Dynamic Resource Management in Energy Constrained Heterogeneous Computing Systems Using Voltage Scaling

Jong-Kook Kim, *Member*, *IEEE*, Howard Jay Siegel, *Fellow*, *IEEE*, Anthony A. Maciejewski, *Fellow*, *IEEE*, and Rudolf Eigenmann, *Senior Member*, *IEEE*

Abstract—An ad hoc grid is a wireless heterogeneous computing environment without a fixed infrastructure. This study considers wireless devices that have different capabilities, have limited battery capacity, support dynamic voltage scaling, and are expected to be used for eight hours at a time and then recharged. To maximize the performance of the system, it is essential to assign resources to tasks (match) and order the execution of tasks on each resource (schedule) in a manner that exploits the heterogeneity of the resources and tasks while considering the energy constraints of the devices. In the single-hop ad hoc grid heterogeneous environment considered in this study, tasks arrive unpredictably, are independent (i.e., no precedent constraints for tasks) and have priorities and deadlines. The problem is to map (match and schedule) tasks onto devices such that the number of highest priority tasks completed by their deadlines during eight hours is maximized while efficiently utilizing the overall system energy. A model for dynamically mapping tasks onto wireless devices is introduced. Seven dynamic mapping heuristics for this environment are designed and compared to each other and to a mathematical bound.

Index Terms—Ad hoc, distributed heterogeneous computing, dynamic resource allocation/management, dynamic voltage scaling, energy-aware computing, task priorities and deadlines.

1 Introduction

An *ad hoc grid* is a *heterogeneous computing (HC)* environment consisting of mobile battery-powered computing devices that communicate using wireless connections. Ad hoc grid (ad hoc networked) environments enable users to communicate and share computational load and results with other users in the system to coordinate efforts to accomplish a mission. Examples of applications of ad hoc grids include wildfire fighting, disaster management, and military situations [31]. HC is the coordinated use of various resources with different capabilities to satisfy the requirements of varying task/application mixtures. When the resources are wireless and mobile, the limited battery capacity becomes a constraint and power or energy management becomes a critical issue. As devices are heterogeneous, battery capacity may also be heterogeneous. The heterogeneity of the resources and tasks in an HC system is exploited to maximize the performance or the cost-effectiveness of the system (e.g., [9], [14], [18], and [30]). An important research problem is how to assign resources to the tasks (*match*) and to order the tasks for execution on the resources (*schedule*) to maximize some performance criterion of an HC system. This procedure is called *mapping* or *resource allocation*. A *resource management system (RMS)* takes care of allocating resources of a certain system. The power management aspect further complicates this problem.

Two different types of mapping are static and dynamic. *Static mapping* is performed when tasks are mapped in an offline planning phase, e.g., planning the schedule for a set of production jobs. *Dynamic mapping* is performed when the tasks are mapped in an online fashion, e.g., when tasks arrive at unpredictable intervals and are mapped as they arrive (workload is not known *a priori*). In both cases, the mapping problem has been shown, in general, to be NP-complete (e.g., [12], [15], and [22]). Thus, the development of heuristic techniques to find near-optimal solutions for the problem is an active research area (e.g., [7], [8], [9], [10], [14], [16], [17], [26], [29], [33], [40], and [44]).

In this research, the *dynamic mapping* of tasks onto devices is studied. Simulation is used for the evaluation and comparison of the heuristics developed in this paper. As described in [29], dynamic mapping heuristics can be grouped into two categories, immediate and batch mode. Each time a mapping is performed, *immediate mode* heuristics only consider the new task for mapping, whereas *batch mode* may consider the new task and tasks awaiting execution, thus having more information about the task mixture before mapping. Both approaches are attempted in this paper.

- *J.-K. Kim is with the School of Electrical Engineering, Korea University, Anam-Dong, Sungbuk-Gu, Seoul 136-701, South Korea. E-mail: jongkook@korea.ac.kr.*
- *H.J. Siegel is with the Department of Electrical and Computer Engineering and the Department of Computer Science, Colorado State University, Fort Collins, CO 80523. E-mail: hj@colostate.edu.*
- *A.A. Maciejewski is with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: aam@colostate.edu.*
- *R. Eigenmann is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285. E-mail: eigenman@ecn.purdue.edu.*

Manuscript received 16 Oct. 2007; revised 3 Apr. 2008; accepted 10 June 2008; published online 20 July 2008.
Recommended for acceptance by I. Ahmad, K.W. Cameron, and R. Melhem.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2007-10-0380.
Digital Object Identifier no. 10.1109/TPDS.2008.113.

The power management is accomplished by using *dynamic voltage scaling (DVS)* [39]. DVS is based on exploiting the relationship between the CPU supply voltage of a device and the power usage (e.g., Crusoe [11] and ARM7D [5]). The relationship between power and energy is that *energy* consumed is *power* multiplied by the amount of time that power is used. The relationship of power to voltage is a strictly increasing convex function, represented by a polynomial of at least second degree [21]. Most processors that support DVS use discrete levels. The DVS technique allows the reduction of a CPU's energy usage (through CPU voltage (clock frequency) reduction) at the expense of increasing the task execution time. The DVS mechanism in this research will be managed by the system administrator or the resource manager and is transparent to the user.

In the environment for this research, the devices are wireless and can communicate with each other (e.g., peer to peer communication). An example scenario can be a wildfire-fighting situation in a remote area (e.g., controlling a forest fire), where the firefighters are equipped with mobile devices that will form an *ad hoc* network with no base station. For scenarios such as these, devices in this research are assumed to be close enough to allow a single-hop ad hoc network. The batteries for these devices are assumed to be recharged after a certain amount of time (e.g., recharged after an eight hour shift or work day). For example, in the fire-fighting scenario, it is typical for firefighters to have a scheduled break for food and rest after a shift and so can recharge their batteries at that time. Using a device, a user can request a program (*task*) to be executed, receive data, and send data. A device performing a computation may receive input data from other devices or external sources. The resulting output will be sent back to the task requester.

For the efficient use of the overall system energy available, it may be best for certain tasks to be executed on a remote, rather than the local, device. The reasons are 1) limited energy remaining on the local device, 2) a remote device can execute the task using less energy, and 3) a remote device can complete the task by its deadline. An RMS makes this decision of locating a "suitable" device.

Tasks can have different priority levels (i.e., high, medium, or low) and a deadline. The primary goal of this research is to complete as many high-priority tasks by their deadlines as possible during a given interval of time (i.e., eight hours). The secondary performance goal is to maximize the sum of the weighted priorities of medium- and low-priority tasks completed by their deadlines during that interval of time. This sum builds on our FISC measure in [25]. The motivation of using these two performance metrics instead of makespan is that the number of tasks introduced to the system can be huge, and therefore, not all tasks can be completed. The important objective of an RMS is to complete as many tasks as possible while taking the system level energy into consideration. The reason for the primary goal is because, in this environment, high-priority level tasks are considered to be infinitely more important than medium- and low-priority tasks and need to be completed. For the secondary goal, it is still beneficial to complete as many tasks as possible but because tasks have different priorities (value) the goal considers this difference.

We want to design resource management heuristics that will generate robust resource allocations [2]. Consider the three robustness questions from that in [3] for this environment. The first question is: what behavior of the system makes it robust? Here, we say the system is operating in a robust way if it can execute all of the high-priority tasks. The second question is: what uncertainty is the system robust against? In this study, it is the uncertainty of which, when, and how quickly tasks of different priorities will arrive. The third question is: quantitatively, how robust is the system? If we strictly enforce the robustness requirement of completing all of the high-priority tasks, then the robustness metric that can be used to compare two different resource allocations that complete all the high-priority tasks is the value of the medium and low-priority tasks it can complete in addition to the high0priority tasks. If neither resource allocation can complete all of the high-priority tasks, neither meets the strict requirement, but the one that completes a greater percentage of the high-priority tasks is better. Alternatively, in the situation where the system is so oversubscribed that none of the heuristics employed can complete all of the high-priority tasks, the robust requirement can be relaxed to be that a given prespecified percentage (less than 100) of the high-priority tasks complete.

