contains K PEs. The k-th PE pe_k has two properties:

- $pe_k.v^{op} \in V$ is the processor operating voltage
- $pe_k.f^{op} \in F$ is the processor operating frequency

$1 \leq k \leq K$, K is the total number of PEs.

A cluster C is defined by its set of processing elements

$$C = \bigcup_{1 \leq k \leq K} \{pe_k\} \tag{9}$$

3.4. Parallel Task Model

A parallel task with precedence constrains is modeled as a Directed Acyclic Graph (DAG) – $T = (J, E)$:

- J: a set of jobs (nodes in a DAG)

  $$J = \bigcup_{1 \leq n \leq N} \{job_n\} \tag{10}$$

  where,
  job_n is a job in the parallel task J.
  N is the total number of jobs.

  A job, job_n, has 3 properties:

  – $weight$ is the instruction number of job_n.
  – t^{st} is the starting time of job_n.
  – t is the execution time of job_n. if job_n is executed on pe_k, the job execution time is calculated as follows:

  $$job_n.t = \frac{job_n.weight \times CPI}{pe_k.f^{op}} \tag{11}$$

  where, CPI is the number of cycles per instruction of pe_k. It is determined by both the hardware and software of the cluster C, for example, computer architecture and instruction set (ie, RISC or CISC). $job_n.t^0$ is the job_n's execution time when PE is running with the maximum frequency f_{max}. Equation 11 calculates job execution based on PE's operating frequency.
  – t^{end} is the end time of job_n. We have:

  $$job_n.t^{end} = job_n.t^{st} + job_n.t \tag{12}$$

  Based on Equation 11 and Equation 8, the energy consumption to execute job_n can be calculated as follows:

  $$\xi_n = \gamma \times v^2 \times job_n.weight \tag{13}$$

  where, γ is a constant determined by the cluster C, and irrelevant with the parallel task T. v is the PE supply voltage during the job_n's execution.

- E: a set of precedence constraints (edges in a DAG)
  E defines partial orders (operational precedence constraints) on J. e_{ij} is an edge between job_i and job_j, it means that job_i must be completed before job_j can begin, $1 \leq i, j \leq N$, $job_i, job_j \in J$. e_{ij} sometime can also be represented $job_i < job_j$.

  e has one property:
  $e_{ij}.cost \geq 0$, is the amount of data required to be transferred from job_i to job_j, $1 \leq i, j \leq N$, $job_i, job_j \in J$. Data are transferred from the PE where job_i is executed to the PE where job_j is executed.
  As we are studying a homogeneous cluster, without loss of generality, $e_{i,j}.cost$ can also be normalized as communication time. Now we discuss the relationship between $e_{i,j}.cost$ and PE's operating frequency. It shows in [24] that the energy consumption and communication cost as processor frequency varies for four common MPI calls when different size of data are transferred among PEs. From the experiment results we can see energy can be saved up to 31% with at most 5% communication time increase. In this paper, we ignore the communication time increase. In other words, when a PE's supplied voltage is scaled down, the data communication time remains unchanged.

4. Research Problem Definition

Here we firstly consider the best-effort scheduling research problem. Without damaging the performance of parallel task execution (task execution time), the best-effort scheduling algorithm tries to reduce the energy consumption for task execution.

Before we bring up the formal definition of the above research issues, the following term definitions are introduced.

- TST: Task Starting Time of T

  $$TST = \min_{1 \leq n \leq N} job_n.t^{st} \tag{14}$$

- TFT: Task Finish Time of T

  $$TFT = \max_{1 \leq n \leq N} job_n.t^{end} \tag{15}$$

- $makespan$: the schedule length of T

  $$makespan = TFT - TST \tag{16}$$

- $Schedule$: Task Schedule
  The $schedule_n$ of job_n is a mapping from job_n to a PE pe_k with task starting time $job_n.t^{st}$.

  $$schedule_n : job_n \rightarrow (pe_k, job_n.t^{st}) \tag{17}$$

The schedule of parallel task T, $Schedule$, is defined as:

$$Schedule = \bigcup_{1 \le n \le N} schedule_n \tag{18}$$

A feasible schedule of parallel task T keeps the partial orders between jobs in T.

Based on the above definitions, the best-effort scheduling issue is defined as: given parallel task T and a cluster C, find a feasible schedule $Schedule$, which 1) gives the minimum schedule length $makespan_{best}$ of T, and 2) reduce as much energy consumption as it can without increasing $makespan_{best}$.

