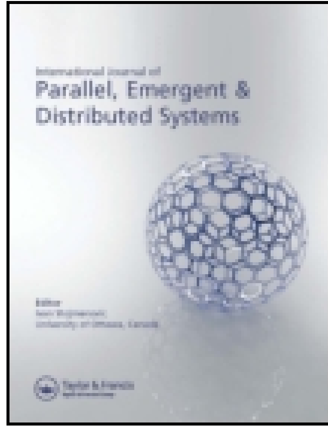
This article was downloaded by: ["Queen's University Libraries, Kingston"]
On: 07 July 2015, At: 11:54
Publisher: Taylor & Francis
Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: 5 Howick Place, London, SW1P 1WG



International Journal of Parallel, Emergent and Distributed Systems
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/gpaa20

Cooperative ad hoc computing: towards enabling cooperative processing in wireless environments

Waleed Alsalih [a] , Selim Akl [a] & Hossam Hassanein [a]
[a] School of Computing at Queen's University , Kingston, Ont., Canada , K7L 3N6
Published online: 16 Jan 2008.

To cite this article: Waleed Alsalih , Selim Akl & Hossam Hassanein (2008) Cooperative ad hoc computing: towards enabling cooperative processing in wireless environments, International Journal of Parallel, Emergent and Distributed Systems, 23:1, 59-79, DOI: 10.1080/17445760701445013

To link to this article: http://dx.doi.org/10.1080/17445760701445013

PLEASE SCROLL DOWN FOR ARTICLE

Taylor & Francis makes every effort to ensure the accuracy of all the information (the "Content") contained in the publications on our platform. However, Taylor & Francis, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the Content. Any opinions and views expressed in this publication are the opinions and views of the authors, and are not the views of or endorsed by Taylor & Francis. The accuracy of the Content should not be relied upon and should be independently verified with primary sources of information. Taylor and Francis shall not be liable for any losses, actions, claims, proceedings, demands, costs, expenses, damages, and other liabilities whatsoever or howsoever caused arising directly or indirectly in connection with, in relation to or arising out of the use of the Content.

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden. Terms & Conditions of access and use can be found at http://www.tandfonline.com/page/terms-and-conditions

The International Journal of Parallel, Emergent and Distributed Systems,
Vol. 23, No. 1, February 2008, 59–79

Taylor & Francis
Taylor & Francis Group

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

Cooperative *ad hoc* computing: towards enabling cooperative processing in wireless environments

WALEED ALSALIH*, SELIM AKL† and HOSSAM HASSANEIN‡

School of Computing at Queen's University, Kingston, Ont., Canada K7L 3N6

(Received 12 October 2006; revised 15 January 2007; in final form 22 February 2007)

Mobile applications are becoming more popular as they provide the convenience of accessing services and information anywhere and at anytime. However, due to size and weight restrictions, mobile computing devices are limited in terms of battery energy and processing power. Running complex applications on resource-limited mobile computing devices is a real challenge. This paper proposes a cooperative paradigm for *ad hoc* computing in which a set of heterogeneous computing devices form a cooperative system on the fly, and whenever a resource-limited computing device in such a system has a resource-consuming application to be run, it uses resources of other devices to surmount the problem of resource limitation. To study the potential capability of this paradigm, we have designed energy-aware allocation and scheduling algorithms to make the most of all available resources in such a cooperative environment. These algorithms are static in the sense that they are based on discrete snapshots of the system topology. Although we consider these algorithms as preliminary work towards our objective of enabling cooperative computing, they have shown exciting results that encourage us to pursue our study. The main contributions of this paper are the novel allocation and scheduling algorithms which form a pioneering work towards enabling energy-aware cooperative processing in mobile computing environments.

Keywords: *Ad hoc* computing; Task scheduling; Task allocation; Wireless networks; Energy management

1. Introduction

The evolution of distributed systems in the 1970s has allowed several personal computers to work together in one system and has brought about a great improvement over the independent personal computers in terms of processing power, access to remote services and information, fault tolerance, resource sharing, resource availability and so on [1]. On the other hand, mobile computing has recently gained popularity, and the need for mobile applications in different fields has become significant. But unfortunately, rigorous resource

*Corresponding author. Tel.: + 1-613-533-6000. Ext. 78232. Fax: + 1-613-533-6513. Email: waleed@cs.queensu.ca
†Email: akl@cs.queensu.ca
‡Email: hossam@cs.queensu.ca

The International Journal of Parallel, Emergent and Distributed Systems
ISSN 1744-5760 print/ISSN 1744-5779 online © 2008 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/17445760701445013

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

limitations in mobile computing devices preclude the full utilization of mobile applications in real life scenarios. However, in an analogy to how distributed systems have improved the computing value of personal computers, cooperative *ad hoc* computing has the potential to leverage the durability and processing power of pervasive and mobile computing devices.

We envision the *ad hoc* computing environment to be composed of a set of heterogeneous computing devices, of different characteristics and capabilities, which form a wireless network on the fly and start working in such a cooperative behavior that reduces the effect of resource limitation in mobile computing devices. To argue the need for such a cooperative paradigm, let us consider the following scenarios.

1.1 Mobile computing for emergencies

The triage is a process carried in hospital emergency rooms to sort injured people into groups based on their needs to immediate treatment. The same process is actually needed in disaster areas with a large number of casualties. Identifying the most severe injuries so quickly has proven to be very effective in saving lives and controlling severe injuries. But unfortunately, both technical and human resources available in the emergency room are not available in such a disaster field. Mobile computing has been proposed to automate and expedite the triage process in disaster fields. First, low-power vital sign sensors are attached to each patient in the field. These sensors send medical data about the patient to nearby first responders who are provided with mobile computing devices and medical applications that are able to process and analyze the received data in order to make a decision on who needs the most immediate treatment [2]. Running such an application on a resource-limited mobile computing device is challenging; such a device may not have enough energy to run the medical application and/or may not have the processing power needed to make a decision early enough. Alleviating the effect of these limitations, which is the main objective of cooperative computing, would save more lives in the future.

1.2 Computing-saturated spaces

Computation offloading [3–5] was introduced as a technique by which a resource-limited computing device can defer the execution of a computational task to a nearby stationary server, which has virtually unlimited energy and has more capabilities in terms of processing power. It is possible, through a suitable computation offloading platform, to have a building, a lab or a meeting room equipped with a sufficient number of stationary servers that can be accessed via a wireless medium, and provide mobile devices with processing power and energy by taking over the execution of computational tasks initiated on roaming mobile computing devices. It is obvious that what a mobile computing device can perform in such a space is much more than what it can perform independently. Servers have been used in distributed environments to provide different kinds of services and information; there is nothing to prevent them from providing processing power and energy.

However, as is the case with all profound technologies, *ad hoc* computing poses several challenges, and it is the job of researchers and engineers to overcome these challenges to make the technology "weave itself into the fabric of everyday life until it is indistinguishable from it" [6]. The following are some of the challenges posed by the *ad hoc* computing paradigm:

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

(i) Since almost all mobile computing devices are battery-operated, energy has become one the most critical resources. Classical protocols and algorithms, which assume wall-powered computers, are not suitable any more for *ad hoc* computing.
(ii) The processing power of a mobile computing device is much lower than that of a stationary one. This actually emerges the need for cooperative processing and dispatching computational tasks to other devices.
(iii) Mobile environments lack the preinstalled infrastructure.
(iv) Mobility of computing devices imposes the challenge of dynamic topology.
(v) High degree of heterogeneity is another challenge in mobile computing environments.

