Proc. 27th IEEE International Performance Computing and Communications Conference (IPCCC), Dec. 2008.

Distributed Energy-Efficient Scheduling for Data-Intensive Applications with Deadline Constraints on Data Grids*

Cong Liu[1], Xiao Qin[2], Santosh Kulkarni[2], Chengjun Wang[2], Shuang Li[2], Adam Manzanares[2], and Sanjeev Baskiyar[2]
University of North Carolina at Chapel Hill[1]
Auburn University[2]

Abstract

Although data duplications may be able to improve the performance of data-intensive applications on data grids, a large number of data replicas inevitably increase energy dissipation in storage resources on the data grids. In order to implement a data grid with high energy efficiency, we address in this study the issue of energy-efficient scheduling for data grids supporting real-time and data-intensive applications. Taking into account both data locations and application properties, we design a novel Distributed Energy-Efficient Scheduler (or DEES for short) that aims to seamlessly integrate the process of scheduling tasks with data placement strategies to provide energy savings. DEES is distributed in the essence - it can successfully schedule tasks and save energy without knowledge of a complete grid state. DEES encompasses three main components: energy-aware ranking, performance-aware scheduling, and energy-aware dispatching. By reducing the amount of data replications and task transfers, DEES effectively saves energy. Simulation results based on a real-world trace demonstrate that with respect to energy consumption, DEES conserves over 35% more energy than previous approaches without degrading the performance.

1. Introduction

Distributed scientific applications in many cases require access to massive data sets. In High Energy Physics (HEP) applications [8], for example, a handful of experiments have started producing petabytes of data per year for decades. Data grids [7] have served as a technology bridge between the need to access extremely large data sets and the goal of achieving high data transfer rates by providing geographically distributed computing resources and large-scale storage systems. When it comes to distributed systems such as data grids, it is the responsibility of schedulers to decide where to run applications (the terms application and task are used interchangeably throughput this paper) based on the applications' specific requirements as well as system workload conditions. Data resources are of paramount importance for many data-intensive applications - from long running simulations to remote sensing; from biological sequence analysis to video-on-demand systems [11]. A key factor in the process of scheduling data-intensive tasks is the location of input data required by the tasks. A straightforward strategy to enhance performance of data-intensive applications on data grids is to replicate popular data sets (i.e., frequently accessed data sets) to multiple resource sites, thereby offering higher data access speeds compared to maintaining the data sets in a single site. A wide range of data replication strategies, which are practical and effective, have been commonly applied in distributed data centers [15][12]. However, making too many replicas may ultimately lead to a number of drawbacks. First, it is challenging to maintain consistency among replicas in large scale distributed systems such as grids. Second, it is nontrivial to efficiently generate replicas of massive data sets on the fly in data grids. Last but not least, a large number of data replicas inevitably and dramatically increase the energy dissipation in storage resources, which in turn often leads to large electricity bills. Recent studies show that large-scale clusters may require 40TWh per year, costing over $4Billion per year at the price of $100 per MWh [6].

* The work reported in this paper was supported by the US National Science Foundation under Grants No. CCF-0742187, No. CNS-0757778, No. CNS-0831502, No. OCI-0753305, No. DUE-0621307, and No. DUE-0830831, and Auburn University under a startup grant.

Clearly, it is a non-trivial task to improve the performance of data-intensive applications through data replicas while reducing energy dissipation in storage systems in data grids. It is necessary to make better tradeoffs between energy efficiency and high-performance for data-intensive applications since they are two conflicting design goals. In this paper, we investigate an approach to seamlessly integrate data placement strategies with task scheduling, in which both energy efficiency and real-time requirements (e.g., tasks' deadlines) are fully addressed. In particular, we develop a novel Distributed Energy-Efficient Scheduler called *DEES* containing three key components: energy-aware ranking, performance-aware scheduling, and energy-aware dispatching. By leveraging an array of data placement strategies, *DEES* is able to maximize the number of tasks completed before their corresponding deadlines while replicating data in an energy-efficient way. To furnish *DEES* with an energy-efficient task dispatching mechanism that dispatches real-time tasks to peer computing sites, one has to simultaneously consider three factors: computational capacities of peer computing sites, energy consumption introduced by tasks, and data location. An interesting property of *DEES* is that the scheduling overhead of *DEES* does not necessarily increase when data grids scale up. This is quite different from most other grid scheduling techniques in which a centralized scheduler for a data grid inherently exhibits an undesirable performance bottleneck and single point failures may occur. Unlike most existing schedulers deployed in data grids, *DEES* does not require full knowledge of workload conditions of all the computing sites in a data grid. One must consider that obtaining full knowledge of the state of the grid is a difficult task.

The remainder of this paper is organized as follows. A review of recent related work is given in Section 2. Section 3 describes the system model. Section 4 presents the detailed design of *DEES*. Section 5 presents a comprehensive set of simulations that were used to evaluate the performance of *DEES*. Conclusions and future work appear in Section 6.

2. Related work

Unlike traditional parallel and distributed systems, in which required data usually resides in sites where tasks are allocated, data grids allow data required by applications to be distributed across multiple sites. The significance of data placement in task scheduling and dispatching in data grids has led to several innovative strategies having been proposed in recent years.