5. Voltage scaling for non-critical jobs

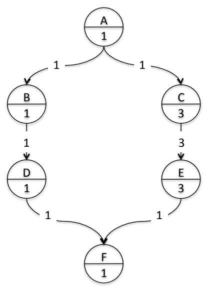

Figure 1: An example DAG

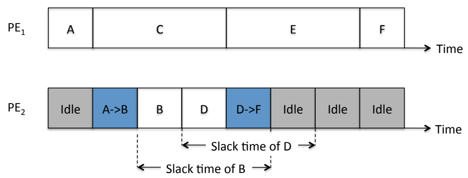

Figure 2: An example Gantt chart

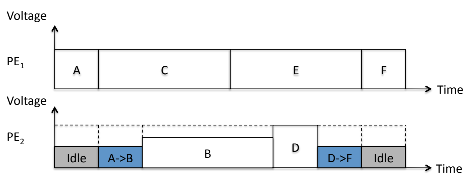

Figure 3: Example power Gantt chart

This section discusses how to scale down non-critical jobs' voltages with DVFS technique, which is the basis of the PATC and the PALS presented in the next two sections. Figure 1 is an example parallel task to be scheduled. In Figure 1, job IDs and job execution costs are marked inside the jobs and the communication costs are labeled on the links. The scheduled task graph is shown in Figure 2 as a Gantt chart. The Dominant Sequence (DS) of a scheduled task graph in a Gantt chart is a set of time slots of job execution and data communication from the first job to the last job, of which the sum of computation costs and communication costs is the *makespan*.

The DS in Figure 2 is "$A \rightarrow C \rightarrow E \rightarrow F$". It should be aware that a DS may across multiple PEs. As the best-effort scheduling algorithm does not extend the *makespan*, supplied voltages of PEs during the time slots of task execution and data communication in the DS is not changed. Supplied voltages of other time slots in a Gantt chart are considered be scaled down. For example, in Figure 2 jobs B and D have chance to extend their execution time and scale down their supplied voltages.

To discuss the algorithm for scaling voltages on non-critical time slots, we need to compute the slack time for a non-critical job. We have job_n's earliest start time is:

$$job_n.\overleftarrow{t^{st}} = \max_{\{job_m | job_m < job_n\}} \{job_m.t^{end} + e_{m,n}.cost\} \tag{19}$$

job_n's latest finish time is:

$$job_n.\overrightarrow{t^{end}} = \min_{\{job_l | job_l > job_n\}} \{job_l.t^{st} - e_{l,n}.cost\} \tag{20}$$

$\{job_m | job_m < job_n\}$ and $\{job_l | job_l > job_n\}$ are job_n's precursor set and successor set respectively. Then job_n's slack time can be calculated as:

$$job_n.slack = job_n.\overrightarrow{t^{end}} - job_n.\overleftarrow{t^{st}} \tag{21}$$

We can find in Figure 2 the slack time of job B and D.

Assume job_n is a non-critical job and is executed on pe_k. Then job_n's execution time can be extended to $job_n.slack$ without violating precedence constraints (without changing the finish time of its precursors and the start time of its successors). pe_k's operating frequency can be scaled to $pe_k.f^{op}$,

$$pe_k.f^{op} = f_{max} \times \frac{job_n.t^0}{job_n.slack} \tag{22}$$

where, $job_n.t^0$ is job_n's execution time when pe_k is operated with f_{max}. $job_n.t^0$ is discussed in Section 3.4 and can be calculated in Equation 11.

Algorithm 1 shows how to scale down non-critical jobs. For each PE, it scans all time slots (line 2–3). When the PE is idle or transfers data in a time slot, Algorithm 1 scales the PE's operating frequency to the lowest (line 4–6). When a time slot executes a non-critical job, it calculates its slack time, extends the job's execution time to the slack time, and scales down the PE's operating frequency to a proper value (line 7–9).

After we scale down the voltages of non-critical jobs in a scheduled task graph, the total power consumption can be calculated with the model defined in Section 3.2.

6. The PATC algorithm

We summarize several obvious rules to guide the design of the PATC algorithm and the PALS algorithm.

1. Equation 13 shows that given a certain task, a PE's supply voltage could be scaled down to a proper voltage to reduce the task's energy consumption. Certainly, this action may lead an increase of task execution time.

Algorithm 1 Non-critical time slot voltage scaling algorithm

BEGIN
FOR each PE pe_k DO
  FOR each time slot in pe_k's Gantt chart DO
    IF pe_k is idle or it executes a communication phase THEN
      scale down pe_k's operating frequency to lowest
    ENDIF
    IF pe_k executes a non-critical job job_n THEN
      calculate $job_n.slack$ as Equation 21.
    scale pe_k's frequency to $pe_k.f^{op}$ as Equation 22.
    ENDIF
  ENDFOR
ENDFOR
END

2. Research in [24] indicates that during the communication phase, the PE's supply voltage should be scaled down to the lowest level.
3. When a PE is idle (there is no task execution and data communication), its supply voltage should be leveled down to the lowest level.