This paper makes a significant contribution towards surmounting some of the above challenges as follows.

- It formulates novel energy-aware allocation and scheduling problems that are suitable for recent and future *ad hoc* computing environments in which energy is one of the most valuable resources.
- It proposes a heuristic-based allocation algorithm that shows empirically the ability to generate task assignments which are near-optimal in terms of minimizing the total consumed energy.
- It proposes a heuristic-based scheduler that reconciles energy saving with performance improvement, and provides the system with the ability to set different importance weights for each of them.

In the proposed scheduling schemes, a set of tasks are scheduled and assigned to different devices based on the most recent snapshot of the network topology. The network topology is assumed to stay unchanged until the execution of all tasks. Several methods have been proposed to maintain these snapshots for wireless networks with moderate mobility rates [7]. However, while using discrete snapshots is a valid assumption in some wireless environments such as the computing-saturated spaces discussed above and the smart environments [8], some other environments possess a high degree of mobility and require dynamic, realtime scheduling, which is the topic of a current investigation. We believe that this work originates new research directions in the field of energy-aware task scheduling (ETS) towards energy conservative, high performance cooperative systems.

The remainder of this paper is organized as follows. Section 2 describes the general structure of a cooperative *ad hoc* computing system. Section 3 formulates the allocation problem of finding a proper assignment of tasks to heterogeneous devices, which minimizes energy consumption, and presents a heuristic-based greedy algorithm for obtaining a near-optimal solution to this problem. Section 4 states the more general problem of finding a schedule that minimizes a cost functionof energy consumption and makespan (i.e. the time required to have all tasks completed), and presents a heuristic-based algorithm to solve it approximately. Finally, Section 5 concludes this paper by summarizing the main contribution of this work, discussing the importance and viability of our approach, and suggesting some future research directions.

2. Cooperative *ad hoc* computing architecture

Besides the fact that most mobile computing devices are equipped with wireless connection facilities, several computing platforms have been designed specially for mobile computing

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

devices to support process migration and remote execution [3–5]. This makes it feasible for heterogeneous mobile and stationary computing devices to work in a cooperative manner. Furthermore, recent advances in Mobile *Ad hoc* wireless NETworks (MANETs) facilitate constructing such a system on the fly [9,10].

An *ad hoc* computing environment consists of a set of heterogeneous computing devices that may range from small, resource-limited, mobile devices to stationary, wall-powered, powerful servers. Whenever a resource-limited computing device in such an environment has a set of tasks (or subtasks¶) to be executed (which may have dependencies and communication requirements among themselves), it uses all available resources in nearby computing devices. A mobile computing device may use resources of one or more nearby stationary servers and/or even the resources of one or more other nearby mobile devices. A personal digital assistant (PDA), for example, may use the resources of a nearby laptop. Figure 1 shows the general architecture of the proposed cooperative *ad hoc* computing.

The wireless network allows different devices to discover and communicate with each other. The remote execution platform facilitates the process of migrating a task to the remote server and getting the results back to the client. However, neither the wireless network nor the remote execution platform is the focus of this paper. An *ad hoc* computing environment involves a set of consumers, i.e. mobile applications (tasks), and a set of resources, i.e. energy and processing power. Therefore, finding a proper *schedule* plays the main role for energy saving and performance improvement. A schedule includes a task assignment that shows for each task which device is to execute it and an execution order that shows the starting time of each task. Finding an efficient schedule in this environment is the focus of the next two sections.

3. Energy-aware task allocation

The main deterrent from enabling ubiquitous mobile computing is the battery energy limitation. Cooperative computing reduces the effect of such a limitation by distributing the computational tasks of a mobile computing device amongst all available, nearby devices in an energy-efficient way. This section looks into the problem of finding the proper task allocation that minimizes the overall consumed energy. Energy is consumed by the execution of a task and by the interaction between two tasks assigned to different computing devices (processors).

The classical task allocation problem is the problem of finding a proper assignment of tasks to processors that minimizes the sum of the total execution time and the total communication time. This problem has been proven to be NP-Complete in its general form [11], i.e. no polynomial algorithm has been found to solve the problem, and it is believed by most of the researchers in the field to be extremely hard (or even impossible) to find one. However, in this energy-aware allocation problem, the objective is to minimize the sum of the total execution energy and the total communication energy. Since both time and energy can be seen as a cost metric to be minimized, the two problems are equivalent, i.e. a time-based classical allocation algorithm can be used as an energy-aware allocation algorithm.

¶A task may be partitioned to a set of subtasks. However, the task partitioning problem is out of the scope of this work. Rather, it is assumed that the task is already efficiently partitioned.

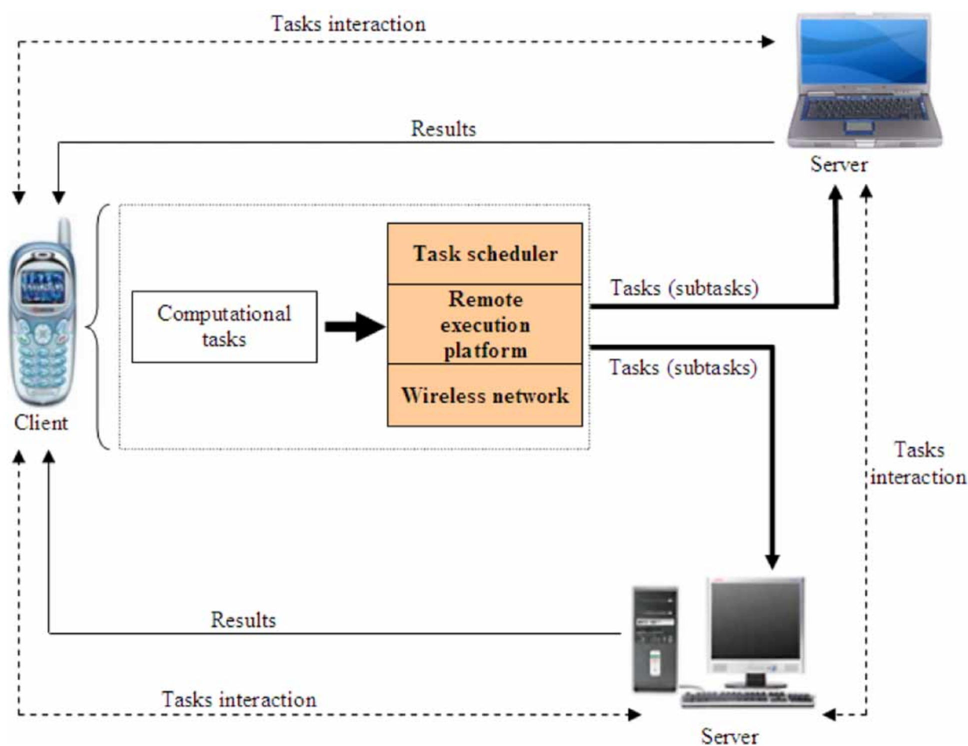


Figure 1. Cooperative *ad hoc* computing.

However, classical allocation algorithms assume that the communication links are identical, i.e. the communication cost between any two processors is the same. This is not a valid assumption in a wireless mobile computing environment, which imposes a high degree of heterogeneity among different communication links. In a wireless communication environment, the communication links have different costs and may be asymmetric, i.e. the cost of a link from a processor p_i to another processor p_j may be different from that of the link from p_j to p_i.