Data replications are used to reduce communication costs and avoid data access hotspots. Ranganathan and Foster [13] considered dynamic task scheduling along with data staging requirements. In contrast to this work, it is known that computation scheduling and data replication phases are partly independent of one another. Simulation results have shown that task scheduling and data placement can be optimized separately [13]. Kosar and Livny [9] proposed a scheduler named Stork for data placement in grids. Their motivational rationale is to efficiently complete computational cycles by placing data close to computational resources. Mohamed and Epema [10] developed an algorithm named Close-to-File (*CF*) that schedules tasks on sites with enough processing capacity that are close to the sites where the required data (input files) reside. *CF* uses an exhaustive algorithm to search across all combinations of computing and data sites to find a result with the minimum computation and transmission cost.

Chang [4] developed the Hierarchical Cluster Scheduling (HCS) algorithm and the Hierarchical Replication Strategy (HRS), which maximize the required data in a region in order to fetch replicas faster. HCS not only considers computational capacity and data location, but also takes cluster information as an input. The rationale behind HRS is that nearby data has a higher priority to access as compared to generating new replicas. Chakrabarti [3] proposed the Integrated Replication and Scheduling Strategy (IRS) that decouples task scheduling from data placement. It calculates the popularity of current required files and replicates the most popular data for the next set of tasks.

However, none of the aforementioned studies consider energy efficiency. QoS requirements, such as application deadlines, have also not been taken into account. Furthermore, most of the aforementioned algorithms require full knowledge of the state of the entire grid, which is difficult and/or expensive to obtain and maintain.

3. System model

3.1 Data grid model

As shown in Fig. 1, geographically distributed sites are interconnected through a WAN. We define a site as a location that contains computing resources and large-scale storage systems. Heterogeneity and dynamicity cause resources in grids to be distributed unevenly.

The internal structure of each site is shown in Fig. 2. Each site consists of storage resources, computing resources and a ticket server. Storage resources are used to store data while computing resources are devoted to computations. A ticket server is the server

used to run the scheduler, which is in charge of sending, receiving, and processing tickets. A ticket is a very small file that contains certain attributes of the task. Users submit tasks to the ticket server. The scheduler tries to schedule, as many tasks as possible, on the local site. Unscheduled tasks are sent to the most promising neighbors using tickets.

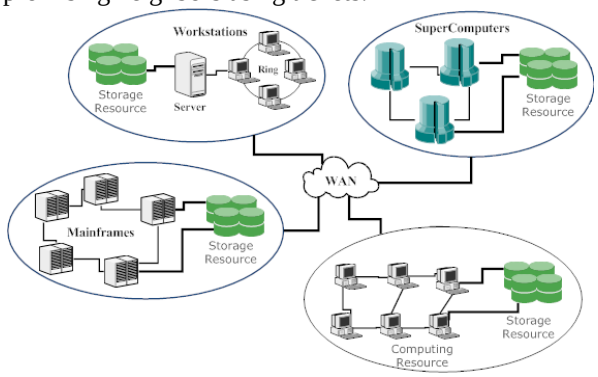


Fig. 1. Data grid model

We consider scheduling Bag-of-Tasks (BoT) applications, in which tasks are independent of each other. BoT applications are used in a variety of scenarios, including parameter sweeps [1] and computer imaging [14]. Furthermore, because of the independence of their tasks, BoT applications can be successfully executed over geographically distributed grids. This feature has been demonstrated by SETI@home [2]. It is arguably true that BoT applications are most suited for grids [5], where communication costs can easily become a bottleneck for highly-coupled parallel applications.

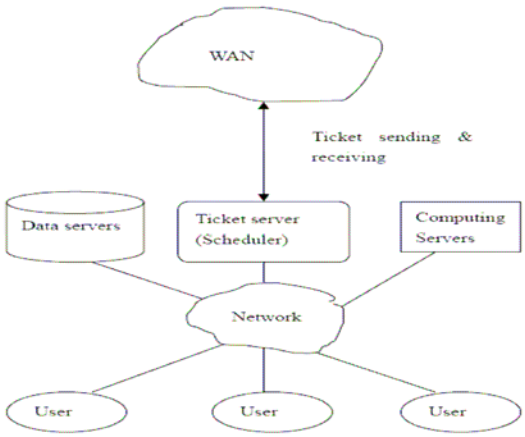


Fig. 2. The internal structure of a site

3.2 Energy consumption model

We model the energy consumption of task execution and data replica transfers. It is assumed that the storage center at each site has enough storage space to accommodate every data request. The parameters used in this section are shown in Table 1.

Table 1. Important notation

Notations	Description
N	total number of tasks
W	total number of machines within the grid
Q	total number of sites within the grid
s_v	site v
m_k	machine k
t_i	task i
d_i	deadline of task i
α	read energy consumption rate (*watt*)
β	write energy consumption rate (*watt*)
λ	network energy consumption rate (*watt*)
$\varepsilon_{k,v}$	computation power of m_k at s_v (*watt*)
$b_{u,v}$	bandwidth between s_u and s_v
$s(t_i)$	size of t_i's execution code
$s(d_i)$	size of data d_i
e_i	number of instructions of t_i, in terms of MI (Million Instruction)
$cc_{k,v}$	computing capacity of m_k at s_v, in terms of MIPS (Million Instruction Per Second)
$r_{i,u}$	time to read d_i from s_u
$w_{i,v}$	time to write d_i to s_v
$p_{k,v}$	idle time period of m_k at s_v
$\eta_{k,v}$	idle power of m_k at s_v (*watt*)

If the energy consumed by cooling systems is not considered, the total energy consumption of a data grid, E_{total}, can be expressed as:

$$E_{total} = E_{comp} + E_{comm} + E_{rep}, \quad (1)$$

where E_{comp} is the total energy consumption of computing resources, E_{comm} is the total energy consumption of communication, and E_{rep} is the total energy consumption of replicating data.