This section presents the Power Aware Task Clustering (PATC) algorithm for parallel task scheduling. Traditional task clustering algorithm takes the following steps: 1) task clustering by zeroing edges, 2) cluster merging if the number of task clusters is greater than the number of PE, 3) task execution ordering in each task cluster, 4) each task cluster is allocated with a PE.

Traditional task clustering algorithm reduces the makespan by zeroing edges of high communication costs. Our Power Aware Task Clustering (PATC) algorithm, on the contrary, guides the edge zeroing process with objective of reducing power consumption. As shown in Algorithm 2, the PATC algorithm firstly marks all edges as unexamined and allocate each task a separate cluster. After sorting all edges in descending order of communication time, the PATC algorithm repeatedly merges tasks by zeroing the edges with high communication cost if the total power consumption is not increased. How to scale non-critical jobs' voltage and calculate the power consumption of a scheduled task graph have been discussed in Section 5.

Inside each cluster, tasks are executed in the order of their *b_level*. *b_level* is a normal priority assignment for jobs, which is defined as the length of a longest path from that job to the exit job. *b_level* is calculated with Algorithm 3.

7. The PALS algorithm

This section presents the Power Aware List-based Scheduling (PALS) algorithm for parallel tasks. The PALS algorithm (shown in Algorithm 4) firstly employs the ETF (Earliest Task First), a list-based scheduling algorithm (shown in Algorithm 5), to find the best-effort task response time for T. Then, it tries to reduce the energy consumption with the following methods:

Algorithm 2 The PATC algorithm

BEGIN
Initially all edges are marked unexamined and each task forms a separate cluster
Sort all edges in a descending order according to their communication costs
REPEAT
  Zero the highest unexamined edge in the sorted list if the power consumption of the scheduled task graph does not increase
  Mark the edge examined
  When two clusters are merged, the tasks are ordered according to their *b_level*.
UNTIL all edges are marked examined
END

Algorithm 3 *b_level* calculation

BEGIN
$r\_list \leftarrow$ a list of all jobs $Job_i \in J$ sorted in a reversed partial order
Initialize all jobs in *rtopo_list*: $b\_level(Job_i) \leftarrow 0$
FOR each Job $Job_i \in rtopo\_list$ DO
$max\_length \leftarrow 0$
  FOR each immediate succeeding job Job_j of job Job_i DO
    $length \leftarrow b\_level(Job_j) + e_{i,j}.cost$
    IF ($length > max\_length$) THEN
      $max\_length \leftarrow length$
    ENDIF
  ENDFOR
  $b\_level(T_i) \leftarrow Job_i.weight + max\_length$
  ENDFOR
END

- scale down PE's voltages to a proper level, thus extending the execution time of the non-critical jobs without affecting the critical path.
- scale the PE's voltage when it is idle or when it is in the data communication phase.

Algorithm 4 The PALS algorithm

1. schedule tasks via the ETF scheduling algorithm 5
2. scale down PE's voltages for all non-critical jobs with Algorithm 1

Given a parallel task T, the ETF algorithm [33, 40] is described in Algorithm 5. The Algorithm 5 allocates each job with a priority which can be calculated via different methods, for example, bottom level and top level [2]. In our implementation, we use the bottom level. The bottom level of a node (job) in a DAG is the longest path beginning with the node and the top-level is the longest path reaching the node. The length of a path is defined as the sum of the weights of its nodes and edges. Then, Algorithm 5 selects ready jobs with the highest priority and schedules it on the PE with earliest task starting time.

Algorithm 5 The ETF scheduling algorithm

$job_n.level$: priority of task $job_n \in J$
$ready\_job\_list$: list of jobs that are ready to be executed
PE_list: list of PEs
$pe_k.t^{available}$: PE's available time.
BEGIN
FOR each job $job_n \in J$ DO
    compute $job_n.level$
ENDFOR
put all ready jobs into $ready\_job\_list$
sort all jobs $job_n \in ready\_job\_list$ in decreasing order of $job_n.level$
put all PEs into PE_list
sort all PEs $pe_k.t^{available} = 0$
REPEAT
  IF ($ready\_job\_list \neq \emptyset$) THEN
    get a job, job_n, from $ready\_job\_list$
    get a PE, pe_k, which has the earliest available time $pe_k.t^{available}$
    schedule job_n on pe_k
    arrange the communicate phase, calculate starting time and finish time of job_n on pe_k
    delete the task from $ready\_job\_list$
    update PE_list with increasing order $pe_k.t^{available}$
  ENDIF
  update $ready\_job\_list$
UNTIL (every job $job_n \in J$ has been scheduled)
END

8. SLA management for Green Computing

In previous sections, we make a study on reducing power consumption without increasing task execution time, which is termed as the "best-effort scheduling issue". This section we analyze an interesting scenario: if a user is environmental respect and want to reduce power consumption by increasing its task execution time.