Stone's allocation algorithm [12] was proven to be optimal in terms of minimizing the sum of the total execution time and the total communication time when there are exactly two processors and the communication links in both directions are identical (i.e. they have the same transmission rate). Therefore, if there are only two processors and the communication links between them are identical in terms of energy consumption, Stone's algorithm can be used to find the minimum-energy task assignment.

In a more general case with an arbitrary number of processors and identical communication links, the algorithm of Abraham and Davidson [11] was proven to give a solution no worse than twice the optimal. Therefore, when this assumption applies in terms of energy cost, the same algorithm can be used as an energy-aware approximation algorithm.

3.1 Problem formulation

Classical allocation models, which have been proposed for distributed and parallel systems, are not suitable for mobile computing environments where energy is probably the most

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

critical resource. We define new task model, processor model and cost function to formulate this energy-aware allocation problem abstractly.

- Task model $\mathcal{T}$
  - (i) Tasks: $T = (t_1, t_2, \ldots, t_n)$ is a set of tasks to be executed, where n is the number of tasks.
  - (ii) Interaction among tasks: Data is an $n \times n$ matrix, where Data(i, j) is the amount of data units (e.g. bytes) to be sent from t_i to t_j.
  - (iii) Execution energy consumption of each task on each processor: ExecE is an $n \times m$ matrix, where m is the number of processors and ExecE(i, j) is the amount of energy consumed by t_i when executed on p_j.

    A *Directed Acyclic Graph* (DAG) is usually used to represent the task model. It is a directed graph in which there is a node for every task and a directed edge for every interaction relation. The weight of an edge from t_i to t_j is the amount of data sent from t_i to t_j, i.e. Data(i, j).
- Processor model $\mathcal{P}$
  - (i) Processors: $P = (p_1, p_2, \ldots, p_m)$ is a set of processors available in the system.
  - (ii) Estimated communication energy consumption among processors: CommE is an $m \times m$ matrix, where CommE(i, j) is the amount of energy consumed by one unit of data to travel from p_i to p_j.This includes the energy consumed by the source device, the destination device, and any other device involved along the path between source and destination.It is assumed that

$$\text{CommE}(i, i) = 0, 1 \leq i \leq m$$

    i.e. the cost of communication from a processor to itself is zero. It is also assumed that the task model and the processor model stay unchanged until all tasks complete execution (this is known as static allocation).
- Cost function $\mathcal{C}$

$$C = \text{communication_energy} + \text{execution_energy},$$

  where

- communication_energy is the total amount of energy consumed by interaction among tasks assigned to different processors.
- execution_energy is the total amount of energy consumed by the execution of all tasks.The problem can be formulated now as follows: Given a task model $\mathcal{T}$ and a processor model $\mathcal{P}$, find a task assignment $\mathcal{S}$ that maps each task to a processor in such a way that minimizes the following cost function:

$$C = \text{communication_energy} + \text{execution_energy}$$

  Having heterogeneous communication costs among different processors makes this problem distinct from classical allocation problems.

3.2 Task allocation algorithm

Since the allocation problem is NP-Complete [11], we introduce a heuristic-based algorithm that has been proven empirically to give near-optimal results. The algorithm we propose for

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

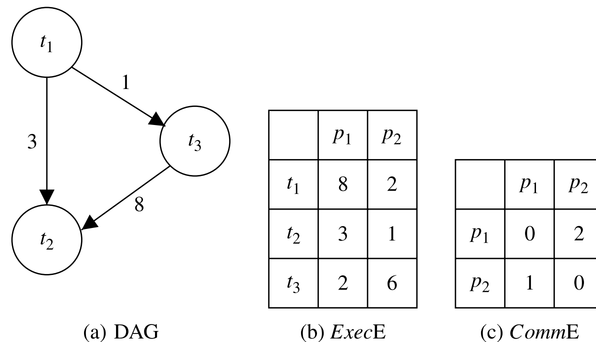
this problem tries to minimize the total consumed energy by assigning each task to the processor that executes it with the minimal energy. When an assignment decision is to be made for a task t, this algorithm considers the energy consumption of t's execution on different processors and the energy consumption of t's interaction with tasks that have been already assigned. However, t's interaction with tasks that have not yet been assigned is not considered because the cost of such interaction depends heavily on the assignment of those tasks and is impossible to predict. The algorithm follows.

(1) While there are still unallocated tasks
  (1.1) Pick any unallocated task t_i.
  (1.2) Assign t_i to a processor p_j, such that ExceE(i, j) $\leq$ ExceE(i, k), $1 \leq k \leq m$.
  (1.3) For any unallocated task t_c that receives data from t_i
    (1.3.1) For any processor p_k
      (1.3.1.1) $\mathrm{ExecE}(c, k) = \mathrm{ExecE}(c, k) + \mathrm{Data}(i, c) \times \mathrm{CommE}(j, k)$.
  (1.4) For any unallocated task t_c that sends data to t_i
    (1.4.1) For any processor p_k
      (1.4.1.1) $\mathrm{ExecE}(c, k) = \mathrm{ExecE}(c, k) + \mathrm{Data}(c, i) \times \mathrm{CommE}(k, j)$.

This algorithm runs in time $O(e + n)$ where e is the number of communication links among different tasks and n is the number of tasks [13]. The idea of adding the communication cost to the execution cost was used first by Lo [14] in her min-cut-based allocation algorithm. However, her algorithm assumes identical communication costs between different processors and minimizes the sum of the total execution time and the total communication time. Figure 2 shows an example of simple task and processor models consisting of two processors and three tasks. Figure 3 shows the task allocation steps carried by this allocation algorithm if tasks are assigned in the following order: t_3, t_1, and then t_2.

3.3 Experimental results

We conducted comprehensive experiments to show the quality of our allocation algorithm results. However, there is no classical task allocation algorithm that considers an arbitrary number of processors and arbitrary communication costs, which is the normal situation in a



	p_1	p_2
t_1	8	2
t_2	3	1
t_3	2	6

(b) *Exec*E

	p_1	p_2
p_1	0	2
p_2	1	0

(c) *Comm*E

Figure 2. An example of processor and task models.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

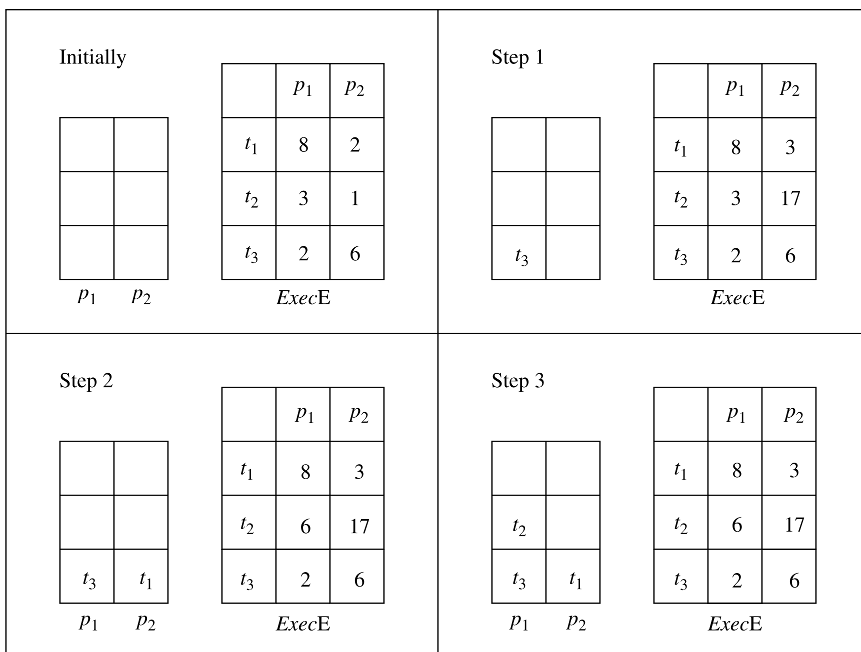


Figure 3. The task allocation steps of the example in figure 2.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

mobile wireless communication environment, and this led us to generate different benchmark categories, each of which has some restrictions so that we can compare our energy-aware task allocation algorithm (ETA) with another allocation algorithm that works only under these restrictions. In addition to the restricted categories, we generated a general category of unrestricted benchmarks, which have an arbitrary number of processors and heterogeneous communication links, and compared the results of ETA against the optimal allocation using a non-polynomial algorithm. However, benchmarks of this category have such a small number of tasks and processors that it is possible to get the optimal solution in a reasonable time using a non-polynomial algorithm.