E_{comp} contains two parts: (*i*) E^c is the energy consumed to execute all tasks, and (*ii*) E^i is the energy consumed when machines are idle. E^c can be written as:

$$E^c = \sum_{i=1}^{N}\sum_{k=1}^{W}\sum_{v=1}^{Q} E^c_{i,k,v}, \quad (2)$$

where $E^c_{i,k,v}$ is the energy consumption for executing a task t_i on machine m_k at site s_v, N is the total number of tasks, W is the total number of machines within the grid, and Q is the total number of sites within the grid. $E^c_{i,k,v}$ can be expressed as:

$$E^c_{i,k,v} = \begin{cases} \varepsilon_{k,v} \times \dfrac{e_i}{cc_{k,v}} & \text{if } t_i \text{ is scheduled on } m_k \text{ at } s_v \\ 0 & \text{otherwise} \end{cases}, \quad (3)$$

where e_i is the number of instructions of t_i, $cc_{k,v}$ is the computing capacity of m_k at s_v, and $\varepsilon_{k,v}$ is the computation power of m_k at s_v.

The idle energy consumption, E^i, can be defined as:

$$E^i = \sum_{k=1}^{W} (p_{k,v} \times \eta_{k,v}), \quad (4)$$

where $p_{k,v}$ is the idle time period of m_k, and $\eta_{k,v}$ is the idle power of m_k.

Thus, according to Eq. (2), (3), and (4), the total computation energy consumption, E_{comp}, can be written as the aggregate of E^c and E^i for all tasks:

$$E_{comp} = \sum_{i=1}^{N} \sum_{k=1}^{W} \sum_{v=1}^{Q} E^c_{i,k,v} + \sum_{k=1}^{W} (p_{k,v} \times \eta_{k,v}). \quad (5)$$

The total communication energy consumption, E_{comm}, is composed of two parts: (*i*) $E^{td}_{i,j,o,v}$ is the energy consumed to transfer t_i's required dataset, the j^{th} data d_j, from s_o to s_v, where s_o is the site at which d_j is located, and (*ii*) $E^{tt}_{i,u,v}$ is the energy consumed to transfer the execution code of t_i from s_u to s_v, where s_u is t_i's local site. $E^{td}_{i,j,o,v}$ can be expressed as:

$$E^{td}_{i,j,o,v} = \begin{cases} 0 & if\ o = v \\ \lambda \times \dfrac{s(d_j)}{b_{o,v}} & if\ o \neq v \end{cases}, \quad (6)$$

where $s(d_j)$ is the size of d_j, $b_{o,v}$ is the bandwidth between s_o and s_v, and λ is the network energy consumption rate. If s_v already has the data (i.e. $o=v$), no data transfer cost will be incurred. $E^{tt}_{i,u,v}$ can be expressed as:

$$E^{tt}_{i,u,v} = \begin{cases} 0 & if\ u = v \\ \lambda \times \dfrac{s(t_i)}{b_{u,v}} & if\ u \neq v \end{cases}. \quad (7)$$

Note that if t_i is executed on the local site (i.e. $u=v$), no task transfer cost will be incurred.

Therefore, E_{comm}, which is composed of the aggregate of $E^{td}_{i,j,o,v}$ and $E^{tt}_{i,u,v}$ for all tasks, becomes:

$$E_{comm} = \sum_{i=1}^{N} \sum_{v=1}^{Q} \left(E^{td}_{i,j,o,v} + E^{tt}_{i,u,v}\right) \times x_{i,v}, \quad (8)$$

where $x_{i,v}$ can be defined as:

$$x_{i,v} = \begin{cases} 1 & if\ t_i\ is\ scheduled\ at\ s_v \\ 0 & otherwise \end{cases}. \quad (9)$$

$$subject\ to: \quad \forall i,\ \sum_{v=1}^{Q} x_{i,v} = 1$$

The third part of the total energy consumption is the data replication energy consumption. The energy consumed to replicate t_i's required dataset d_j from s_o to s_v, denoted as $E^{r,w}_{i,j,o,v}$, contains two components: (*i*) $E^r_{i,j,o}$ is the energy consumed to read d_j from s_o, and (*ii*) $E^w_{i,j,v}$ is the energy consumed to write d_j to s_v. $E^r_{i,j,o}$ can be expressed as:

$$E^r_{i,j,o} = \alpha \times r_{j,o}, \quad (10)$$

where $r_{j,o}$ is the time to read d_j from site s_o, and α is the read energy consumption rate. Similarly, $E^w_{i,j,v}$ can be expressed as:

$$E^w_{i,j,v} = \beta \times w_{j,v}, \quad (11)$$

where $w_{j,v}$ is the time to write d_j to site s_v, and β is the write energy consumption rate. So the total energy consumed to replicate d_j from s_o to s_v, $E^{r,w}_{i,j,o,v}$, is:

$$E^{r,w}_{i,j,o,v} = \begin{cases} 0 & if\ o = v \\ \alpha \times r_{j,o} + \beta \times w_{j,v} & if\ o \neq v \end{cases}. \quad (12)$$

Therefore, E_{rep}, which is composed of the aggregate of $E^{r,w}_{i,j,o,v}$ for all tasks, can be written as:

$$E_{rep} = \sum_{i=1}^{N} \sum_{v=1}^{Q} (E^{r,w}_{i,j,o,v} \times x_{i,v}). \quad (13)$$

By substituting Eq. (5),(8), and (13) into Equation (1), the total energy consumption of a data grid, E_{total}, can be derived as:

$$E_{total} = \sum_{i=1}^{N} \sum_{k=1}^{W} \sum_{v=1}^{Q} E^c_{i,k,v} + \sum_{k=1}^{W} (p_{k,v} \times \eta_{k,v}) + \sum_{i=1}^{N} \sum_{v=1}^{Q} \left(E^{td}_{i,j,o,v} + E^{tt}_{i,u,v} + E^{r,w}_{i,j,o,v}\right) \times x_{i,v}. \quad (14)$$

3.3 Energy consumption analysis

We describe four scenarios that may occur when a task request is scheduled. Let us analyze the energy consumption in each scenario.