Green computing is a research topic to make computing with environmental concerns [37], for example, reduced energy consumption and reduced CO_2 emissions. We develop power aware scheduling for parallel task in the context of green SLA (Service Level Agreement for Green Computing). Users can specify not only performance requirements for computing services, but users can also specify green computing requirements for executing their jobs. We define the green SLA in three phases:

- Green SLA contract definition
  Our previous work [37] has summarized a number of green computing metrics, such as Data Center Infrastructure Efficiency (DCiE) [36], [4], Power Usage Effectiveness (PUE) [4], Data Center energy Productivity (DCeP) [15], Space Watts and Performance (SWaP) [1], storage, network, and server utilization. The green SLA contract definition phase creates various green SLA templates based on above green computing metrics. Typical metrics includes task response time, CO_2 emission, and power consumption. This phase also contains green SLA template publication and discovery.

- green SLA negotiation & monitoring
  Users develop their green SLA specification based on SLA templates and make a negotiation with computing resources, for example, a high performance cluster. Here are some examples of green computing service specifications:
  – Establish an execution service for x minutes if the total carbon emission of the service is below y tons.
  – I would like to accept z% task execution time increase to reduce w energy consumption.

- green SLA enforcement
  When a green SLA is reached, computing resources then execute the specified green services. For example, schedule tasks based on specified task execution time, CO_2 emission and power consumption. We develop energy aware scheduling algorithms for parallel tasks based on user's green SLA specifications.

Figure 4 shows the conceptual framework for green SLA based on energy aware scheduling in a cluster. Before a resource consumer submits a parallel job to a cluster, she/he firstly negotiates with a resource provider with normal performance metrics, like job response time, as well as with green metrics, for example, power consumption or CO_2 emission. After an agreement is reached, the user then submits his/her job to the resource. The resource provider then schedules the incoming job to an energy aware cluster to guarantee the green metrics and computing performance.

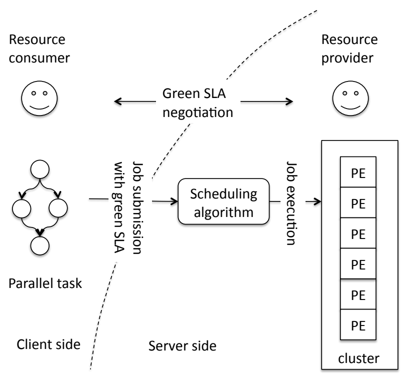

Figure 4: Concept framework for green SLA based energy aware scheduling in a cluster

With the green SLA negotiation, users agree to accept a tolerable performance loss, for example, additional 10% of task execution time, to reduce more energy consumption and make their computing more green. In contrast to the best-effort scheduling research problem, we term this research issue as the energy-performance tradeoff scheduling issue, whose main objective is to reduce energy consumption for task execution with an acceptable performance punishment.

The energy-performance tradeoff scheduling issue can be defined as:
given parallel task T, a cluster C, and the schedule length $makespan_{best}$, of a best-effort schedule, find a feasible schedule which tries to minimize energy consumption by giving Task Execution Time $makespan \leq (1+\eta)\times makespan_{best}$. $\eta > 0$ is the accepted task execution time extension, which is determined by the green SLA negotiation.

9. Energy-Performance Tradeoff Scheduling Algorithm via Green SLA

Now we discuss the energy-performance tradeoff problem: if a user agrees to tolerate an increase of his/her job execution time, for example, η of schedule length of the best-effort scheduling algorithm, how to schedule jobs to save more energy?

The energy-performance tradeoff algorithm is shown in Algorithm 6. It firstly gets the best-effort scheduling length via Algorithm 5. Then, it scales both the critical time slots in Algorithm 7 and non-critical time slots in Algorithm 1.

Algorithm 6 Energy-performance tradeoff scheduling algorithm

1. schedule tasks via the ETF scheduling algorithm 5
2. scale down PE's voltages for critical jobs with Algorithm 7
3. scale down PE's voltages for non-critical jobs with Algorithm 1

The Algorithm 7 firstly extends the critical time slots. Assume job_n is a critical job and it is executed on pe_k. It has been proved in [26] that distributing the free slack time "evenly" (proportional to the original critical time) is optimal as the power consumption is a convex function of PE frequency. Therefore job_n's slack time can be calculated as:

$$job_n.slack = job_n.t^0 \times \eta \tag{23}$$

Where,
$job_n.t^0$ is job_n's execution time when pe_k is operated with f_{max}.
η is the agreed extension of parallel task's execution time.

pe_k's operating frequency can be scaled to $pe_k.f^{op}$,

$$pe_k.f^{op} = f_{max} \times \frac{job_n.t^0}{job_n.slack} \tag{24}$$

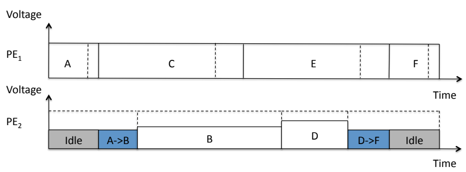

Figure 5: Energy-performance tradeoff power Gantt chart

10. Algorithm complexity analysis

In this section, we present an analysis on the time complexity of the algorithms discussed above.