The comparison was based on the overall consumed energy. We used task graphs for free (TGFF) [15] to generate random task and processor models (Benchmarks) of three different categories.

Category I has 200 benchmarks, each of which has about 150 tasks and about 300 inter-task communication links. The average execution energy consumption is 1000 nJ. The average number of data units on each communication link is 100 bytes. We also assumed a two-processor system with identical communication links (CommE(1, 2) $=$ CommE(2, 1) $=$ 1 nJ/byte). Since there are only two processors and the communication links between them are identical, Stone's allocation algorithm can be used to generate the optimal allocation in polynomial time, and we can compare the allocations of ETA against the optimal ones. Even though it seems curious to compare an optimal algorithm with a non-optimal one, we believe that the performance of ETA in this restricted case can be considered as an indication of its performance in the general case where the optimal algorithm is not applicable. It turns out that ETA allocations were within 2.46% of the optimal allocation in all generated benchmarks. ETA allocations were also optimal in 4% of the generated

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

benchmarks. Figure 4 shows the results of both algorithms in 20 randomly selected benchmarks.

Category II has 200 benchmarks, each of which has about 150 tasks and about 300 inter-task communication links. The average execution energy consumption is 1000 nJ. The average number of data units on each communication link is 100 bytes. We also assumed a ten-processor system with identical communication links (CommE(i, j) $= 1$ nJ/byte, $i \neq j$). Since the communication links are identical, the algorithm of Abraham and Davidson can be applied to generate allocations, which are no worse thantwice the optimal (i.e. 2-Approximation), and we can compare the allocations generated by ETA against those of the approximation algorithm. It turns out that the allocations of ETA consume, on average, 12.75% less energy compared to those generated by the approximation algorithm. ETA allocations are worse than those of the approximation algorithm in less than 6% of the tested benchmarks, and the difference in those benchmarks was never more than 5.9% of the approximation algorithm allocations. In about 25% of the tested benchmarks, ETA allocations are better than those of the approximation algorithm by less than 10%. In about 69% of the benchmarks, ETA allocations are better than those of the approximation algorithm by morethan 10% but less than 31%. Figure 5 shows the results of both algorithms in 20 randomly selected benchmarks.

Category III has 200 benchmarks, each of which has a maximum of 15 tasks and about 30 inter-task communication links. The average execution energy consumption is 1000 nJ. The average number of data units on each communication link is 100 bytes. With each benchmark, we generated a random processor model of four processors with heterogeneous communication links. The average communication energy consumption between any two processors is 5 nJ/byte. We used a non-polynomial algorithm to find the optimal allocation and compare it with the one generated by ETA algorithm. It turns out that ETA allocations consume, on average, 5.21% more energy compared to the optimal one. In 0.5% of the generated benchmarks, ETA allocations are worse than twice the optimal (in only one of the tested benchmarks, the ETA allocation consumes 106% more energy compared to the optimal one). In 3.5% of the tested benchmarks, ETA allocations consume more than 25% (but less than 100%) more energy compared to the optimal one. In 26.5% of the tested benchmarks, ETA allocations consume more than 5% (but less than 25%) more energy compared to the optimal one. And in 69.5% of the tested benchmarks, ETA allocations are within 5% of the optimal one. ETA allocations are optimal in 52.5% of the tested

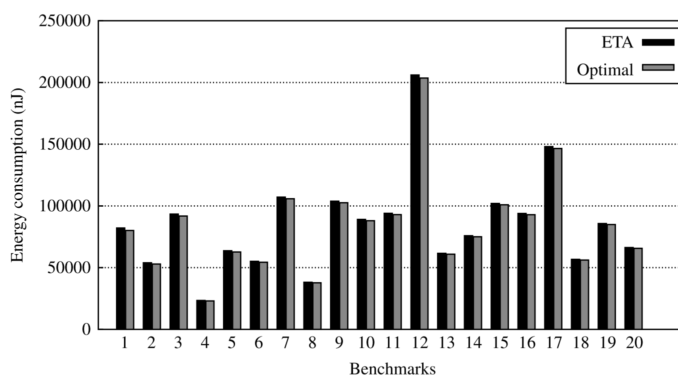


Figure 4. Comparison between ETA results and the optimal ones generated using Stone's algorithm in category I.

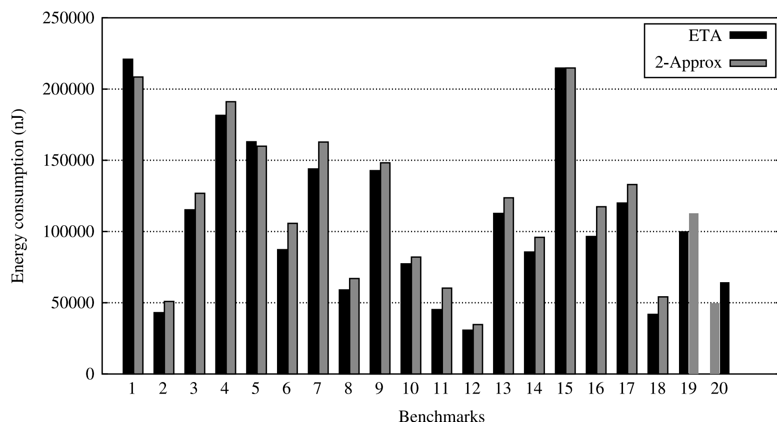


Figure 5. Comparison between ETA results and those of Abraham and Davidson approximation algorithm (2-approx.) in category II.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

benchmarks. Figure 6 shows a comparison between ETA results and the optimal ones in 20 randomly selected benchmarks.

4. Energy-aware task scheduling

The algorithm described in Section 3 finds a task allocation such that the total consumed energy is minimized regardless of the time needed to have all tasks completed. Therefore, it may lead to an energy-efficient, yet delay tolerant system. Inan *ad hoc* computing environment, both energy saving and performance may be of significant importance. However, their relative importance is not static and depends on some factors, such as current energy level, type of tasks being executed, and so on.