The contributions of this research include 1) the modeling of dynamically mapping tasks onto wireless devices while managing power using the DVS method, 2) the design, analysis, and comparison of seven resource allocation methods for this environment, and 3) the mathematical bound derivations on the heuristics performance.

Section 2 discusses the heterogeneous ad hoc environment followed by a summary of the literature related to this work. In Section 4, the heuristics studied in this research are presented. Section 5 describes the simulation setup. The results are examined in Section 6, and the last section gives a brief summary of this research.

2 Energy Constrained Environment

The *ad hoc* grid environment is controlled by a *resource management system (RMS)*. The RMS performs matching, scheduling, and power management to maximize the goal stated earlier. In this environment, the wireless devices have limited battery capacity (energy). The users are allowed one battery for the operation of a given device for an interval of time. The batteries are recharged after eight hours, and the battery capacity is different for different devices. The devices employ DVS for power management. The number and value of the discrete voltage levels may vary among the devices.

The users send task requests to the RMS. Once a task request is received, the RMS locates a "suitable" device and sends a task execution command (Fig. 1). If an input data is required, the data is communicated directly to the executing device from the source. A source could be other wireless devices or outside sources (e.g., a weather station). The result of the task execution (e.g., a wind direction estimate) is sent back to the task requester device, if the task was not executed on that device. The tasks discussed here have a priority level (e.g., high, medium, or low) and a deadline. If the task cannot complete by its deadline, it has no value.

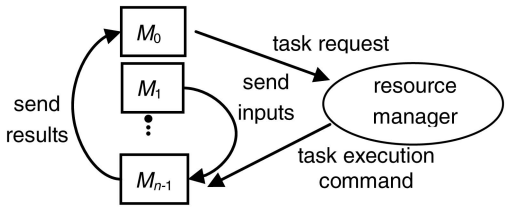

Fig. 1. The ad hoc grid HC environment model considered in this study.

The communication of inputs and results is assumed to be done directly from device to device (i.e., single-hop ad hoc network) using the IEEE 802.11b standard (a popular wireless standard). In this research, only one device receives or sends data at any instant in time. This scheme is desirable when a certain quality of service must be met for the tasks. If other communications are allowed while a task is still communicating, then the communication time for that task is no longer guaranteed, which complicates the quantifications of the communication time. A time division multiplexed communication scheme may be considered in future work.

In this environment, it is assumed that the types of devices that may connect to the system are known. In addition, there is a predetermined set of tasks that a user can request. However, it is not known a priori exactly which tasks will be requested and when they will be requested. In an example military scenario, there are predetermined types of wireless devices allowed to connect to the military system. In this environment, there is a set of tasks that may be requested for execution (e.g., target determination and troop deployment decisions). A requested task is executed on a "suitable" device, and the information is sent back to the task requester. Because it is assumed that all the devices and the tasks are known, the task execution times on those devices are assumed to be known to the RMS (e.g., execution times can be determined by running the tasks on the devices). The estimated execution times of each task on each machine is assumed to be known based on user supplied information, experiential data, task profiling and analytical benchmarking, or other techniques (e.g., [1], [18], [19], [24], and [42]). Determination of the execution times is a separate research problem, and the assumption of such information is a common practice in mapping research (e.g., [19], [23], [24], [27], [37], and [41]).

It is assumed that all devices are equipped with all programs required and only input data is needed to execute a task and send back results. Thus, the time to communicate a task request to the RMS and to send a task execution command to a device is assumed to be negligible. We make the simplifying assumption that the RMS is located on a dedicated machine that has unlimited power and that the devices are within transmission range of the RMS (the relaxation of these assumptions may be considered in future work). In a real environment, the RMS would not have a machine with unlimited energy, and so, its consumption would need to be added to the model.

3 Related Work

There has been much research on power constrained (power-aware) resource management in uniprocessors (e.g., [6], [20], [32], and [43]). The research in [6] presents a static scheduling solution of periodic tasks on a processor assuming the worst-case scenario, a dynamic reclaiming algorithm for tasks that complete before their worst-case scenario, and an adaptive speed adjustment mechanism to anticipate the probable early completion of future task executions. A power minimizing approach for variable-voltage systems is developed in [20], where tasks are periodic and independent. The method described in [32] assumes a dynamic preemptive environment where periodic independent tasks arrive and leave a system. In [43], a formal analysis of the minimum energy scheduling problem is provided for a single processor and a model that assumes a task with an arrival time and deadline. The difference between these studies and our research is that our energy constrained ad hoc grid environment considers multiple heterogeneous devices and nonperiodic independent tasks with priorities and deadlines that need input and/or output communicated. The fact that our environment has heterogeneous multiple devices adds new issues to the resource allocation problem.

Some research projects have explored a multiprocessor environment with static resource management (e.g., [13], [34], [45], and [46]). In [13], a genetic algorithm is used to synthesize distributed heterogeneous embedded systems. Using a static schedule derived from a list scheduling scheme, the study in [34] does static and dynamic power management. The work in [45] describes a linear programming method that statically schedules periodic tasks on heterogeneous processing elements. The research in [46] assumes homogeneous processors and frame-based tasks. In static mapping, information of all tasks is known and the execution time of the heuristic itself is not a constraint. The difference is that our research explores a dynamic environment where the arrival time of a task is not known prior to its arrival and the task mapping time must be fast.

The research in [28] statically schedules periodic tasks onto homogeneous processing elements first using the tool in [13], and then, slots are created in this static schedule to accommodate aperiodic tasks with hard deadlines. They assume that the minimum interval between two hard aperiodic tasks is larger than the lowest common multiple period of all periodic tasks. Then, an online scheduler modifies the system to minimize the response times for aperiodic tasks with soft deadlines. The static schedule is unchanged and the soft aperiodic tasks are run when there is unused time. In our research, all of the devices are heterogeneous and all tasks are aperiodic with hard deadlines. Because all tasks are aperiodic, slots are not created among task periods, i.e., the RMS approaches are quite different. Furthermore, our research considers the case where not all tasks with hard deadlines can complete and does not assume a minimum interval between the arrivals of two tasks.

The research in [38] tries to send tasks to another device to be computed. It uses a distributed economic-based subcontracting protocol to determine which device to use. The goal of the devices in [38] is to find a suitable device that can execute tasks to save energy. A cost is associated with devices that are willing to execute a task for other devices. The device that wants to move one of its tasks to

another device bargains with those willing devices. The underlying model of our work differs in that the environment in our research assumes that all devices are capable of DVS and tasks have deadlines and priorities.

The research in [35] and [36] studies static RMSs for minimizing energy consumption for a heterogeneous ad hoc grid. The differences are that in our research, the heuristics operate dynamically, each device supports DVS, tasks have priorities, and it is assumed that not all tasks are completed before their hard deadlines.

4 Heuristic Descriptions

4.1 Mapping Event

A dynamic mapping approach is designed to assign resources to new tasks faster than the anticipated average arrival rate of the tasks. Therefore, the heuristics that are developed have a limit on the time each computation of a new mapping can take.

A *mapping event* occurs when a new task arrives. For immediate mode heuristics, at any mapping event, only the new task is considered for mapping onto devices. For batch mode heuristics, at any mapping event, the new task and the tasks in the device queues still awaiting execution are considered together for device assignment, i.e., previously mapped, but unexecuted tasks can be *remapped*. The exception is that the first task in each machine's wait queue (this task is not the task that is currently executing) is not considered remapping. The reason for this is to reduce the chance of a device becoming idle if during a mapping event the currently executing task finishes. While it is still possible that a device may become idle, it is highly unlikely for the assumptions in this research (mean execution times of tasks and mean execution times of mapping events described in Sections 5 and 6, respectively). These tasks that are considered for remapping are called *mappable tasks*. If a task arrives while a mapping event is in progress, the current mapping event is not disturbed. When the current mapping event is completed, the next mapping event starts and includes any tasks that have arrived.

4.2 Scheduling Communications

The following are same for all heuristic approaches. All communications are scheduled as early as possible. If there are previous communications scheduled, then current ones are inserted in the gaps between the ones already scheduled if possible, or else, they are put at the end of the communication scheduling queue. It is assumed that a communication between one device and another is not broken. Communications from different sources can be scheduled in different gaps.