Algorithm 7 Algorithm of voltage scaling for all time slots

```
1 BEGIN
2 FOR each PE pe_k DO
3   FOR each time slot in pe_k's Gantt chart DO
4     IF pe_k executes a critical job job_n THEN
5   calculate its job_n's slack time as Equation 23
6   scale pe_k's frequency to pe_k.f^op as Equation 24.
4   ENDFOR
3   FOR each time slot in pe_k's Gantt chart DO
4     IF pe_k is idle or it executes a communication phase THEN
5       scale down pe_k's operating frequency to lowest
6     ENDIF
7     IF pe_k executes a non-critical job job_n THEN
8       calculate job_n.slack as Equation 21.
9     scale pe_k's frequency to pe_k.f^op as Equation 22.
10      ENDIF
11   ENDFOR
12 ENDFOR
13 END
```

10.1. Analysis of the PTAC Algorithm

10.1.1. Algorithm 1

Algorithm 1 scales the supply voltage of a PE. Assuming we have K PE's, with t time slots, line 2 will occur at most K times, where as the inner loop starting at line 3 will occur t times. The operations from lines 4 - 10 are constant time operations, thus the upper bound of this algorithm is $O(Kt)$.

10.1.2. Algorithm 2

This algorithm forms the task clusters. Line 2 is executed $|E|$ times. The sorting in line 3 can be done in $|E|lg|E|$ time via quicksort. Lines 5 and 6 are constant time operations, each of which is part of a loop of $|E|$ iterations. Line 7 issues a call to Algorithm 3 when two clusters are merged. Since initially, each Task forms a cluster, we have C clusters and a total of T tasks. At most, Algorithm 3 will be called CT times. $O(|E| + |E|lg|E| + CT * A3)$ where $A3$ represents the complexity of the b_level calculation, or Algorithm 3.

10.1.3. Algorithm 3

This algorithm computes the b_level for a task. This algorithm is called by Algorithm 2. The sorting of line 2 can be done in $|J|lg|J|$ time. Line 3 is an initialization that occurs $|J|$ times. Lines 4 and 6 are a double loop, however each loop inner loop only iterates through a job J's children. Thus, the total number of iterations for lines 4-13 occurs $|E|$ times. Thus, Algorithm 3's complexity is $O(|J|lg|J| + |J| + |E|)$ Thus, our loose upper bound for the PTAC algorithm is $O(|E| + |E|lg|E| + C(|J|lg|J| + |J| + |E|))$

10.2. Analysis of the PALS Algorithm

10.2.1. Algorithm 4

Algorithm 4 simply executes algorithms 1 and 5. For example, Algorithm 5 will be executed T times, where T represents the total number of tasks.

10.2.2. Algorithm 5

This algorithm schedules the jobs of a task on to the PEs. Lines 1-4 are simply descriptions, or comments. Line 6-8 compute the priority for each job in the task, and execute N times, where N represents the number of jobs in the task. Line 9 is also execute N times, and simply adds jobs to a list. The sorting of the jobs in line 10 can be done in $O(NlgN)$ time. Line 11 is linear complexity, like line 9, and simply places the PEs into a list. This is done K times. The sorting in line 12 can be done in $KlgK$ time. The loop in lines 13-23 loops through each job in the list. This is done N times. Each operation between lines 13-23 can be considered to be done in constant time, for example, retrieving a job from the list in line 15 is constant. The complexity for Algorithm 5 is thus $O(N + NlgN + K + KlgK)$.

10.2.3. Algorithm 6

Algorithm 6 represents the energy-performance tradeoff algorithm. Line 1 makes T calls to algorithm 5, where T represents the number of tasks. Likewise, in lines 2 and 3, algorithm 6 calls algorithm 7 and algorithm 1 T times. Thus, the complexity of algorithm 6 is $O(T(Kt+N+NlgN+K+KlgK+A7))$ where $A7$ represents the complexity of algorithm 7.

10.2.4. Algorithm 7

Algorithm 7 scales down the voltage for the critical path, thus increasing the execution time of the task as a whole. The outer loop on line 2 is executed K times for the K PEs. The first inner loop on line 3 gets executed t times, where t represents the number of time slots in the Gantt chart. Lines 4, 5, and 6 are constant time operations.

The second inner loop has t loops, for each time slot in the Gantt chart. Thus, the complexity for the PALS algorithm is $O(Kt + Kt)$, or simply $O(Kt)$

11. Performance Study With Simulation

We make a simulation study on the proposed best-effort scheduling algorithm and energy-performance tradeoff scheduling algorithm. Several task sets are generated with the Synthetic DAG generation tool [35]. We simulate a cluster with multiple Turion MT-34 processors, whose operating points are shown in Table 1.