This section formulates a more general scheduling problem that aims at both conserving energy and improving performance. The cost function to be minimized here consists of an energy metric, a performance metric, and dynamic system-defined importance weights for

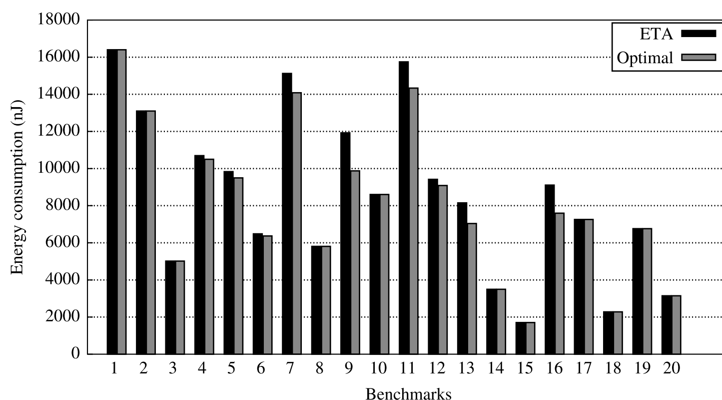


Figure 6. Comparison between ETA results and the optimal ones in category III.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

each metric. The allocation problem defined in Section 3 can be seen as a special case of this scheduling problem.

The novelty of this scheduling problem stems from having two different cost metrics to be minimized, namely, delay (makespan) and energy. These cost metrics are of different natures; while parallelism helps in minimizing the makespan, it does not have a clear effect on energy consumption and while the execution order affects the makespan, it too has no effect on energy consumption. A preliminary version of this work, which focuses on the scheduling algorithm, has been introduced in Ref. [16].

4.1 Problem formulation

In such a multi-objective scheduling problem, there is a need for task and processor models that consider both delay and energy. Such models have never been addressed before. We propose the following task model, processor model and cost function to formulate this scheduling problem abstractly.

- Task model $\mathcal{T}$
  - (i) Tasks: $T = (t_1, t_2, \ldots, t_n)$ is a set of tasks to be executed, where n is the number of tasks.
  - (ii) Interaction among tasks: Data is an $n \times n$ matrix, where Data(i, j) is the amount of data units (e.g. bytes) to be sent from t_i to t_j.
  - (iii) Precedence relation among tasks: $<$ is a partial order defined on T, where $t_i < t_j$ means that t_j cannot start execution before t_i has completed its execution and Data(i, j) data units have been received by the processor on which t_i is to be executed.
  - (iv) Execution time of each task on each processor: ExecT is an $n \times m$ matrix, where m is the number of processors and ExecT(i, j) is the execution time of t_i when executed on p_j.
  - (v) Execution energy consumption of each task on each processor: ExecE is an $n \times m$ matrix, where ExecE(i, j) is the amount of energy consumed by t_i when executed on p_j.
- Processor model $\mathcal{P}$
  - (i) Processors: $P = (p_1, p_2, \ldots, p_m)$ is a set of processors available in the system.
  - (ii) Estimated communication delay among processors: CommT is an $m \times m$ matrix, where CommT(i, j) is the amount of time required for one unit of data to travel from p_i to p_j.
  - (iii) Estimated communication energy consumption among processors: CommE is an $m \times m$ matrix, where CommE(i, j) is the amount of energy consumed by one unit of data to travel from p_i to p_j. This includes the energy consumed by the source device, the destination device, and any other device involved along the path between source and destination.

    We assume that CommT(i, i) $=$ CommE(i, i) $= 0$, $1 \le i \le m$, i.e. the cost of communication from a processor to itself is zero.We also assume that the task model and processor model stay unchanged until all tasks complete execution.
- Cost function $\mathcal{C}$

$$C = \alpha \text{makespan} + \beta \text{total_consumed_energy}$$

where

- *makespan* is the time by which all tasks are completed.
- total_consumed_energy is the total amount of energy consumed by task execution plus the total amount of energy consumed by task interaction among tasks.
- α and β are system attributes representing relative importance weights between processing performance (time) and energy saving.

This cost function is composed of two cost metrics of different units. While *makespan* is a time measured in microseconds, milliseconds, seconds, and so on, total_consumed_energy is energy measured in joules. Adding two cost metrics of different units is confusing, if not meaningless. However, it is possible to use a new cost unit (cost *tokens* for example) and to map both time units and energy units to cost *tokens* by using suitable, meaningful values for α and a. Thus, α(token/time unit) represents how many tokens are equivalent to one time unit and β(token/energy unit) represents how many *tokens* are equivalent to one energy unit. However, other methods may be used to unify the unitsof both cost metrics and that has no effect on the proposed algorithm.

The problem can be formulated now as follows: Given a task model $\mathcal{T}$ and a processor model $\mathcal{P}$, find a schedule $\mathcal{S}$ that maps each task to a processor and determines the starting time of each task in such a way that minimizes the following cost function:
$C = \alpha$ makespan $+$ β total_consumed_energy, such that:

- There is no execution overlapping among tasks that are assigned to the same processor.
- All precedence relations are satisfied.
- The schedule is non-preemptive, i.e. each task is assigned to only one processor and once a task starts execution, it can not be interrupted until it finishes.

4.2 List scheduling

List scheduling (LS) is a well known heuristic in scheduling algorithms for homogeneous systems, i.e. systems consisting of identical processors, and was introduced first by Hu in Ref. [17]. LS is a priority based heuristic in which each task is assigned a priority and whenever more than one task contend for execution, the one with higher priority is selected. The general LS algorithm can be expressed as follows.

(1) Each task is assigned a priority.
(2) A ready queue is initialized to have those tasks with no predecessor (the ready queue is a priority queue).
(3) While the ready queue is not empty
  (3.1) Get a task t from the front of the queue (the one with the highest priority).
  (3.2) Select a processor p to run t.
  (3.3) Find the starting time of t on p.
  (3.4) Insert in the ready queue all tasks of which all predecessors have been already executed.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

Two important decisions to be made here are:

(i) In Step 1, what criteria to be used for priority?
(ii) In Step 3.2, how to select a processor to run a task?

Both decisions should be made carefully in such a way that "logically" and "intuitively" will improve the quality of the generated schedule. Before discussing the optimality of LS, we define the following terms:

- A terminal task is a task with no successor. Task 5 in figure 7 is a terminal task.
- The length of a path in a DAG is the sum of the execution times of all tasks along this path. Note that the communication is not included in the length of a path.
- The level of a task in a DAG is the length of the longest path from the corresponding node to any exit node. Level of task 4 in figure 7 is 2 (if all tasks have execution time of one unit).
- A ready task is a task with no predecessor or a task of which all predecessors have completed execution.

In the case of identical processors, using the level of a task as its priority is presumed in the literature to be an effective technique to minimize the makespan, especially when communication is neglected. It is also obvious that a processor that becomes available first should be selected to run a particular ready task.

In Ref. [17], Hu presents a level-based LS algorithm that generates optimal schedules when the task graph is a tree, processors are identical, and all tasks have the same execution time. Coffman and Graham in Ref. [18] show that LS is optimal for scheduling tasks on an arbitrary task graph with identical execution times on only two processors. With an arbitrary number of processors, an arbitrary task graph, and arbitrary task execution times, Graham in Ref. [19] derives a precise bound on the makespan of level-based LS. He proves that the makespan of a level-based schedule would never be more than $(2 - (1/m))$ times the makespan of the optimal schedule where m is the number of processors. In the same paper, Graham also shows that this bound can be improved to $((4/3) - (1/3m))$ if there is no precedence relation. However, when the communication delay is significant, the quality of pure level-based LS is lowered because the communication delay is excluded while computing the level of each task. It is even quite hard to include the communication delay in the level computation because that depends heavily on the task assignment. This problem is called the level number problem for LS [11]. Nevertheless, some approaches have been proposed to adapt level-based

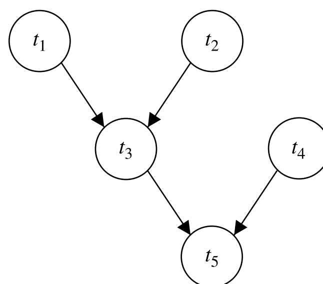


Figure 7. An example of a DAG of five tasks.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

LS so that it takes into consideration the communication delay. Task duplication [11], for example, duplicates the execution of the same task on different processors to eliminate the delay caused by communication originated from that task. It is, however, not clear yet how to find a low complexity algorithm with duplication [20]. Another approach is to divide the scheduling process into two stages: task assignment and execution ordering. In the first stage, where communication is excluded, a pure level-based LS is used to assign tasks to different processors. Once the task assignment is decided, the communication delay becomes explicit and can be included in the level computation. The next stage uses a more accurate level computation (that includes communication delays) to reorder the task execution locally on each processor. Even though no quality bounds have been derived for the general problem, extensive experimental studies and quality analyses have shown that two-stage LS is an effective heuristic when communication delay is significant [20].