4.3 Opportunistic Load Balancing (OLB) and Minimum Energy Greedy (MEG)

The immediate mode *OLB* heuristic is a common method for scheduling tasks. At a mapping event, among the devices that can map the new task without violating its deadline and have enough energy to complete the task, the heuristic selects the device that will be ready (i.e., executes all the tasks already in its queue) first to map the new task. This is a simplistic method that ignores the relationship between the needs of the task to be assigned and the capability of the devices in the ad hoc grid.

At a mapping event, the immediate mode *MEG* heuristic selects the device that can complete the task by its deadline and executes the task using the minimum amount of energy. This is a scheme that ignores other tasks that are already in the system.

The following is same for both heuristics. If no device can complete the task by its deadline, the task is deleted from the system. The *energy consumed status and the device availability status (system status)* is updated at every mapping event.

4.4 Minimum Energy Minimum Completion Time (ME-MC) Heuristic

The immediate mode *ME-MC* heuristic is based on the general concept of the switching algorithm in [29]. The basic idea behind this heuristic is to first try to map tasks onto their "best" machine according to some metric. But, when the load on the system becomes unbalanced, the strategy is changed to balance the load. When the load is balanced, then the scheme is changed back to the "best" machine method. For this method, a load balance ratio is used to determine whether the system is load balanced.

In this study, two different load balance ratios are calculated. One is for the high-priority tasks and other is for the medium- and low-priority tasks. The reason for the two different load balance ratios is that when high-priority tasks arrive, the high-priority tasks are inserted behind the last high-priority task in front of all medium- and low-priority tasks or at the front of a device's wait queue. The *primary load balance ratio* is the ratio of the earliest device availability time over all the devices in the suite to the latest device availability time. For this ratio, the device availability times are determined using the last high-priority task in each queue. If there are no high-priority tasks in a device queue, then the device available time is the completion time of the task that is running if it is the only task on the device. If there are other tasks on the device, then the device available time is the completion time of the first waiting task. The *secondary load balance ratio* is same as the primary load balance ratio except that it is calculated with all tasks. For both load balance ratios, a common high threshold and low threshold are established by experimentation (high threshold $>$ low threshold).

Initially, the system maps new tasks onto their minimum energy consumption device using the slowest speed level. If the task that arrived is a high-priority task and there are no devices that can complete the high-priority task by its deadline, then the speed level of the devices is increased starting from device 0 using the method described below to test if there are devices that can complete the high-priority task with a speed level increase. When increasing a device's speed level, the total number of speed levels of a device is taken into consideration. For example, assume a device 1 that has 16 speed levels and another device 2 that has four speed levels. If device 1 increased its speed levels at least four times, only then can device 2 be considered for speed level increase. Only the speed level for the device finally selected for mapping is increased. Once the speed level of a device is increased to a faster level, the device will not try to

execute tasks at a lower speed level later. All tasks mapped earlier will complete faster than when the speed level was lower (before the speed level is increased), thus guaranteeing that tasks mapped earlier complete by their deadline. At any mapping event, the speed level is increased at most two times. This is to avoid increasing the speed level to accommodate the current task while not leaving enough energy for future use.

The Switching Algorithm heuristic can be summarized by the following procedure. The *total energy consumed* is equal to the total CPU energy used plus the energy used for communication (details of CPU and communication energy are discussed in Section 5):

1. Determine the priority level of the new task.
2. Calculate the primary (or secondary) load balance ratio.
3. If the primary (or secondary) load balance ratio $>$ high threshold, then current method is to use the minimum *energy consumption* device to map the new task.
   If the primary (or secondary) load balance ratio $<$ low threshold, then current method is to use the minimum *completion* time device to map the new task.
   If low threshold $\leq$ primary (or secondary) load balance ratio $\leq$ high threshold, then current method is the one used at the previous mapping event to map the new task.
4. If the task is a medium- or low-priority task, assuming that it will be mapped at the end of a device queue, determine all devices that can complete the task by its deadline
   if the task cannot be completed on any device, it is deleted from the system
   else, select a device using the current method, map the task to this device, and all communications are scheduled using the method in Section 4.2.
5. Initialize "iteration" to the number of speed level changes on the device where the speed level was changed the most.
   If the task is a high-priority task, assuming that it will be mapped (inserted) after the last high-priority task in a device queue, determine all devices that can complete the task by its deadline.
   do until a device is selected for mapping or iteration is increased twice.
   if the task cannot be completed its deadline on any device, increase the speed level *(note that when trying to increase a device's speed level, the total number of speed levels of a device is taken into consideration).*
   for each device, increase one speed level if the (maximum number of speed levels over all devices)/(total number of levels on the device) $\leq$ iteration and test if the device can complete the task.
   $\text{iteration} = \text{iteration} + 1$
   else, select a device using the current method, map the task to this device
   if the task cannot be completed on any device, return all device's speed level to the level before this task arrived and drop the task.
   else, return all unselected devices' speed level to the level before this task arrived.
6. Check all devices as follows: If there is enough energy on a device to continually execute at the highest speed level and transmit data for the rest of the remaining time (until the end of the eight hour period), then the speed level for that device is increased to the highest speed level.
7. Update the system status.

4.5 Minimum Energy Minimum Energy (ME-ME) Heuristic

The batch mode *ME-ME* heuristic is based on the general concept of the Min-Min (greedy) idea in [22]. The Min-Min type heuristic performed very well in previous studies of different environments (e.g., [10] and [29]). The basic idea of a Min-Min type heuristic is to find the "best" device for all tasks that are considered, and then among these task/device pairs, it selects the "best" pair to map first. To determine which device or which task/device pair is the best, a fitness value is used. The *fitness value* of a task on a given device for this study is 1) the energy consumed for high-priority tasks, and 2) the energy consumed multiplied by the weighted priority divided by the execution time of the task for medium- and low-priority tasks. The energy consumed is equal to the energy used by the CPU plus the energy used for communication. This method also starts the simulation by using the slowest speed level of devices to map tasks.

The ME-ME procedure starts at a mapping event, and it is assumed that none of the mappable tasks are mapped, i.e., they are not in any device queue.

1. All high-priority tasks are considered first, then the other tasks are considered.
2. All high-priority tasks in the mappable task list are checked to see if they can be completed by their deadline.
3. If there are some tasks that cannot be completed on any device, then the speed level is increased or the task is dropped using the method detailed within step 5 of Section 4.4.
4. For each high-priority task in the mappable task list, find the device that gives the task its minimum fitness value (the first "ME") among the devices that can complete the task by its deadline using the current speed level and ignoring other tasks in the mappable task list.
5. Among all the task/device pairs found from above, find the pair that gives the minimum fitness value (the second "ME"), map the task to the device, and remove the task from the mappable task list.
   Input or results communication is scheduled using the method in Section 4.2.
6. Update the system status.
7. Do steps 2 to 6 until all high-priority tasks are mapped and then do the same for medium- and low-priority tasks except the speed level is not increased.
8. Check all devices as follows: If there is enough energy on a device to continually execute at the

highest speed level and transmit data for the rest of the remaining time (until the end of the eight hour period), then the speed level for that device is increased to the highest speed level.
9. Update the system status.

4.6 Contention Resolved Minimum Energy (CRME) Heuristic

The batch mode *CRME* heuristic is based on the general concept of the suffrage idea in [29]. The CRME heuristic applies the same fitness value calculation used in the ME-ME heuristic (Section 4.5), but when deciding which task to map, the task that "suffers" most if not mapped to its "first choice machine" is selected.

The CRME procedure starts at a mapping event. When the mapping event begins, it is assumed that none of the mappable tasks are mapped, i.e., they are not in any device queue.