- **S1. Local execution and local data:** The task execution is performed at its local site s_u where the required input data is located.
- **S2. Local execution and remote data:** The task execution is performed at its local site s_u and the input data is replicated from a remote site s_o.
- **S3. Remote execution and same remote data:** The task, whose local site is s_u, is executed at a remote site s_v and the input data is already located at the same remote site
- **S4. Remote execution and different remote data:** The task, whose local site is s_u, is executed at a remote site s_v and the input data is replicated from another remote site s_o.

Hence, $E_{i,k,v}$, which is the energy consumed to schedule t_i on m_k at s_v, can be expressed as:

$$E_{i,k,v} = E^{c}_{i,k,v} + E^{td}_{i,j,o,v} + E^{tt}_{i,u,v} + E^{r,w}_{i,j,o,v}$$
$$= \begin{cases} E^{c}_{i,k,v} + 0 + 0 + 0 & if\ S1 \\ E^{c}_{i,k,v} + E^{td}_{i,j,o,v} + 0 + E^{r,w}_{i,j,o,v} & if\ S2 \\ E^{c}_{i,k,v} + 0 + E^{tt}_{i,u,v} + 0 & if\ S3 \\ E^{c}_{i,k,v} + E^{td}_{i,j,o,v} + E^{tt}_{i,u,v} + E^{r,w}_{i,j,o,v} & if\ S4 \end{cases} . \quad (15)$$

From Eq. (15), it is observed that executing a task at a site where its data is located is the most energy efficient, because no data transfer and replication cost is incurred. Compared to the *local execution and remote data* scenario, executing the task at a remote site where data is located is still more energy efficient because of the fact that the size of a task's input data is usually much larger than the size of its execution code.

3.4 Motivational example

We observe that reducing the amount of data replication and task transfers can effectively save energy. This is in contrast to making as many data replications as possible, which improves performance at the cost of energy. We give a simple case study that demonstrates our ideas. An environment with different types of processors is simulated as follows:

- P1: AMD Athlon 64 X2 6400+ with 85W TDP
  Energy Consumption Rate: Busy: 104w idle:15w
- P2: AMD Athlon 64 X2 3800+ with 35W TDP
  Energy Consumption Rate: Busy: 47w idle: 11w
- A network with energy consumption rate of *(60w)*

As shown in Fig. 3, three sites A, B, and C are connected by two links that have the same bandwidth. For simplicity, all the attributes are assigned as standard unit values. The performance-driven scheduling algorithm tends to assign a task to a machine that provides the minimum completion time. Using this algorithm, t_1 will be scheduled on B, t_2 scheduled on C, t_3 scheduled on B, and t_4 scheduled on C. This is the combination that will produce minimum completion times, so that the total units of data transferred, denoted as DT_1, become:

$$DT_1 = s(t_1) + s(d_1) + s(t_2) + s(d_1) + s(t_3) + s(d_2) + s(t_4) + s(d_2) = 16$$

According to our energy consumption model, the total energy consumption (considering the energy consumption when machines are idle and busy) is:

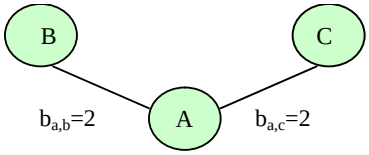
$$E^{1}_{total} = 954.8\ joules$$

If the scheduling algorithm considers energy efficiency, tasks will be scheduled on resources that not only consume the least amount of energy, but also are capable of meeting the task deadlines. Using this energy aware scheduler, t_1 and t_2 will be scheduled on A, and t_3 and t_4 scheduled on B. In this case, the total units of data transferred, denoted as DT_2, become:

$$DT_2 = s(t_3) + s(d_2) + s(t_4) = 6$$

According to our energy consumption model, the total energy consumption is:

$$E^{2}_{total} = 596\ joules$$



Site	Data Property		Machine Property	
	ID	Size	ID	Computing Capacity
A	d_1	4	P2	1
	d_2	4		
B	-	-	P1	5
C	-	-	P1	5

Task	Local site	e	s(e)	Deadline	Data
t_1	A	3	1	6	d_1
t_2	A	3	1	6	d_1
t_3	A	5	1	4	d_2
t_4	A	5	1	4	d_2

Fig. 3. A motivation example

Compared with the performance-driven scheduling algorithm, the energy-efficient scheduling algorithm schedules the same number of tasks that meet their deadlines. However, the energy-efficient algorithm saves more energy by executing tasks with local data on local resources, which reduces task transfers and data replications. Moreover, when making a data replication to a remote site, maximizing the utilization of this replica also saves energy. The energy saving is achieved by assigning as many tasks as possible to the remote site.

4. Scheduling Algorithm

The design goal of *DESS* is three fold: (*i*) Maximize the number of tasks meeting their deadlines, (*ii*) Minimize energy consumption, and (*iii*) Provide scalability.