4.3 Scheduling algorithm

Even if we ignore the communication cost and energy (i.e. minimizing the makespan only), this scheduling problem is NP-Complete [11]. Therefore, we developed a heuristic-based algorithm to solve the problem approximately. Our scheduling algorithm is divided into two stages: task assignment and then execution ordering. In the first stage, each task is assigned to one processor to execute it. In the second stage, the execution order in each processor is decided.

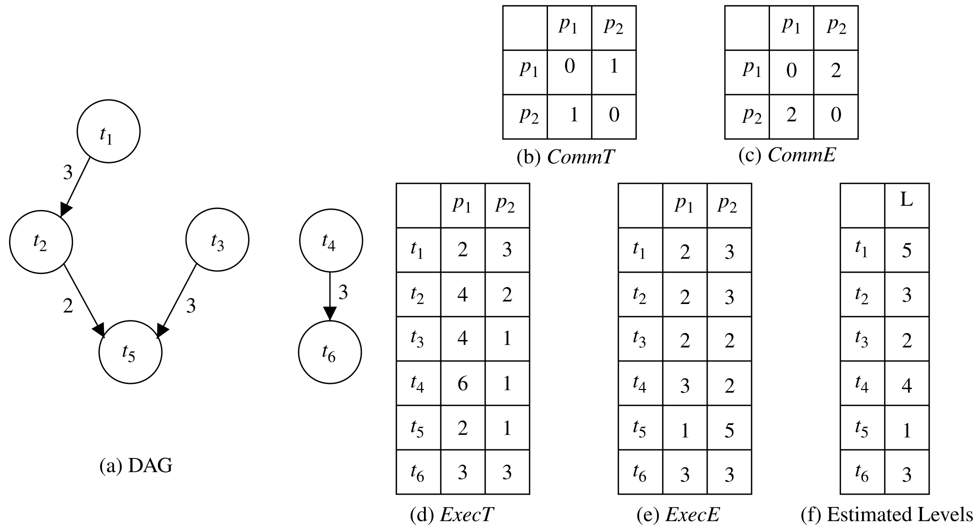
4.3.1 Task assignment. The task assignment algorithm uses a modified version of the level-based LS heuristic that is adapted to consider heterogeneous processors and energy consumption. In this scheduling problem, the execution time of a task varies from one processor to another and, therefore, it is hard to calculate the level of a task because that depends heavily on the task assignment. Nevertheless, it is possible to use the minimum execution time of each task as its predicted execution time and then to use this predicted execution time in computing the level of this task, i.e. it is in fact an estimated level rather than an actual one. In this algorithm, the priority of a task is its estimated level. Figure 8(a)–(e) shows an example of task and processor models, and figure 8(f) shows the corresponding tasks estimated levels which are computed based on the minimum execution time of each task.

On the other hand, this algorithm uses a greedy processor selection policy based on the finishing time and energy consumption on each processor. The cost of assigning task t to processor p is:

$$\text{AssignmentCost}(t, p) = \alpha FT + \beta \,\text{consumed_energy}$$

where:

- AssignmentCost(t, p) is the cost of assigning task t to processor p.
- FT is the finishing time of t on p which is affected by the communication delay, execution delay, and current load on p.
- consumed_energey: is the total energy consumed by executing t on p which includes the energy consumed by execution and energy consumed by communication.
- α and β are the same system attributes of the cost function defined in Section 4.1.What is meant by communication here is the communication from t's predecessors to p.



	p_1	p_2
p_1	0	1
p_2	1	0

(b) *CommT*

	p_1	p_2
p_1	0	2
p_2	2	0

(c) *CommE*

(a) DAG

	p_1	p_2
t_1	2	3
t_2	4	2
t_3	4	1
t_4	6	1
t_5	2	1
t_6	3	3

(d) *ExecT*

	p_1	p_2
t_1	2	3
t_2	2	3
t_3	2	2
t_4	3	2
t_5	1	5
t_6	3	3

(e) *ExecE*

	L
t_1	5
t_2	3
t_3	2
t_4	4
t_5	1
t_6	3

(f) Estimated Levels

Figure 8. An example of a processor model, a task model, and the corresponding estimated levels.

The algorithm simply selects the processor that can run t with minimal cost. The task assignment algorithm can be described as follows.

- First, assign each task a priority equal to its estimated level which includes the minimum execution time of each task along the longest path.
- Maintain a ready queue which includes those tasks of which all predecessors have been scheduled. This ready queue is initialized to have those tasks which have no predecessors, and whenever a task is scheduled, the ready queue is updated if some tasks become ready.
- As long as the ready queue is not empty, the ready task with the highest priority is removed from the front of the ready queue and assigned to the processor that can execute it with the minimum cost, i.e. using the *AssignmentCost* function. It is also scheduled to start as early as possible on the selected processor.Figure 9 shows the task assignment process of the example in figure 8 when $\alpha = 1$ token/μs and $\beta = 1$ token/nJ.

4.3.2 Execution ordering. There are two factors that may affect the execution ordering quality of the task assignment algorithm described in the previous section:

(i) The communication delay was excluded in computing the estimated level.
(ii) The minimum execution time was used as an estimate of the execution time of each task.

This may affect the makespan quality of the same assignment. However, even though finding the optimal order for a given assignment is NP-hard [20], several heuristics have been proven by experiment to be effective in minimizing the makespan [20]. One of the heuristics proposed in Ref. [20] is used for our execution ordering algorithm and it can be described as follows. Once the task assignment is decided, the exact communication delay and the exact execution time of each task can be determined and, therefore, a more efficient execution order can be made based on the exact level. The execution order is based on computing the exact level of each task which includes the exact execution times and the exactcommunication

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

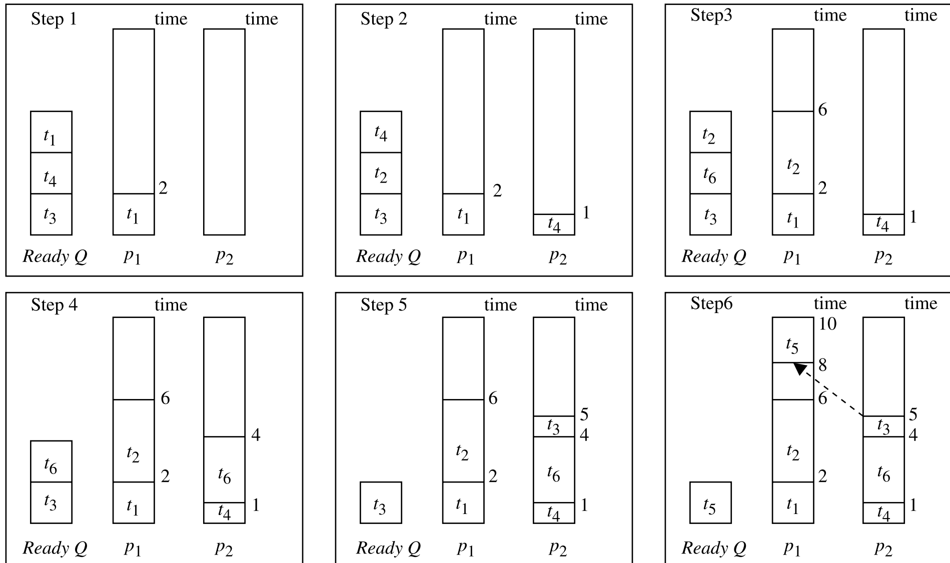


Figure 9. Illustration example for the task assignment process of figure 8 example with $\alpha = 1$ token/μs and $\beta = 1$ token/nJ.