Table 1: Operating points for the Turion MT-34 processor

Frequency (GHz)	Supply Voltage (V)
1.8	1.20
1.6	1.15
1.4	1.10
1.2	1.05
1.0	1.00
0.8	0.90

In this simulation for best-effort scheduling, we are interested how much energy is saved given various parallel tasks and PE numbers in the cluster. We define the resource competition to execute a parallel task, $\zeta(T)$, in a cluster as follows:

$$\zeta(T) = \frac{N}{P} \tag{25}$$

where, T is the parallel task, N is the job number of T, and P is the PE number for executing T. Resource competition shows the task execution situation, like how many precedences exist between jobs, how many jobs are scheduled, and how many jobs are executed on each PE.

Table 2: Comparison of energy savings between different energy aware scheduling algorithm

Energy aware DAG scheduling algorithm	Maximum energy saving
EADUS & TEBUS [47]	16.8%
Energy Reduction Algorithm [21]	25%
LEneS [16]	28%
ECS [22]	38%
PATC	39.7%
PALS	44.3%

The PATC and the PALS can achieve up to 39.7% and 44.3% energy saving respectively in the simulation. Table 2 compares our algorithm with other energy aware DAG scheduling algorithms in term of max energy saving. EADUS & TEBUS [47] uses the duplication strategies for scheduling DAG based parallel tasks in a cluster to reduce power consumption. However, EADUS & TEBUS do not use DVFS to reduce energy consumption, thus leading less energy savings. Compared with LEneS [16], Energy Reduction Algorithm [21], and ECS [22], the PATC and PALS can achieve more energy saving as

- The PATC and PALS reduce the energy consumption during the communication phase
- The PATC and PALS reduce power consumption when a PE is idle, and
- The PATC and PALS try to extend job slack time whenever it is possible.

Figure 6 shows the energy savings when running the PALS algorithm in different scenarios of numbers of PEs and resource competition. For a close view, Figure 7 and Figure 8 shows two special cases of 1) Energy savings when running the PALS algorithm with different scenarios of resource competition and PE number is set as 50; 2) Energy savings when running the PALS algorithm with different PE numbers and resource competition is set as 6. From above figures we can see that the energy saved increases as the number of PEs increases. This can be explained as follows: when the number of PEs increases, intuitively there are less jobs executed in a PE, then the jobs have more of a chance to scale their execution time and PE supply voltages. If we fix the number of PEs, the energy saving

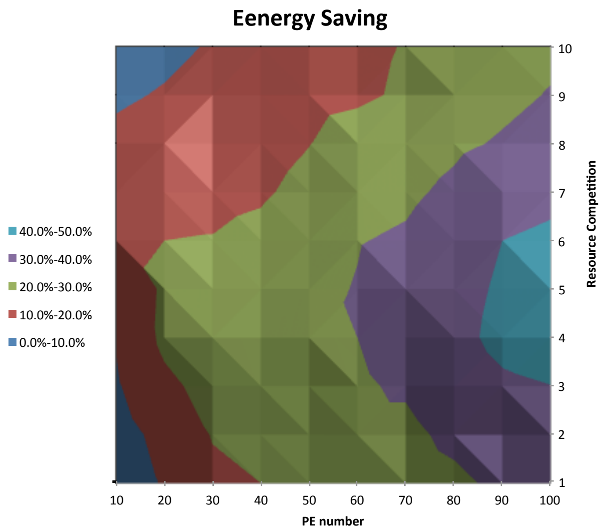

Figure 6: Energy savings of best-effort scheduling algorithm

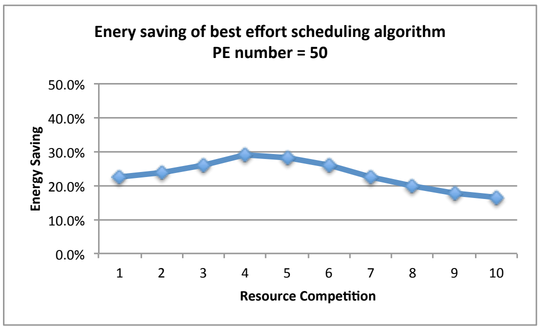

Figure 7: Energy savings of best-effort scheduling algorithm (PE number =50)

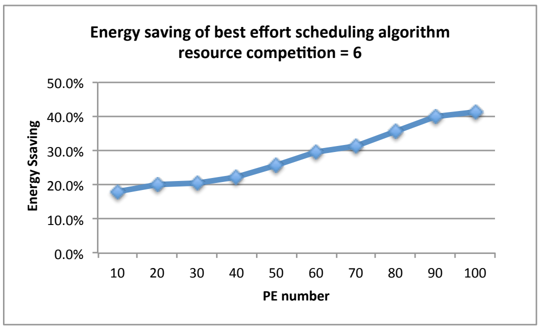

Figure 8: Energy savings of best-effort scheduling algorithm (Resource competition = 6)

firstly increases, achieves it maximum value, and then it deceases. This can be explained by the fact that the percentage of jobs on the critical path firstly increases then decrease. The length of critical path gives the limit that non-critical jobs can extend to.