DEES consists of three phases: Ranking, Scheduling, and Dispatching, as shown in Fig. 4. First incoming tasks at each site are ranked in a task queue. Tasks are grouped together according to the data location. Second, the scheduler at each site assigns each task to a specific local resource. Finally unscheduled tasks are dispatched to remote sites, where the same

algorithm is used to make scheduling decisions. The pseudo code of *DEES* is shown in Fig. 5.

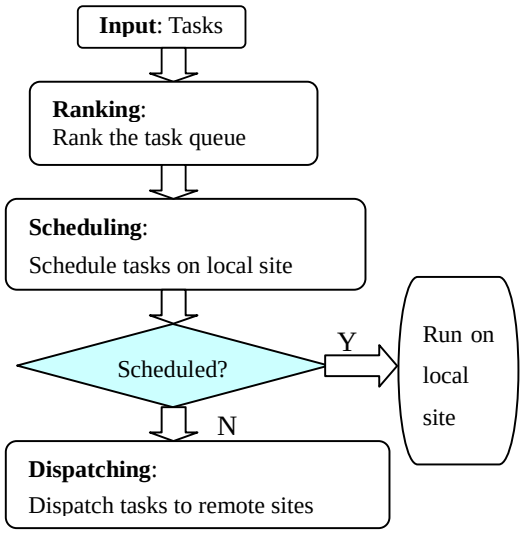


Fig. 4. Diagram for the Scheduler

DEES

1. Group tasks according to the data location
2. Rank the local task group first, and then sort other task groups by number of tasks in descending order
3. Within each group, rank tasks by decreasing deadline
4. for each task group g_i in order
5. Schedule every unscheduled $t \in g_i$ on the machine that gives the minimum completion time while meeting the deadline
6. if unscheduled tasks left
7. *Dispatch*
end *DEES*

Fig. 5. DEES

4.1 Ranking

Tasks are sorted into a task queue. Tasks requiring the same data are grouped together. The task group whose data resides in the local site, called *local task group*, is ranked first. Then other task groups are ranked in descending order, according to the number of tasks in the task group. Within each group, tasks are ordered by increasing deadline. Thus, tasks with shorter deadlines are scheduled sooner.

4.2 Scheduling

The task grouping policy always schedules the local task group onto local resources first. According to the energy consumption analysis, it minimizes data replication so that energy can be saved. For tasks whose required data is located at remote sites, by grouping them according to the data location, *DEES* is able to further reduce data replication and data transfer in the *Dispatching* phase, which will be described in detail in Section 4.3.

DEES schedules tasks on a group basis. A local task group is scheduled first. In order to schedule task t_i on site s_u, *DEES* selects machine m_k at s_u that can complete t_i within its deadline and provide the minimum completion time. After processing all tasks, remaining unscheduled tasks will be dispatched to remote sites.

4.3 Dispatching

Our dispatching strategy delivers tasks within each task group to the data site. For task group g_j whose data site is s_o, scheduling decisions are made by s_o's scheduler based on its local resource status and task information of g_j. If s_o cannot schedule all tasks in g_j, then unscheduled tasks are dispatched to s_o's immediate neighbors using tickets in a breadth-first manner. A ticket has several attributes: task ID, deadline, number of instructions, size of execution code, location of the required input data, schedulable flag, and route information.

In order to make tradeoffs between energy efficiency and real-time performance, we propose a ranking system, called Highest Rank First (HRF), to rank s_o's neighbors. The rank of s_o's neighbor s_v is defined as:

$$rank(g_i, s_v, s_o) = \varepsilon \times n + \mu \times \frac{1}{E_{rep}^{i,o,v} + (E_{comm}^{i,o,v} + E_{comp}^{i,n,v})/n}, \quad (16)$$

where n is the number of tasks in g_j that can be scheduled on s_v, ε is a coefficient concerning the task deadline, μ is a coefficient concerning energy saving, $E_{rep}^{i,o,v}$ is the energy consumed to replicate g_i's data from s_o to s_v, $E_{comm}^{i,o,v}$ is the energy consumed to transfer g_i's data and n unscheduled tasks from s_o to s_v, and $E_{comp}^{i,n,v}$ is the energy consumed to execute these n tasks at s_v.

The neighboring site with the top ranking will be considered first. Neighbors are checked using a breadth-first search. If there are tasks unscheduled in g_i after visiting all neighbors of s_o, the nearest neighbor (which has the fastest link bandwidth) will search its neighbors using the same algorithm. This process continues until suitable remote resources have been found, or all sites have been visited.

Introducing ε and μ, we enable *DEES* to manage the two conflicting goals of saving energy and meeting deadlines. If the incoming tasks are mission-critical, ε is set to 1 and μ is set to 0, which means the neighbor that can schedule more tasks is given preference. When energy consumption becomes more important, ε is set

to 0 and μ is set to 1. Thus, the neighbor that consumes the least amount of energy will be considered first. In addition, we studied the impact of different ε and μ values in the simulation. Our goal is to find a balanced pair of these values that enables *DEES* to save energy while giving the best performance. The pseudo code of the dispatching phase is shown in Fig. 6.

Dispatch
1. **while** unscheduled task group g_i at site s_u, requiring data d_j located at s_o
2. Send tasks within g_i to s_o
3. Sort s_o's neighbors by *HRF*
4. **for** each neighbor that has not been visited by g_i
5. Replicate d_j from s_o to s_v which is ranked first
6. Send schedulable portion of g_i from s_o to s_v
7. Update the task queue in s_o
8. Mark s_v as visited by g_i

end *Dispatch*

Fig. 6. Dispatch

4.4 Complexity

Let n be the number of incoming tasks at each site, m the number of machines within each site, and s the number of sites. Then, the complexity of the ranking phase is *O(nlogn)*, of *Schedule* is *O(nm)* and of *Dispatch* is *O(ns)*. Therefore, the complexity of *GDS* is *O(nlogn)*, assuming $s < logn$ and $m < logn$. We note that the complexity of *Close-to-Files is O(Nms)*, where N is the total number of incoming tasks.