1. All high-priority tasks are considered first, then the other tasks are considered.
2. All high-priority tasks in the mappable task list are checked if they can be completed by their deadline.
3. If there are some tasks that cannot be completed on any device, then the speed level is increased or the task is dropped using the method detailed within step 5 of Section 4.4.
4. For each task in the mappable task list, find the device that gives the task its minimum fitness value among the devices that can complete the task by its deadline using the current speed level, ignoring other tasks in the task list.
5. If there is contention among any of the high-priority tasks (i.e., two or more high-priority tasks have the same minimum fitness value device), select the task that will suffer the most (the task with the largest difference of fitness value between the best and the second best devices) to map onto the device selected. Else, map all the high-priority tasks. All communications are scheduled using the method in Section 4.2.
6. Remove the above mapped task(s) from the mappable task list.
7. Update the device availability and energy consumed status.
8. Repeat steps 2 to 7 until all high-priority tasks are mapped and do the same for the medium- or low-priority tasks except the speed level is not increased.
9. Check all devices as follows: If there is enough energy on a device to continually execute at the highest speed level and transmit data for the rest of the remaining time (until the end of the eight hour period), then the speed level for that device is increased to the highest speed level.
10. Update the system status.

4.7 Originator and Random

The immediate mode *originator* heuristic executes the task on the device that originated the task. This heuristic is run to compare to the performance of heuristics that utilizes other devices in the system. The immediate mode *random* heuristic maps the new task on a randomly selected device when the new task arrives. This heuristic is run to compare to the performance of the guided heuristics. The following is for both heuristics. The method in Section 4.2 is used for communication scheduling. If the selected device cannot complete the task by its deadline or there is not enough energy to complete the task, the task is deleted from the system. The energy consumed status is updated at every mapping event.

4.8 Upper Bound (UB)

Two UB methods are presented in this section. Each time the environment is simulated, the overall *UB* is determined by selecting the tighter bound of the two methods.

The *first UB (UB1)* uses the arrival time of tasks, priority of tasks, the deadline of the tasks, and the time interval between the arrivals of tasks based on the UB in [26]. The bound ignores the communication and the energy consumed. The tasks that have arrived before or at the mapping event are called *selectable tasks*. At any mapping event, only the selectable tasks are considered for the calculation of the UB. Let $ETC(i,j)$ be the *estimated time* to *complete* of task i on device j, and let Q_i be equal to the priority weighting of task i divided by the minimum $\mathrm{ETC}(i,j)$ over all machines.

The scheme starts by initializing all tasks' *remaining ETC* values, $\mathrm{rETC}(i,j)$, to the minimum $\mathrm{ETC}(i,j)$ over all devices. The UB1 follows the procedure described below:

1. At a mapping event, determine the *total aggregate computation time (TACT)* until the next task arrives. That is, TACT = time interval between arrival times of the new task and the next task multiplied by the number of machines.
2. Selectable tasks with $\mathrm{rETC}(i,j) > 0$ are put in a task list.
3. Sort high-priority tasks in the task list using minimum ETC values. Then, the medium and low-priority tasks are sorted together based on Q_i.
4. If there are high-priority tasks in the task list, select the high-priority task a that has the minimum ETC value. Else, select the medium/low-priority task a with the highest Q_a from the task list.
5. If $\mathrm{TACT} < \mathrm{rETC}(a,j)$.
   - if the selected task is high-priority,
     - subtract TACT from $\mathrm{rETC}(a,j)$
   - if the selected task is medium or low-priority
     - add $(Q_a \times \mathrm{TACT})$ to the secondary metric
     - subtract TACT from $\mathrm{rETC}(a,j)$
     - done (i.e., $\mathrm{TACT} = 0$)
   - if $\mathrm{TACT} \geq \mathrm{rETC}(i,j)$
     - if the selected task is high-priority
       - add one to the primary metric (i.e., the number of high-priority tasks completed)
       - subtract $\mathrm{rETC}(a,j)$ from TACT (this becomes the new TACT), $\mathrm{rETC}(a,j) = 0$
     - if the selected task is medium or low-priority
       - add $(Q_a \times \mathrm{TACT})$ to the secondary metric (i.e., the sum of the weighted priorities of medium and low-priority tasks)

subtract $\mathrm{rETC}(a,j)$ from TACT (this becomes the new TACT), $\mathrm{rETC}(a,j) = 0$

6. Repeat steps 4 and 5 until TACT is equal to 0 or there are no selectable tasks with $\mathrm{rETC}(a,j) > 0$.
7. Repeat steps 1 to 6 until the end of the simulation.

The *second UB (UB2)* uses the energy consumed information of tasks. The *total energy available* is the sum of all devices' maximum energy available. The energy consumed is equal to the energy used by the CPU plus the energy used for communication. The UB2 starts by determining the minimum energy consumed over all devices for each task. Then, the high-priority tasks are ordered in the task list using minimum energy consumed and then the medium and low-priority tasks are ordered using the minimum energy consumed divided by the weighted priority. Using this order, the number of tasks completed is computed by adding the energy consumed by the tasks until the sum exceeds the total energy available.

While two methods were attempted, UB1 was always tighter than UB2 for the cases considered here. This is despite the fact that, in general, UB1 is an unreachable loose bound for this environment.

The UB calculation explicitly considers all the high-priority tasks first for completion, and then, if the system has resources left they are used for the medium and low-priority tasks. Recall that the primary goal of this research is to complete as many high-priority tasks as possible. Therefore, the UB for the medium and low-priority task completed is shown (in the results) only when a heuristic can achieve the UB on the high-priority task. Only then is it valid to compare the medium and low-priority tasks completed against the UB calculated.

5 Simulation Model

Ten types of wireless computing devices and 50 task types are used in the simulated system. Because the devices and the tasks are known, the *estimated time* to *compute (ETC)* each of tasks on each of these different devices is known. In each simulation of a system, eight devices are picked with equal probability. The arrival of tasks is simulated by mean intertask arrival times using a (memoryless) Poisson distribution. Three scenarios with mean intertask arrival times of 10, 8, and 6 seconds are considered. The mean intertask arrival times are given to loosely generate more and more tasks for the system to handle. Where, at the beginning, the system can handle most of the tasks and later, where there are a lot of tasks, the system could only complete a percentage of the tasks. The system is simulated for 480 minutes (i.e., eight hour work time), with eight bursty periods of 10 minutes that do not overlap with each other. The bursty periods have faster arrival rates (mean is twice as fast as the rate of the normal period).

A 10×50 ETC matrix of the 50 types of tasks on 10 types of devices taking heterogeneity into consideration is generated using the gamma distribution method described in [4], with a COV of 0.9 for task heterogeneity and a COV of 0.6 for device heterogeneity. Two means, 60 and 600 seconds, are used for the ETC matrix. The mean execution time is chosen to represent applications such as downloading files (such as maps or weather reports), generating strategies, etc. When a task is determined to arrive, one of the 50 task types is selected with equal probability. A *trial* is defined as one such simulation of the HC system (one 10×50 ET matrix). For each of the six scenarios (three mean intertask arrival time multiplied by two mean execution times), 50 trials are run for all heuristics.

Each task is assigned a priority level of high, medium, or low, with equal likelihood. The priority levels of medium and low are given a weighting of four and one. This weighting is to calculate the performance of the value of medium- and low-priority tasks completed by their deadlines (secondary goal) if the number of high-priority tasks completed by their deadlines (primary goal) is comparable for some heuristics.

For each device, the maximum battery capacity, the maximum CPU energy consumption rate, and the number of discrete levels for DVS are given. The discrete levels for DVS correspond to the speed at which the CPU is run and defined as *speed levels*. The environment assumes the IEEE 802.11b standard for wireless communication. It is assumed that the data communication and the task computation or execution can be done simultaneously. Based on two types of wireless devices (a laptop and a handheld), the energy consumption rates are determined. These two devices can be selected with equal probability. The maximum CPU energy consumption rates are determined using a uniform distribution with a range of 0.1 to 0.3 for laptops or 0.01 to 0.03 for handheld devices. The reason for the two ranges is that the CPU energy consumption rate of a laptop is about 10 times higher than that of a handheld device (based on sample devices from the Dell website). Based on sample communication adapters (e.g., Linksys) for the two types of devices, the transmission energy consumption rate is 0.6 (about three times the CPU energy consumption rate of a laptop) or 0.2 (about 1/3 of transmission energy consumption rate of a laptop) for the laptops or the handhelds, respectively. The reception energy consumption rate and the idle (communication) energy consumption rate are assumed to be 65 percent and 25 percent of the transmission power consumption rate, respectively. For the simulation study, the maximum battery capacity (energy) of device j, $\mathrm{BC}(j)$, is set to the maximum CPU energy consumption rate plus the transmission energy consumption rate, multiplied by the maximum operation time. The maximum operation time is determined using a uniform distribution with a range of one to two hours. This means that if the CPU is used at the maximum speed level, and the device is always transmitting, then the battery capacity is only enough to operate the device for one to two hours.