In the simulation for energy-performance tradeoff scheduling, we are more interested in the relationship between the energy saved and the extended task execution time, as shown in Figure 9. From Figure 9 we can see that:

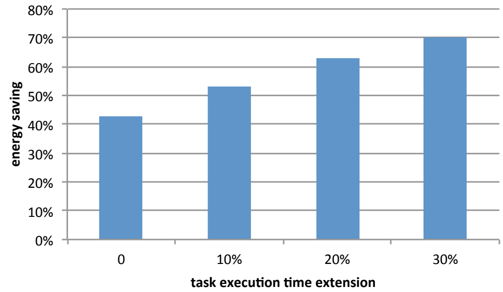

Figure 9: Energy savings vs. makespan extension

- When the makespan extension increases, the energy savings also increase.
- Then energy savings increase much when the makespan extension is less then 30%.

These observations can conclude that the green SLA negotiation is feasible. When users pay additional tolerant task execution time, which is less than 30%, less than 70% energy savings can be achieved. This is a win-win game.

12. Conclusion and Future Work

Recently, the need for efficient algorithms to minimize wasted server energy has become increasingly important. Dynamic voltage and frequency scaling (DVFS) technique has proven to be a highly effective technique to achieve low power consumption for high performance computing by dynamically scaling processor speed. We develop our research on minimizing energy for precedence-constrained parallel task execution. This paper proposes two scheduling algorithms in DFVS-enabled clusters for executing parallel tasks: the PATC and PALS. The proposed algorithms search the slack time for non-critical jobs without increasing scheduling length. We also develop green SLA based mechanism to reduce energy consumption by return users tolerant increased scheduling makespans. The proposed scheduling algorithm is examined via a simulation study. Test results show that the scheduling algorithm is efficient to reduce the power consumption of a DVFS-enabled cluster. Future work includes the deployment of the power aware scheduling algorithm in some real applications, for example, the the sparse Cholesky decomposition.

Acknowledgement

Dr. Lizhe Wang is supported by “One-hundred talent” program of Chinese Academy of Sciences.

[1] SWaP (Space, Watts and Performance) Metric. Web Page.

[2] Ishfaq Ahmad, Yu-Kwong Kwok, and Min-You Wu. Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors. In *ISPAN*, pages 207–213, 1996.

[3] Sanjeev Baskiyar and Kiran Kumar Palli. Low power scheduling of dags to minimize finish times. In *13th International Conference on High Performance Computing*, pages 353–362, 2006.

[4] Christian Belady. The Green Grid Data center Efficiency Metrics: PUE and DCIE. Technical report, The Green Grid, Feb. 2007.

[5] Damien Borgetto, Michael Maurer, Georges Da-Costa, Jean-Marc Pierson, and Ivona Brandic. Energy-efficient and sla-aware management of iaas clouds. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 25:1–25:10, New York, NY, USA, 2012. ACM.

[6] Wu chun Feng, Avery Ching, and Chung-Hsing Hsu. Green supercomputing in a desktop box. In *Proceedings of the 21th International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–8, 2007.

[7] Wu chun Feng and Thomas Scogland. The green500 list: Year one. In *Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing*, pages 1–7, 2009.

[8] Georges Da Costa, Marcos Dias de Assunção, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and Amal Sayah. Multi-facet approach to reduce energy consumption in clouds and grids: the green-net framework. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 95–104, New York, NY, USA, 2010. ACM.

[9] John Zahorjan Derek L. Eager, Edward D. Lazowska. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Software Eng.*, 12(5):662–675, 1986.

[10] William Forrest. How to cut data centre carbon emissions? Website, December 2008.

[11] Vincent W. Freeh and David K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 164–173, New York, NY, USA, 2005. ACM.

[12] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *SC*, page 34, 2005.

[13] Rong Ge, Xizhou Feng, Wu chun Feng, and Kirk W. Cameron. Cpu miser: A performance-directed, run-time system for power-aware clusters. In *ICPP*, page 18, 2007.

[14] Ian Gorton, Paul Greenfield, Alexander S. Szalay, and Roy Williams. Data-intensive computing in the 21st century. *IEEE Computer*, 41(4):30–32, 2008.

[15] The Green Grid. A Framework for Data Center Energy Productivity. Technical report, Feb. 2008.

[16] Flavius Gruian and Krzysztof Kuchcinski. Lenes: task scheduling for low-energy systems using variable supply voltage processors. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 449–455, 2001.