Table 2. Characteristics of system parameters

Parameter	Value(fixed)-(varied)
Number of jobs	(9600)-(1600,3200,6400,9600 12800,16000,19200, 22400)
Number of sites	(32)-
Site processing speed	8*8 nodes
Number of datasets	(200)-(100,200,400)
Task execution time range (Uniform distribution)	(1,500) second
Size of datasets	(500-800MB short jobs, 800MB-1GB medium jobs, 1-2GB long jobs)-(500MB-2GB)
Dataset popularity distribution	(Uniform)-(Uniform, Normal, Geometric)
Dataset popularity threshold	(2)-(2,4,6,8,10)

5. Simulation

We conducted extensive simulations based on the San Diego Supercomputer Center (SDSC) SP2 log to evaluate *DEES*. The real trace was sampled on a 128-node IBM SP2 (67,665 jobs from April 1998 to April 2000) [16]. The system parameters in a simulated grid system are chosen to resemble real-world workstations such as IBM SP2 nodes. **Error! Reference source not found.** summarizes the key parameters of the simulated grid system used in our experiments. Each data point is an average of 30 runs.

To reveal the strengths of DEES, we compared it with an effective scheduling algorithm, namely, *Close-to-Files* [10]. The *Close-to-Files* algorithm gives good performance since it takes data locality into account. It always schedules a task to its data site. Doing so helps decrease the amount of data transfer. Moreover, it is an exhaustive algorithm that searches across all combinations of computing and data sites to find a result with the minimum computation and transmission cost, which gives good performance but incurs considerable amount of overhead. The CF algorithm is summarized as follows:

1. First tasks are ranked by decreasing size.
2. For each task, if there exist data sites that can successfully execute the task; schedule it on a data site which is the first according to the alphabetical order of names.
3. If there is no data site that can schedule the task, first find all pairs of execution sites with sufficient idle processors and data sites of the task; then try to find the pair with the minimal data transfer time.
4. If no site can be found to execute this task, it is marked as un-schedulable.
5. Repeat steps 2-4 till all tasks have been processed.

Moreover, in order to demonstrate that DEES is able to reduce energy consumption without sacrificing the system performance, we proposed the *Performance-driven* scheduling algorithm and compare it with DEES. The *Performance-driven* scheduling algorithm can be summarized as follows:

1. First tasks are ranked by increasing deadline.
2. Then each task is scheduled onto the resource that gives the minimum completion time, regardless of the data locality.
3. If no resource can be found to execute a task, this task is marked as un-schedulable.
4. Repeat steps 2-3 till all tasks have been processed.

Note that both *Close-to-Files* and *Performance-driven* algorithms are centralized algorithms that need the knowledge of a complete state of the grid.

The *Guarantee Ratio*, *Normalized Average Energy Consumption* and *Total Energy Consumption* are used as the performance metrics in the evaluation. The algorithm that produces the highest *Guarantee Ratio* with the lowest *Normalized Average Energy*

Consumption and the lowest *Total Energy Consumption* is considered the best algorithm. The *Guarantee Ratio* and *Normalized Average Energy Consumption* are defined as:

$$Guarantee\,Ratio = \frac{N_s}{N_{total}}, \qquad (17)$$

where N_s is the number of scheduled tasks meeting deadlines, and N_{total} is the total number of tasks.

$$Normalized\,Average\,Energy\,Consumption = \frac{E_{total}}{N_s}, \qquad (18)$$

where E_{total} is the total energy consumption.

5.1 Impact of ranking coefficients

The first experiment set was to investigate the impact of using different (ε, μ) combinations. In dispatching, neighboring sites are ranked by Eq. (18). We introduce ε and μ, which are two coefficients related to meeting task deadlines and saving energy, respectively. Varying ε and μ, *DEES* is able to switch between the two conflicting goals of saving energy and meeting deadlines.

Fig. 7 shows the performance of *DEES* using different (ε, μ) value pairs with respect to *Guarantee Ratio*. It is observed that *DEES (2, 1)* gives the best performance. This is because *DEES (2, 1)* takes both goals of meeting deadline and saving energy into account, and put more weight onto the deadline meeting part. Neighbors that can schedule more tasks are given preference. Another observation is that *DEES (0, 1)* gives the worst performance, since it only considers energy consumption. Our conclusion is that giving preference to neighbors that can schedule more tasks while consuming satisfactory amount of energy yields higher *Guarantee Ratio*.

With respect to *Normalized Average Energy Consumption*, as shown in Fig. 8, we observe that *DEES (2, 1)* consumes the least amount of energy while *DEES (0, 1)* consumes the most. *DEES (2, 1)* considers both energy consumption and deadline constraints when dispatching tasks to neighbors. Doing so can reduce the energy cost per task. There are two reasons why *DEES (0, 1)* performs the worst. First, fewer tasks can be scheduled since it only cares about the energy consumption when dispatching tasks. This in turn leads to an increase in the energy consumption per task. Second, given that more tasks miss their deadlines at each site, additional data replications may be needed. This is done in order to find remote sites, which may be more than 1 hop away, to meet tasks' deadlines. Therefore it relatively consumes more energy to replicate data and transfer the tasks.