delays, and then ordering the task execution on each processor based on the exact levels of those tasks assigned to it. It is obvious that ordering the execution of tasks on the same processor has no effect on energy consumption. Figure 10 shows the exact levels of the tasks shown in figure 8 after being assigned as shown in figure 9.

Another refinement that can be achieved is to consider a task ready only when all of its immediate predecessors have completed their execution and all of the data it should receive from them have arrived to the processor to which it was assigned. This can help avoid unnecessary idle gaps resulting from scheduling tasks prematurely. The execution ordering algorithm is based on the ready task critical path (RCP) heuristic proposed in Ref. [20] that can be described as follows.

- First, assign each task a priority equal to its exact level which includes the actual execution times and the actual communication delays along the longest path.

	L
t_1	8
t_2	6
t_3	6
t_4	4
t_5	2
t_6	3

Figure 10. Exact levels.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

- At each processor p_i, maintain a local ready queue RQ_i which includes those tasks assigned to p_i of which all immediate predecessors have been executed and all of the data they are supposed to receive from their immediate predecessors have been received by p_i. RQ_i is initialized to have those tasks which have no predecessors.
- At each processor p_i, maintain a local clock LC_i which is always set to the maximum of the earliest time at which p_i is available for execution and the earliest time at which Q_i is not empty.
- Maintain a global clock GL which is always set to the minimum local clock among all processors.
- For each scheduling step at time GL, go to the available processor p_i (i.e. $LC_i = GL$), get the highest priority task from the front of its RQ_i and schedule it to start as early as possible on p_i. If two processors are available at the same time GL, the one with the smallest processor index is chosen.

After applying this execution ordering algorithm to the example in figures 8 and 9, the execution order of tasks assigned to p_1 will be the same because t_1, t_2 and t_5 all have precedence relations. However, in p_2, t_3 will be scheduledto start first before tasks t_4 and t_6 because t_3 has the highest exact level and, therefore, t_5 will be able to start execution on p_1 at time 6 and finish at time 8 leading to a shorter, closer to optimal makespan. The result of the execution ordering process is shown in figure 11. The cost of this schedule is $1 \times 8 + 1 \times 18 = 26$ cost *tokens*. The overall complexity of the assignment algorithm and the execution ordering one is $O(ne + n^2)$ where e is the number of communication links among different tasks and n is the number of tasks [13].

4.4 Experimental results

In order to show the effectiveness of our ETS algorithm, we conducted several experiments on random task and processor models, and compared the results of ETS algorithm with those of two other classical methods. Specifically, we compared ETS with LS, which is equivalent to ETS when β is set to 0 and α is set to 1 (i.e. minimizing the makespan only) and with single processor (SP) scheduling, which assigns all tasks to one processor that can execute them all with minimal total cost. In fact, there is not any other energy-aware

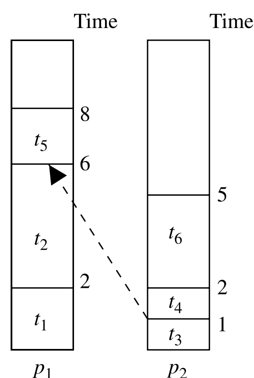


Figure 11. Execution ordering.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

scheduling algorithm to compare with the ETS, which imposed a restriction on our experiment. However, we chose LS to show that a time-based scheduling algorithm is not capable of conserving energy. And we chose SP to show that what all processors can do cooperatively is much better than what the best of them can do independently. The comparison was based on the multi-objective cost function discussed in Section 4.1, which consists of a delay metric and an energy one.

We used TGFF to generate random benchmarks. We used 200 benchmarks each of which has about 200 tasks and about 400 inter-task communication dependency links. The average execution time is 1000 μs and the average execution energy consumption is 1000 nJ. The average data units on each communication link is 100 bytes. We also generated a random processor model of five processors for each benchmark. The communication delay between any two processors has an average of 5 μs/byte (this includes the queuing, medium access, and transmission delays). The average communication energy consumption between any two processors is 5 nJ/byte. In this experiment, we set $\alpha = 1$ token/μs and $\beta = 1$ token/nJ to reflect equal importance weights of performance and energy conservation. Table 1 shows a comparison between ETS, LS and SP in terms of the average makespan, the average consumed energy, and the average total cost over all benchmarks. It turns out that while the makespan of the results generated by ETS is, on average, more than that of the results generated by LS by about 12%, the results generated by ETS consume, on average, 22% less energy compared to those generated by LS, and while the results generated by ETS consume, on average, 15% more energy compared to those generated by SP, the makespan of ETS results is, on average, only 20% of that of SP results. However, in terms of total cost, which is the most important factor, ETS excels. Figures 12–14 show detailed results of 20 randomly selected benchmarks.

Finally, it is worthwhile to notice that while this scheduling algorithm is more general than the allocation algorithm discussed in Section 3, the allocation algorithm is more suitable for the special cases where energy saving is the only concern. Moreover, the allocation algorithm has lower complexity.

5. Conclusion

Towards enabling ubiquitous, high performance mobile computing, we introduce the cooperative *ad hoc* computing paradigm which consists of heterogeneous mobile and stationary computing devices, which are connected via a wireless medium. Through the use of aremote execution platform, the involved devices can help each other in running different mobile applications (tasks) in order to save energy and improve performance. However, this environment can be abstracted to a set of consumers, i.e. mobile applications, and a set of resources, i.e. energy and processing power in each device. Therefore, an efficient scheduler, which finds proper execution plans, is needed; and this was the core of this paper.

Table 1. Comparison of different methods.

	Average makespan (μs)	*Average consumed energy (nJ)*	*Average total cost (cost tokens)*
ETS	45,627	270,631	316,259
LS	41,347	350,827	392,174
SP	236,816	229,367	466,184

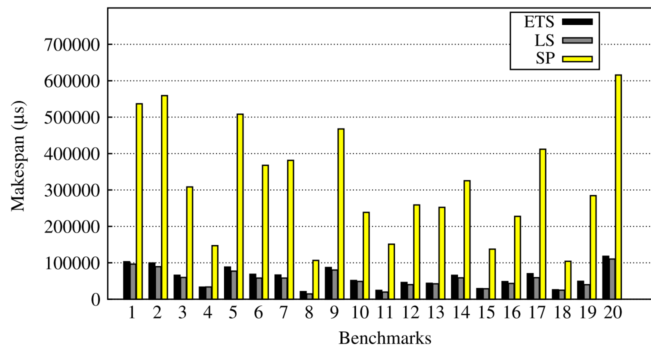


Figure 12. Makespan comparison.