To simplify DVS, this research assumes that each voltage level of a processor corresponds to a clock speed level for the processor. Each device can have 2, 4, 8, or 16 discrete speed levels with equal probability. After the number of levels is decided, the relative speed of each level is determined. The lowest speed level of a device is assumed to be one third of the maximum speed level (e.g., if the maximum speed level is 1.2 GHz, then the lowest speed level will be 400 MHz). We make the simplifying assumption that task execution time varies linearly with the discrete speed level. It is assumed that the voltage switching is done

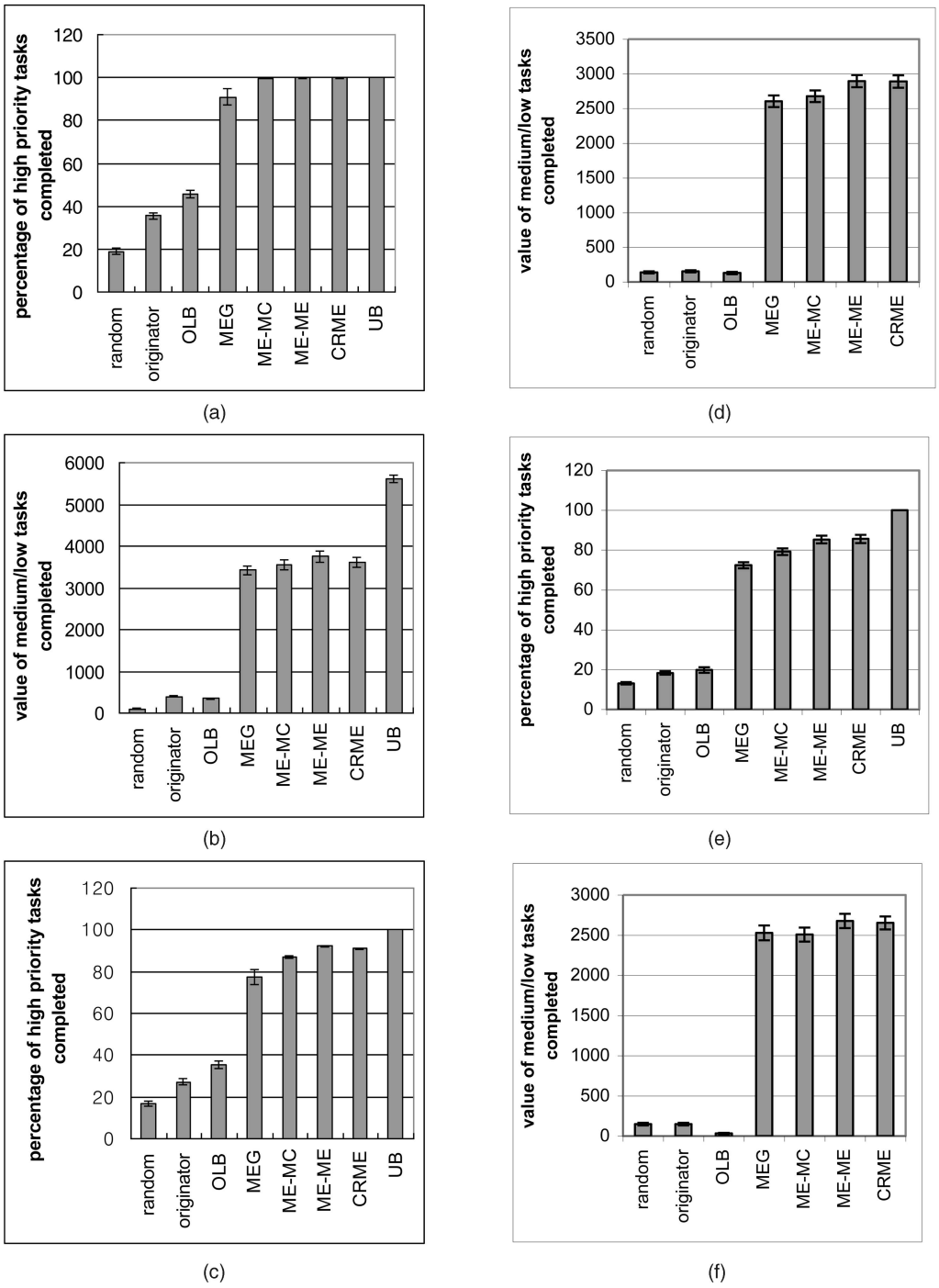

Fig. 2. The simulation results using the mean execution time of 60 seconds and mean intertask arrival of 10 seconds for (a) and (b), 8 seconds for (c) and (d), and 6 seconds for (e) and (f): (a), (c), and (e) show the percentage of high-priority tasks completed, and (b), (d), and (f) show the value of medium and low-priority tasks completed.

dynamically and that the overhead associated with the switching is negligible $(20\ \mu \sim 150\ \mu \mathrm{s})$. The power consumption as a function of speed (voltage) levels is assumed to be a quadratic function. For the example with four speed levels, assume that the maximum energy consumption rate is $\alpha = 0.16$. Using a simple equation of $\text{maximum energy consumption rate} = \alpha \times (\text{relative speed of a speed level to the maximum speed level})^2$, where α is 0.16. The relative speed of the slowest speed level is $1/3$ of the maximum speed level, next will be $5/9$ and $7/9$ of the maximum speed level (linear). Using these fractions, the energy consumption rates for each speed level are calculated. In this example, the energy consumption rates would be $0.16 \times 1/9$, $0.16 \times 25/81$, $0.16 \times 49/81$, and 0.16 from the slowest speed level to the fastest (maximum) speed level, respectively. When the CPU of the device is idle, the CPU energy consumption rate is assumed to be $1/12$ of the maximum energy consumption rate.

The eight devices are assumed to transmit and receive at the speed of 1 Mbps. When tasks need to communicate