[17] Chung-Hsing Hsu and Wu chun Feng. A feasibility analysis of power awareness in commodity-based high-performance clusters. In *CLUSTER*, pages 1–10, 2005.

[18] Chung-Hsing Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *SC*, page 1, 2005.

[19] Kyong Hoon Kim, Rajkumar Buyya, and Jong Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *CCGRID*, pages 541–548, 2007.

[20] Sung J Kim. A general approach to multiprocessor scheduling. Technical report, Austin, TX, USA, 1988.

[21] Hideaki Kimura, Mitsuhisa Sato, Yoshihiko Hotta, Taisuke Boku, and Daisuke Takahashi. Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster. *Cluster Computing, IEEE International Conference on*, 0:1–10, 2006.

[22] Young Choon Lee and Albert Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *CCGRID*, pages 92–99, 2009.

[23] Rongheng Li and Huei Chuen Huang. List scheduling for jobs with arbitrary release times and similar lengths. *J. Scheduling*, 10(6):365–373, 2007.

[24] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[25] Virginia Mary Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. Computers*, 37(11):1384–1397, 1988.

[26] Jiong Luo and Niraj K. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *VLSI Design*, pages 719–727, 2002.

[27] Jiong Luo and Niraj K. Jha. Power-efficient scheduling for heterogeneous distributed real-time embedded systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(6):1161–1170, 2007.

[28] Jiong Luo, Niraj K. Jha, and Li-Shiuan Peh. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. *IEEE Trans. VLSI Syst.*, 15(4):427–437, 2007.

[29] Ali Manzak and Chaitali Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy. In *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, pages 279–282, New York, NY, USA, 2001. ACM.

[30] Steven M. Martin, Krisztian Flautner, Trevor Mudge, and David Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, ICCAD '02, pages 721–725, New York, NY, USA, 2002. ACM.

[31] Abdellatif Mtibaa, Bouraoui Ouni, and Mohamed Abid. An efficient list scheduling algorithm for time placement problem. *Computers & Electrical Engineering*, 33(4):285–298, 2007.

[32] Marcus T. Schmitz and Bashir M. Al-Hashimi. Considering power variations of dvs processing elements for energy minimisation in distributed systems. In *ISSS*, pages 250–255, 2001.

[33] Behrooz A. Shirazi, Krishna M. Kavi, and Ali R. Hurson, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[34] Seung Woo Son, Konrad Malkowski, Guilin Chen, Mahmut T. Kandemir, and Padma Raghavan. Reducing energy consumption of parallel sparse matrix applications through integrated link/cpu voltage scaling. *The Journal of Supercomputing*, 41(3):179–213, 2007.

[35] Frédéric SUTER. Synthetic dag generation. Web Page.

[36] Gary Verdun. The Green Grid metrics: Data center infrastructure efficiency (DCiE) detailed anaysis. Technical report, The Green Grid, Feb. 2007.

[37] Gregor von Laszewski and Lizhe Wang. GreenIT Service Level Agreements. In *Service Level Agreements in Grids Workshop, colocated with IEEE/ACM Grid 2009 Conference*, Banff, Canada, Oct. 2009.

[38] Gregor von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *CLUSTER*, pages 1–10, 2009.

[39] Lizhe Wang, Gregor von Laszewski, Jai Dayal, and Thomas R. Furlani. Thermal aware workload scheduling with backfilling for green data centers. In *IPCCC*, pages 289–296, 2009.

[40] Qingzhou Wang and Kam Hoi Cheng. List scheduling of parallel tasks. *Inf. Process. Lett.*, 37:291–297, March 1991.

[41] Y. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Trans. Computers*, 34(3):204–217, 1985.

[42] Min-You Wu and Daniel Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Trans. Parallel Distrib. Syst.*, 1(3):330–343, 1990.

[43] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 374–, Washington, DC, USA, 1995. IEEE Computer Society.

[44] Gu yeon Wei, Jaeha Kim, Dean Liu, Stefanos Sidiropoulos, and Mark A. Horowitz. A variable-frequency parallel i/o interface with adaptive power-supply regulation. *IEEE J. Solid-State Circuits*, 35:1600–1610, 2000.

[45] Yumin Zhang, Xiaobo Sharon Hu, and Danny Z. Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of the 39th annual Design Automation Conference*, DAC '02, pages 183–188, New York, NY, USA, 2002. ACM.

[46] Ziliang Zong, Adam Manzanares, Xiaojun Ruan, and Xiao Qin. Ead and pebd: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. *IEEE Trans. Computers*, 60(3):360–374, 2011.

[47] Ziliang Zong, Adam Manzanares, Brian Stinar, and Xiao Qin. Energy-aware duplication strategies for scheduling precedence-constrained par-

allel tasks on clusters. In *Proceedings of the 2006 IEEE International Conference on Cluster Computing*, 2006.