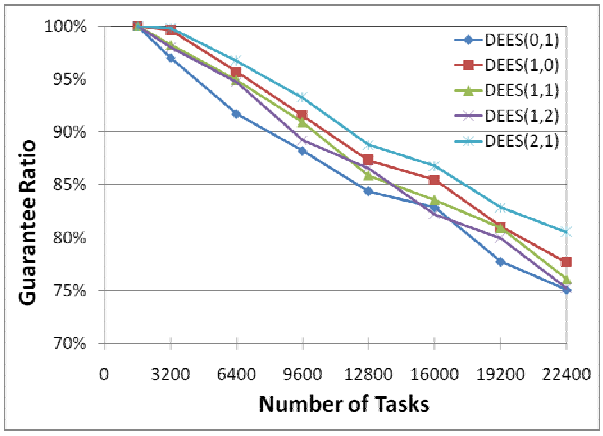


Fig. 7. *Guarantee Ratio* by ranking coefficients

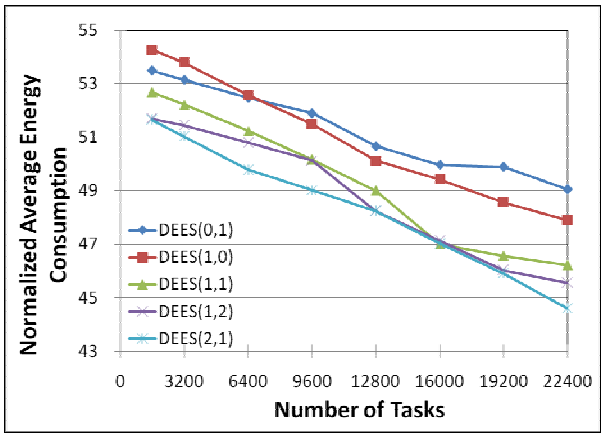


Fig. 8. *Normalized Average Energy Consumption* by ranking coefficients

5.2 Performance

In this experiment set, we compared the performance of *DEES* with *Close-to-Files* and *Performance-driven* algorithms under different task loads. From Fig. 9, we observe that *DEES* yields better performance than *Close-to-Files* and achieves the same performance level as the *Performance-driven* algorithm does. The *Performance-driven* algorithm always schedules a task to a globally best resource that gives the best performance. Since it only focuses on performance but not other factors such as data locality, it yields very good performance with respect to *Guarantee Ratio*. But the fact that DEES gives similar performance as the *Performance-driven* algorithm is importance. Thus, *DEES* not only reduces energy consumption, but it does so without degrading the *Guarantee Ratio*. One reason is because *DEES* always schedules tasks with shorter deadlines first. The final criteria for judging whether a task can be scheduled are the task deadlines. Scheduling those tasks with shorter

deadlines first makes more tasks schedulable. Moreover, *DEES* is fully distributed, which is expected to improve the performance when compared to a centralized algorithm, such as the Performance-driven algorithm, especially when the task load is heavy. Given that *DESS* is fully distributed, while *Close-to-Files* and *Performance-driven* algorithms are centralized algorithms that need knowledge of a complete state of the grid, the results make *DEES* more favorable.

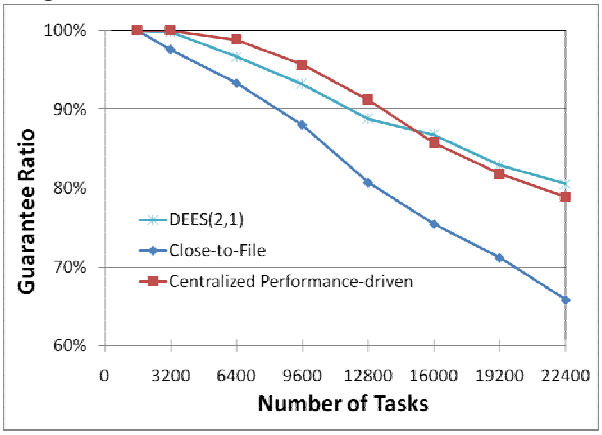


Fig. 9. *Guarantee Ratio* by task loads

With respect to *Normalized Average Energy Consumption*, as shown in Fig. 10, we see that *DEES* consumes much less energy per task than *Close-to-Files* does. On average *DEES* saves over 35% of energy consumed when compared to the other algorithms. This is because *DEES* considers the energy consumed to transfer both tasks and data during dispatching. Moreover, *DEES* groups tasks according to their data accesses and processes tasks on a group basis. Doing so limits the number of data replicas. This is because whenever data is replicated to a remote site, *DEES* always maximizes utilization of the data replicated by scheduling as many tasks as possible to that remote site. By doing so the energy cost of execution per task is reduced. On the other hand, *Close-to-Files* makes dispatching decisions on a single task basis, which may result in unnecessary data replications. Furthermore, since *DEES* schedules more tasks than *Close-to-Files* does, the energy cost per task is expected to be less. The *Performance-driven* algorithm consumes the most amount of energy due to the fact that it is a greedy algorithm that always schedules a task to a resource giving the best performance, regardless of how much data are needed to be replicated and transferred.