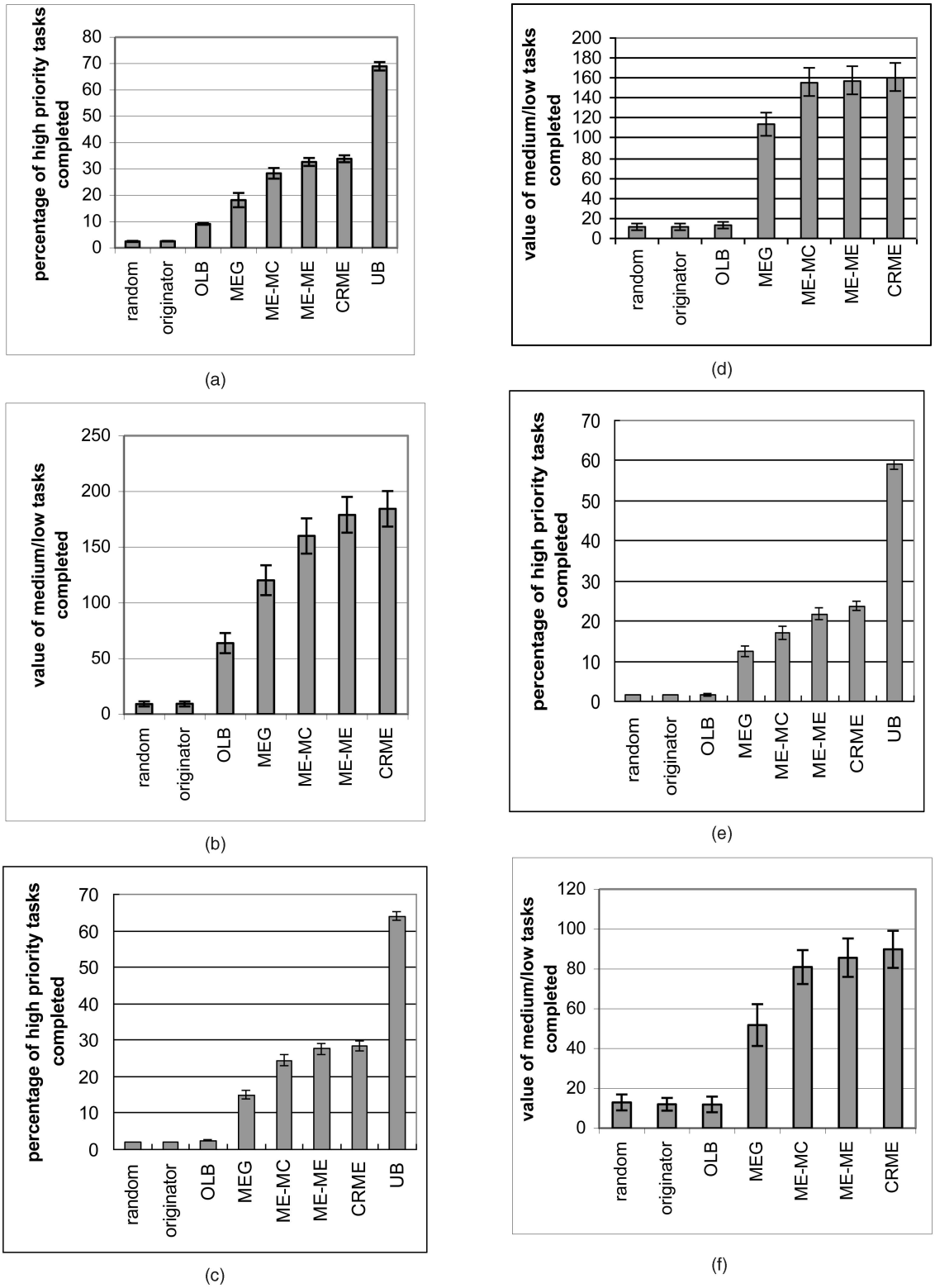

Fig. 3. The simulation results using the mean execution time of 600 seconds and mean intertask arrival of 10 seconds for (a) and (b), 8 seconds for (c) and (d), and 6 seconds for (e) and (f): (a), (c), and (e) show the percentage of high-priority tasks completed, and (b), (d), and (f) show the value of medium and low-priority tasks completed.

input or output, it is assumed that only one communication is allowed at a time. If multiple tasks need input data at this moment in time, only one task at a time may receive its input data (no broadcasting only point-to-point transfer). For simulation purposes, the size of the input data was calculated using 10 Kbits as the mean and a COV of 0.7 with the maximum size of 1 Mbit. The size of the result (output) was calculated using 10 Kbits as the mean and a COV of 0.7 with the maximum size being 10 Mbits. A task may receive input from all other devices and from one outside source

(e.g., a weather station for forecast reports). The maximum total number of inputs a task may need would be eight. The average number of input sources was 2.5 (the number of input sources was calculated using a normal distribution with mean 2.5 and minimum of zero and maximum of eight sources).

In a real system, the hard deadline of a task may be set by the user that requested the task, by the task designer, or the system operator/administrator. This research assumes that when the task arrives, the deadline of the task is given. For our simulation studies, the deadline of task i was equal to its the arrival time plus the overall mean execution time of all tasks plus two times the median execution time of task i on all devices plus the expected communication time (input and result) plus the expected communication wait time (= the mean number of input receptions (2.5) multiplied by seven multiplied by the mean input communication time plus seven multiplied by the mean result communication time).

6 Results

The simulation results for the different mean execution times and mean intertask arrival times are shown. For the random, originator, and OLB heuristics, two different DVS usage were studied. One is to use the fastest speed level for the high-priority tasks while using the slowest speed level for the medium and low-priority tasks. Thus, the speed level used of any given device depends on the task priority. The other is to use the median speed level for all tasks. The median speed level of a device would be the (total number of levels of a device)/2. Therefore, if there are 16 discrete speed levels for a device starting from level one being the slowest, then the median speed level would be level eight. Preliminary tests show that the performance of heuristics using the first method is better than the heuristics using the second method. The first method is used for all figures.

Fig. 2 shows the performance of the heuristics when the mean task execution is 60 seconds. The 95 percent confidence interval of the performance is shown in these figures. Because the confidence intervals of ME-ME and CRME heuristics overlap, these two heuristics are considered to perform comparably. The ME-MC heuristic was a close third. The average runtimes, in seconds per mapping event, of random, originator, OLB, MEG, ME-MC, ME-ME, and CRME are 0.00001, 0.00001, 0.00004, 0.00005, 0.0015, 0.28, and 0.34, respectively.

Fig. 4a shows the performance while increasing the mean task arrival rates (decreasing mean intertask arrival times). As the mean task arrival rates increases, the number of tasks in the system also increases and the percentage of high-priority tasks completed decreases. The average number of tasks per trial was 3,373, 4,185, and 5,688 for the mean intertask arrival time of 10, 8, and 6 seconds. This average includes tasks with mean execution time of 600 seconds.

Fig. 3 shows the results when the mean task execution time is increased to 600 seconds. Overall, the performance degraded. Because of the longer mean execution time, the tasks are more likely to be dropped. The 95 percent confidence interval of the performance is shown in these figures. Because the confidence intervals of ME-ME and CRME heuristics overlap, these two heuristics are considered to perform comparably. Fig. 4b shows the performance while increasing the mean task arrival rates (decreasing mean intertask arrival times). As the mean task arrival rates increases, number of tasks in the system increases and the percentage of high-priority tasks completed decreases.

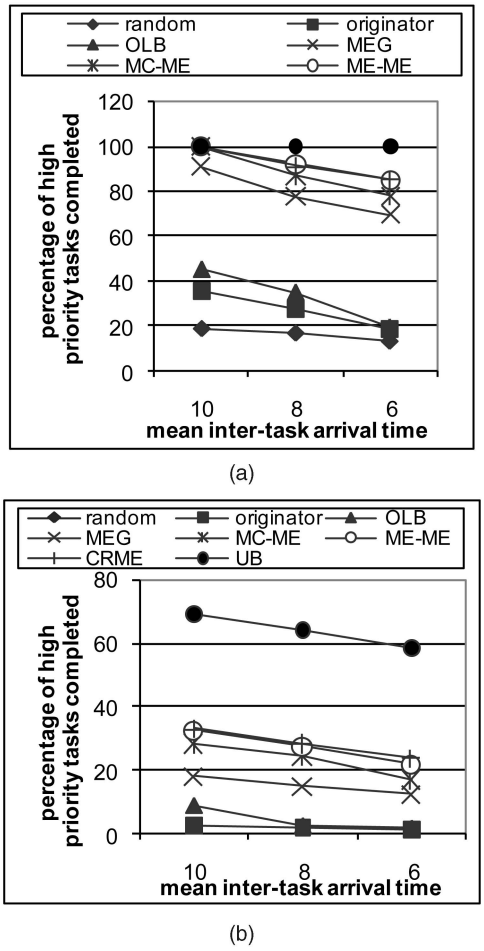

Fig. 4. The percentage of high-priority tasks completed is shown. The mean execution time of (a) 60 seconds and (b) 600 seconds and mean intertask arrival times of 10, 8, and 6 seconds are used. The results for random and originator are colocated in (b).

As it gets more difficult to complete high-priority tasks (as there are more tasks in the system due to increased task arrival rate or as mean task execution times are increased), the batch mode heuristics ME-ME and CRME perform better than the rest of the heuristics (shown in Fig. 4). While remapping, the batch mode heuristics (ME-ME and CRME) consider all mappable tasks in the system, and the order in which the tasks are mapped can be different from the previous mapping event. Therefore, the tasks can be assigned to another machine that is better suited, or they can be rescheduled. The ME-MC only considers the new task that arrived and once the task is mapped, it is not moved to another device nor rescheduled. Also, MC-ME can only increase the speed level for one device per mapping event.