What a mobile computing device can do is limited by how much energy is available in its battery. Having better utilization of device batteries, by designing more energy-efficient hardware and software components, is presumed to be a promising direction. However, even though several methods follow this direction, they are still limited by the amount of energy available in the battery. Computation offloading was the first, and the only, method that overcomes this limitation. Nevertheless, computation offloading assumes the availability of a wall-powered, stationary server or at least a more capable server (even if it is not stationary), and most of the work in this direction is limited to the case where there is one resource-limited device and one relatively more capable one.

In this paper, we introduce a cooperative computing paradigm which gives mobile computing devices the possibility to use all resources in nearby computing devices and provides the optimization methods that make efficient use of such a resource-restricted environment. Using the cooperative processing paradigm, a mobile computing device can perform a computational task that was impossible to be performed by the same device independently. Furthermore, a mobile computing device can have such a high performance that would not be achievable without the cooperation of other computing devices. Moreover, our scheme does not require the existence of a wall-powered server and does not limit the number of involved computing devices. Experiments show that the performance achieved with cooperative computing (i.e. all involved devices are working cooperatively) is much higher than that of the best involved computing device working independently.

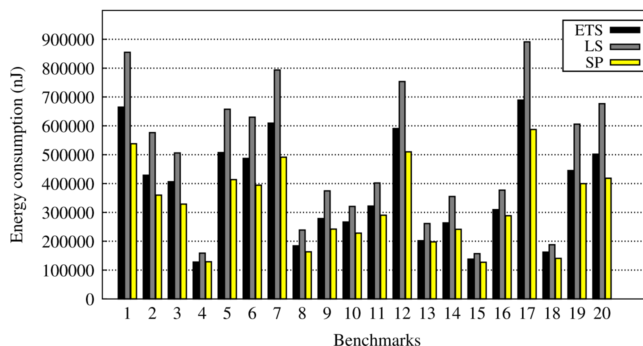


Figure 13. Energy consumption comparison.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

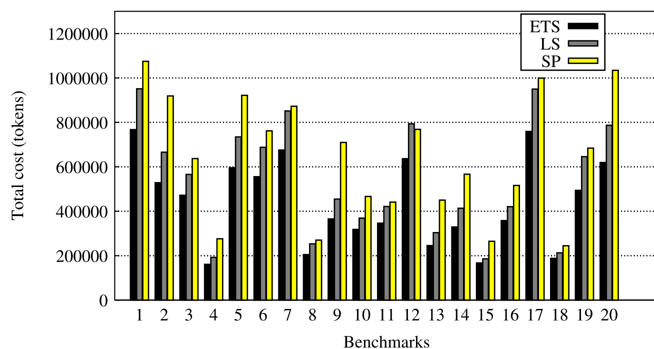


Figure 14. Total cost comparison.

We claim that our approach is practical and can be applied without significant modification to the existing systems. Recently developed remote execution platforms have made it feasible to migrate a computational task to a nearby device via a wireless medium. They are also able to predict the execution time and energy consumption of each task on each device by monitoring different resources and maintaining a database of resource usage statistics. Smart batteries, profiling, and algorithmic complexity are also used to predict resource usage of different tasks. However, the literature lacks a precise optimization method that finds good execution plan alternatives for such a scheme, and we believe that our scheduling and allocation algorithms form a pioneering effort in that direction.

We are now focusing on designing and implementing an *ad hoc* computing middleware that combines the remote execution platform and our optimization algorithms to enable mobile computing over wireless *ad hoc* networks. Besides that, we are working on another scheduling problem in which the objective is to maximize the lifetime of the system rather than minimizing the total consumed energy. Scheduling in computing environments which in cure a high degree of mobility is subject to more investigation [21].

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada, by Communications and Information Technology Ontario, and by the Government of Saudi Arabia.

References

[1] Satyanarayanan, M., 2001, Pervasive computing: vision and challenges, *IEEE Personal Communications*, 149–165, August.
[2] Lorincz, K., Malan, D., Fulford-Jones, T., Nawoj, A., Clavel, A., Shnayder, V., Mainland, G. and Welsh, M., 2004, Sensor networks for emergency response: challenges and opportunities, *IEEE Pervasive Computing*, **20**, 16–23, October–December.
[3] Gurun, S. and Krintz, C., 2003, Addressing the energy crisis in mobile computing with developing power aware software, UCSB, Computer Science Department, MA, Tech. Rep. 2003–15.
[4] Flinn, J., Narayanan, D. and Satyanarayanan, M., 2001, Self-tuned remote execution for pervasive computing, Proceedings of 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), May.

[5] Rudenko, A., Reiher, P., Popek, G. and Kuenning, G., 1999, The remote processing framework for portable computer power saving, Proceedings of ACM Symposium on Applied Computing, February.
[6] Weiser, M., 1991, The computer for the 21st century, *Scientific American*, **265**(3), 66–75, September .
[7] Stojmenovic, I., Russell, M. and Vukojevic, B., 2000, Depth first search and location based localized routing and QoS routing in wireless networks, Proceedings of International Conference on Parallel Processing, August.
[8] Helal, A., Mann, W., Elzabadani, H., King, J., Kaddourah, Y. and Jansen, E., 2005, Gator tech smart house: a programmable pervasive space, *IEEE Computer Magazine*, 64–74, March.
[9] Toh, C., 2002, *Ad Hoc Mobile Wireless Networks: Protocols and Syestems* (Englewood Cliffs: PTR Prentice Hall).
[10] Basagni, S., Conti, M., Giordano, S. and Stojmenovic, I., 2004, *Mobile Ad Hoc Networking* (Wiley-IEEE Press).
[11] E1-Rewini, H., Lewis, T. and Ali, H., 1994, *Task Scheduling in Parallel and Distributed Systems* (Englewood Cliffs: PTR Prentice Hall).
[12] Stone, H., 1977, Multiprocessor scheduling with the aid of network flow algorithms, *IEEE transactions of Software Engineering*, **SE-3**(1), 85–93, September.
[13] Alsalih, W., 2005, Energy-aware task scheduling, Master's thesis, Queen's University, Canada.
[14] Lo, V., 1988, Heuristic algorithms for task assignment in distributed systems, *IEEE Transactions on Computers*, **37**(11), 1384–1397, November.
[15] Dick, R., Rhodes, D. and Wolf, W., 1998, TGFF: task graphs for free, Proceedings of International Workshop on Hardware/Software Codesign, March.
[16] Alsalih, W., Akl, S. and Hassanein, H., 2005, Energy-aware task scheduling: towards enabling mobile computing over MANETs, Proceedings of IEEE Fifth International Workshop on Algorithms for Wireless, Mobile, *Ad Hoc* and Sensor Networks, April.
[17] Hu, T., 1961, Parallel sequencing and assembly line problems, *Operations Research*, **9**(6), 841–848 November.
[18] Coffman, E. and Graham, R., 1972, Optimal scheduling of two-processor systems, *Acta Informatica*, **1**(3), 200–213.
[19] Graham, R., 1966, Bounds for certain multiprocessing anomalies, *Bell Systm Technical Journal*, (1), 1563–1581, November.
[20] Yang, T. and Gerasoulis, A., 1993, List scheduling with and without communication delays, *Parallel Computing*, **19**(12), 1321–1344, December.
[21] Alsalih, W., Akl, S. and Hassanein, H., 2006, Real time task scheduling in MANETs, manuscript in preparation.

Downloaded by ["Queen's University Libraries, Kingston"] at 11:54 07 July 2015