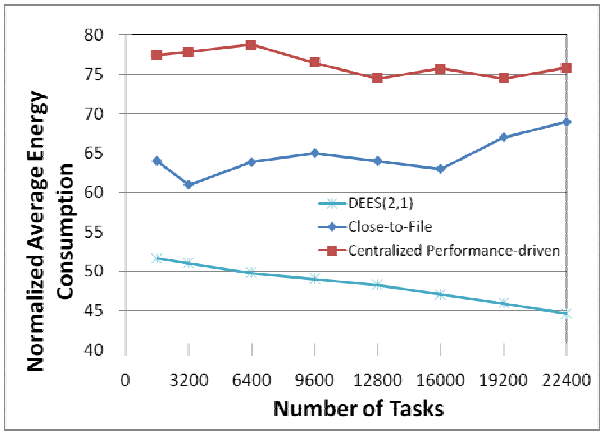


Fig. 10. *Normalized Average Energy Consumption* by task loads

5.3 Total Energy Consumption under Light Workload

When the incoming workload is light (i.e. below 2000 tasks), all the algorithms are able to give a 100% guarantee ratio, as shown in Table 3. This is because the grid has sufficient computing capacity to handle all incoming tasks.

Table 3. Guarantee Ratio under light workload

Algorithm\Task No.	400	800	1200	1600
DEES	100%	100%	100%	100%
Close-to-Files	100%	100%	100%	100%
Performance-driven	100%	100%	100%	100%

Since the three algorithms can successfully schedule all incoming tasks, we are able to conduct a fair comparison regarding the total amount of energy consumption. As shown in Fig. 11, DEES consumes 25% less energy on average than the other algorithms. An interesting observation is that DEES has a lower increasing rate on energy consumption along with the increasing number of tasks. This is again because *DEES* maximizes the utilization of the replicated data by scheduling as many tasks as possible to the site where the data is replicated. Doing so reduces the amount of data replication and transfer. On the other hand, *Close-to-Files* and *Performance-driven* algorithms make scheduling and dispatching decisions on a single task basis, which may result in unnecessary data replications.

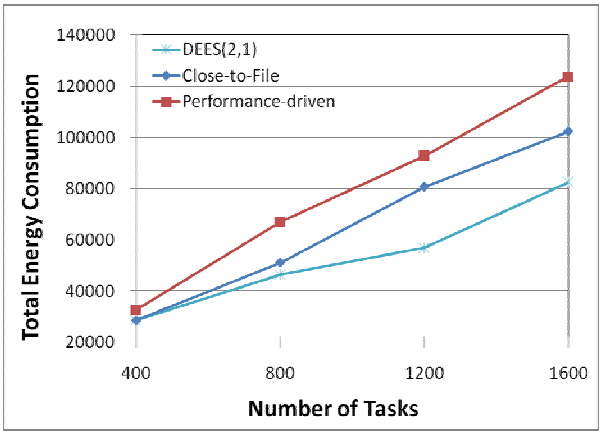


Fig. 11. Total Energy Consumption by task loads

6. Conclusion

In this paper, we proposed a novel energy efficient algorithm to schedule real-time tasks with data access requirements on data grids. By reducing the amount of data replication and task transfers, the proposed algorithm effectively saves energy. Our algorithm is distributed since it does not need knowledge of the complete state of the grid. Detailed simulations demonstrate that *DEES* significantly reduces the energy consumption while increasing the *Guarantee Ratio*. In the future, we will investigate the reliability of *DEES* as well as address temporal fault tolerance.

7. References

[1] D. Abramson, J. Giddy and L. Kotler, "High performance parametric modeling with Nimrod/G: Killer Application for the global grid", *Proceedings of the 14th IPDPS*, 2000, pp. 520-528.

[2] D. Anderson, J. Cobb, and E. Korpela, "SETI@home: An experiment in Public-Resource Computing", *Communication of the ACM*, 2002, vol. 45, pp 56-61.

[3] A. Chakrabarti, R.A. Dheepak, and S. Sengupta, "Integration of scheduling and replication in data grids", *LNCS*, 2004, vol. 3296, pp. 375-385.

[4] R. Chang, J. Chang, and S. Lin, "Job scheduling and data replication on data grids", *Future Generation Computer Systems*, 2007, pp. 846-860.

[5] W. Cirne, et al., "Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach", *Proceedings of the 2003 ICPP*, 2003, pp. 407-416.

[6] R. Doyle, "Energy Management for Server Clusters", *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, 2001, pp. 165.

[7] I. Foster and C. Kesselman, *The Grid2*, Morgan Kauffmann Publishers, 2003.

[8] High Energy Physics experiments [Online] http://www.cithep.caltech.edu [Accessed Sept. 8, 2007]

[9] T. Kosar and M. Livny, "Stork: Making data placement a first class citizen in the grid", *Proceedings of the 24th ICDCS*, 2004, pp. 342–349.

[10] H.H. Mohamed and D.H.J. Epema, "An evaluation of the close-to-files processor and data co-allocation policy in multi-clusters", *Proceedings of the 2004 Cluster*, 2004, pp. 287–298.

[11] X. Qin, "Design and Analysis of a Load Balancing Strategy in Data Grids," *Future Generation Computer Systems*, 2007, vol. 23, no. 1, pp. 132-137.

[12] M. Rabinovich, I. Rabinovich, and R. Rajaraman. "Dynamic Replication on the Internet", *Technical Report HA6177000-980305-01-TM*, AT&T Labs, 1998.

[13] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications", *Proceedings of the 11th HPDC*, 2002, pp. 352-258.

[14] S. Smallen, H. Casanova, and F. Berman, "Applying scheduling and tuning to On-line parallel tomography", *Proceedings of Supercomputing*, 2001, pp. 46.

[15] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm", *ACM Transactions on Database System*, 1997, pp. 255–314.

[16] T. Xie and X. Qin, "A Security-Oriented Task Scheduler for Heterogeneous Distributed Systems", *Proceedings of the 13th HiPC*, 2006, pp. 35-46.