The ME-MC, ME-ME, and CRME heuristics explicitly consider the high-priority tasks first (in the batch for ME-ME and CRME heuristics) to complete. The rest of the

TABLE 1
This Is a Table That Summaries the Heuristic Methods Used for This Research

name	mode	key idea	TC	perf.	best
random	I	random	1	bad	
originator	I	task requester completes own tasks	1	bad	
OLB	I	locate the first available device	O(M)	not good	
MEG	I	locate the device that uses the minimum energy	O(M)	okay	
ME-MC	I	switch between two methods to locate the suitable device	O(M)	good	when there is a tighter constrain on time
ME-ME	B	two phase greedy	$O(N^2)$	good	as more tasks are in the system
CRME	B	determine the task that will suffer most if not given the preference	$O(N^2)$	good	as more tasks are in the system

The mode, key idea, time complexity (TC), overall performance (perf.), and when a method performs the best are briefly described. The I and B for the mode column is immediate mode and batch mode, respectively. The M and N for the time complexity calculation is the number of machines and number of tasks, respectively.

heuristics run the high-priority tasks using the fastest speed level, giving the high-priority tasks a higher chance of completing. A table that compares the heuristics described in this research is shown in Table 1.

7 Conclusions

An ad hoc grid HC environment was modeled and simulated. Seven dynamic heuristics were designed, developed, and evaluated using the HC environment. The environment includes randomly arriving tasks with priorities and a deadline and devices with limited battery capacity that use DVS for power management. In this scenario, a resource manager needs to exploit the heterogeneity of the tasks and resources while managing the energy. The primary goal of this study was to complete as many high-priority tasks as possible, under the constraint of available system energy, during a given interval of time. The secondary goal was to complete as many medium and low-priority tasks as possible to maximize the sum of the weighted priorities of medium and low-priority tasks completed by their deadlines with the same constraints as the primary goal. A mathematical UB was derived.

The batch mode ME-ME and CRME heuristics were the best, and they performed comparably. However, they required significantly more time than the other heuristics. In cases where the mean task execution times are short, the immediate mode ME-MC heuristic may be preferable because it is very fast and can perform nearly comparable to the two best heuristics.

There can many possible directions for future research based on this study. A multihop ad hoc network or a wireless cell network may be used. A more detailed communication scheme and other communication issues may be introduced, such as security and compression/decompression methods. With asynchronous battery recharging as opposed to the synchronous recharging model described in this paper (which assumes all team members return together), the problem of completing tasks while efficiently using the system energy will be the same, but the complexity of the heuristics will increase, and the metric will need to be adapted for individual dynamic changes of available battery energy. Another aspect of future work would be to include the option of decreasing the speed of a processor at a later time. In addition, we can consider relaxing the assumption that the RMS is executed on a device with unlimited energy.

In summary, we have presented various power aware resource allocation heuristics that could be used in disaster situations such as wildfire fighting. There are many interesting future directions that can be pursued building upon this research.

Acknowledgments

The authors thank Sameer Shivle, Prasanna Sugavanam, and T.N. Vijaykumar for their valuable comments. A preliminary version of portions of this material was presented at the 19th International Parallel and Distributed Processing Symposium. This research was supported by the US National Science Foundation under Grant CNS-0615170, the Colorado State University George T. Abell Endowment, and the Korea University Grant. Submitted to the *IEEE TPDS* Special Section on Power-Aware Parallel and Distributed Systems in October 2007.

References

[1] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, and B. Yao, "Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems," *Advances in Computers Volume 63: Parallel, Distributed, and Pervasive Computing*, A.R. Hurson, ed., pp. 91-128, 2005.

[2] S. Ali, A.A. Maciejewski, H.J. Siegel, and J.-K. Kim, "Measuring the Robustness of a Resource Allocation," *IEEE Trans. Parallel and Distributed Systems*, vol. 15, no. 7, pp. 630-641, July 2004.

[3] S. Ali, A.A. Maciejewski, and H.J. Siegel, "Perspectives on Robust Resource Allocation for Heterogeneous Parallel Systems," *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, eds., pp. 41-1-41-30, Chapman & Hall/CRC Press, 2008.

[4] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems," *Tamkang J. Science and Eng.*, special 50th anniversary issue (invited), vol. 3, no. 3, pp. 195-207, Nov. 2000.

[5] *ARM Processor*, http://www.arm.com, July 2007.

[6] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks," *IEEE Trans. Computers*, vol. 53, no. 5, pp. 584-600, May 2004.

[7] H. Barada, S.M. Sait, and N. Baig, "Task Matching and Scheduling in Heterogeneous Systems Using Simulated Evolution," *Proc. 10th IEEE Heterogeneous Computing Workshop (HCW '01) and Proc. 15th Int'l Parallel and Distributed Processing Symp. (IPDPS '01),* Apr. 2001.

[8] I. Banicescu and V. Velusamy, "Performance of Scheduling Scientific Applications with Adaptive Weighted Factoring," *Proc. 10th IEEE Heterogeneous Computing Workshop (HCW '01),* and *Proc. 15th Int'l Parallel and Distributed Processing Symp. (IPDPS '01),* Apr. 2001.

[9] T.D. Braun, H.J. Siegel, and A.A. Maciejewski, "Heterogeneous Computing: Goals, Methods, and Open Problems," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '01),* invited keynote paper, pp. 1-12, June 2001.

[10] T.D. Braun, H.J. Siegel, N. Beck, L. Boloni, R.F. Freund, D. Hensgen, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, and B. Yao, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing,* vol. 61, no. 6, pp. 810-837, June 2001.

[11] *Crusoe/Efficeon Processor,* http://www.transmeta.com, July 2007.

[12] *Computer and Job-Shop Scheduling Theory,* E.G. Coffman Jr. ed., John Wiley & Sons, 1976.

[13] R.P. Dick and N.K. Jha, "MOCSYN: Multiobjective Core-Based Single-Chip System Synthesis," *Proc. Design Automation and Test in Europe Conf. (DATE '99),* pp. 263-270, Mar. 1999.

[14] *Heterogeneous Computing,* M.M. Eshaghian, ed. Artech House, 1996.

[15] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Trans. Software Eng.,* vol. SE-15, no. 11, pp. 1427-1436, Nov. 1989.

[16] *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster and C. Kesselman, eds. Morgan Kaufmann, 1999.

[17] R.F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.D. Lima, F. Mirabile, L. Moore, B. Rust, and H.J. Siegel, "Scheduling Resources in Multiuser, Heterogeneous, Computing Environments with SmartNet," *Proc. Seventh IEEE Heterogeneous Computing Workshop (HCW '98),* pp. 184-199, Mar. 1998.

[18] R.F. Freund and H.J. Siegel, "Heterogeneous Processing," *IEEE Computer,* vol. 26, no. 6, pp. 13-17, June 1993.

[19] A. Ghafoor and J. Yang, "A Distributed Heterogeneous Supercomputing Management System," *IEEE Computer,* vol. 26, no. 6, pp. 78-86, June 1993.

[20] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M.B. Srivastava, "Power Optimization of Variable-Voltage Core-Based Systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 18, no. 12, pp. 1702-1714, Dec. 1999.

[21] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava, "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors," *Proc. 19th IEEE Real-Time Systems Symp. (RTSS '98),* pp. 95-105, Dec. 1998.

[22] O.H. Ibarra and C.E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Non-Identical Processors," *J. ACM '77,* vol. 24, no. 2, pp. 280-289, Apr. 1977.

[23] M. Kafil and I. Ahmad, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency,* vol. 6, no. 3, pp. 42-51, July 1998.

[24] A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C. Wang, "Heterogeneous Computing: Challenges and Opportunities," *Computer,* vol. 26, no. 6, pp. 18-27, June 1993.

[25] J.-K. Kim, D.A. Hensgen, T. Kidd, H.J. Siegel, D.St. John, C. Irvine, T. Levin, N.W. Porter, V.K. Prasanna, and R.F. Freund, "A Flexible Multi-Dimensional QoS Performance Measure Framework for Distributed Heterogeneous Systems," *Cluster Computing,* special issue on cluster computing in science and engineering, vol. 9, no. 3, pp. 281-296, July 2006.

[26] J.-K. Kim, S. Shivle, H.J. Siegel, A.A. Maciejewski, T.D. Braun, M. Schneider, S. Tideman, R. Chitta, R.B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S.S. Yellampalli, "Dynamically Mapping Tasks with Priorities and Multiple Deadlines in a Heterogeneous Environment," *J. Parallel and Distributed Computing,* vol. 67, no. 2, pp. 154-169, Feb. 2007.

[27] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic Task Mapping Algorithms for a Distributed Heterogeneous Computing Environment," *Proc. Fourth IEEE Heterogeneous Computing Workshop (HCW '95),* pp. 30-34, Apr. 1995.

[28] J. Luo and N.K. Jha, "Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-Time Embedded Systems," *Proc. Int'l Conf. Computer-Aided Design (ICCAD '00),* pp. 357-364, Nov. 2000.

[29] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R.F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *J. Parallel and Distributed Computing,* vol. 59, no. 2, pp. 107-121, Nov. 1999.

[30] M. Maheswaran, T.D. Braun, and H.J. Siegel, "Heterogeneous Distributed Computing," *Encyclopedia of Electrical and Electronics Eng.,* vol. 8, J.G. Webster, ed., pp. 679-690, John Wiley & Sons, 1999.

[31] D. Marinescu, G. Marinescu, Y. Ji, L. Boloni, and H.J. Siegel, "Ad Hoc Grids: Communication and Computing in a Power Constrained Environment," *Proc. Workshop Energy-Efficient Wireless Comm. and Networks (EWCN '03) and Proc. 22nd Int'l Performance, Computing, and Comm. Conf. (IPCCC '03),* Apr. 2003.

[32] P. Mejia-Alvarez, E. Levner, and D. Mosse, "Power-Optimized Scheduling Server for Real-Time Tasks," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '02),* pp. 239-250, Sept. 2002.

[33] Z. Michalewicz and D.B. Fogel, *How to Solve It: Modern Heuristics.* Springer, 2000.

[34] R. Mishra, N. Rastogi, Z. Dakai, D. Mosse, and R. Melhem, "Energy Aware Scheduling for Distributed Real-Time Systems," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS '03),* Apr. 2003.

[35] S. Shivle, R. Castain, H.J. Siegel, A.A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco, "Static Allocation of Resources to Communicating Subtasks in a Heterogeneous Ad Hoc Grid Environment," *J. Parallel and Distributed Computing,* special issue on algorithms for wireless and ad hoc networks, vol. 66, no. 4, pp. 600-611, Apr. 2006.

[36] S. Shivle, H.J. Siegel, A.A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco, "Mapping Subtasks with Multiple Versions on an Ad Hoc Grid," *Parallel Computing,* special issue on heterogeneous computing, vol. 31, no. 7, pp. 671-690, July 2005.

[37] H. Singh and A. Youssef, "Mapping and Scheduling Heterogeneous Task Graphs Using Genetic Algorithms," *Proc. Fifth IEEE Heterogeneous Computing Workshop (HCW '96),* pp. 86-97, 1996.

[38] L. Shang, R.P. Dick, and N.K. Jha, "DESP: A Distributed Economics-Based Subcontracting Protocol for Computation Distribution in Power-Aware Mobile Ad Hoc Networks," *IEEE Trans. Mobile Computing,* vol. 3, no. 1, pp. 33-45, Jan.-Mar. 2004.

[39] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," *Proc. Usenix Symp. Operating Systems Design and Implementation (OSDI '94),* pp. 13-23, Nov. 1994.

[40] M.-Y. Wu, W. Shu, and H. Zhang, "Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems," *Proc. Ninth IEEE Heterogeneous Computing Workshop (HCW '00),* pp. 375-385, May 2000.

[41] D. Xu, K. Nahrstedt, and D. Wichadakul, "QoS and Contention-aware Multi-Resource Reservation," *Cluster Computing,* vol. 4, no. 2, pp. 95-107, Apr. 2001.

[42] J. Yang, I. Ahmad, and A. Ghafoor, "Estimation of Execution Times on Heterogeneous Supercomputer Architectures," *Proc. Int'l Conf. Parallel Processing (ICPP '93),* pp. I-219-I-226, Aug. 1993.

[43] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," *Proc. 36th Ann. Symp. Foundations of Computer Science (FOCS '95),* pp. 374-382, 1995.

[44] V. Yarmolenko, J. Duato, D.K. Panda, and P. Sadayappan, "Characterization and Enhancement of Dynamic Mapping Heuristics for Heterogeneous Systems," *Proc. Int'l Workshop Parallel Processing (ICPP '00),* pp. 437-444, Aug. 2000.

[45] Y. Yu and V.K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," *ACM/Kluwer J. Mobile Networks and Applications,* special issue on algorithmic solutions for wireless, mobile, ad hoc and sensor networks, vol. 10, no. 1, pp. 115-131, Feb. 2005.

[46] D. Zhu, R. Melhem, and B.R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 7, pp. 686-700, July 2003.

Jong-Kook Kim received the BS degree in electronic engineering from Korea University, Seoul, in 1998 and the MS and PhD degrees in electrical and computer engineering from Pur due University in May 2000 and August 2004, respectively. He is currently an assistant professor in the School of Electrical Engineering, Korea University, Seoul. He has worked at Samsung SDS's IT R & D Center from 2005 to 2007. His research interests include heterogeneous distributed computing, computer architecture, performance measures, resource management, evolutionary heuristics, energy-aware computing, and efficient computing. He is a member of the IEEE, the IEEE Computer Society, and the ACM. His complete vitae is available at http://jongkook.kim.googlepages.com.

Howard Jay Siegel received the BS degree in electrical engineering and the BS degree in management from the Massachusetts Institute of Technology (MIT), and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science, Princeton University. He is the George T. Abell endowed chair distinguished professor of electrical and computer engineering in the Department of Electrical and Computer Engineering and a professor of computer science in the Department of Computer Science, Colorado State University (CSU). He is the director of the CSU Information Science and Technology Center (ISTeC). ISTeC is a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. He is a fellow of the IEEE and the ACM. From 1976 to 2001, he was a professor in the School of Electrical and Computer Engineering, Purdue University. He is a coauthor of more than 350 technical papers. His research interests include heterogeneous parallel and distributed computing, parallel algorithms, and parallel machine interconnection networks. He was a co-editor-in-chief of the *Journal of Parallel and Distributed Computing* and was on the editorial boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was the program chair/cochair of three conferences, general chair/cochair of seven conferences, and chair/cochair of five workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sciences honor society. For more information, visit http://www.engr.colostate.edu/~hj.

Anthony A. Maciejewski received the BSEE, MS, and PhD degrees from Ohio State University in 1982, 1984, and 1987, respectively. From 1988 to 2001, he was a professor of electrical and computer engineering in the Department of Electrical and Computer Engineering, Purdue University, West Lafayette. He is currently the department head of the Electrical and Computer Engineering Department, Colorado State University. He is a fellow of the IEEE. His complete vitae is available at http://www.engr.colostate.edu/~aam.

Rudolf Eigenmann received the PhD degree in electrical engineering/computer science from ETH Zurich, Switzerland, in 1988. He is a professor in the School of Electrical and Computer Engineering, Purdue University. He is also the interim director of the Computing Research Institute and associate director of Purdue's Cyber Center. His research interests include optimizing compilers, programming methodologies and tools, performance evaluation for high-performance computers, and Internet sharing technology. From 1988 to 1995, he worked as a research scientist at the Center for Supercomputing Research and Development, University of Illinois, Urbana Champaign, where he also served as the leader of the Center's Cedar Fortran compiler group. He has published his work in more than 100 papers in international conference and workshop proceedings and journals. He serves on the editorial boards of the *International Journal of Parallel Programming*, the *IEEE Transaction on Parallel and Distributed Systems Journal*, and the *IEEE Computing in Science and Engineering Magazine*. He has served as the chairman of computer engineering at Purdue's School of ECE and as the chairman of the High-Performance Group, Standard Performance Evaluation Corp. (SPEC). He has also been the general chair and program chair of such conferences as the ACM Symposium Principles and Practice of Parallel Programming, the International Conference on Parallel Processing, the Workshop on Languages and Compilers for High-Performance Computing, and the Workshop on High-Level Interfaces for Parallel Systems. He is the recipient of a 1997 US National Science Foundation CAREER Award. For more information, please visit http://www.ece.purdue.edu/~eigenman. He is a senior member of the IEEE.