Simulation Modelling Practice and Theory 40 (2014) 95–111

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

# An energy-efficient process clustering assignment algorithm for distributed system

Anan Niyom *, Peraphon Sophatsathit, Chidchanok Lursinsap

*Advanced Virtual and Intelligent Computing Research Center, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Thailand*

ARTICLE INFO

*Article history:*
Received 1 July 2013
Received in revised form 23 August 2013
Accepted 11 September 2013
Available online 8 October 2013

*Keywords:*
Distributed assignment algorithm
Dependent task graph
Processing unit
Task scheduling
Energy aware

ABSTRACT

This paper proposes a distributed assignment algorithm for selecting the optimal energy consumption during process execution, idling, and transmission in a distributed system. Selection criteria are based on identifying candidate processing units that are suitable for minimizing idle energy in task scheduling. The proposed algorithm tries to mimic as close to real situation as possible by assuming that each processing unit has multiple capabilities to execute different tasks with different characteristics. Task scheduling can be flexibly carried out to attain optimal energy consumption without any restrictions as those of comparative algorithms. Thus, the energy required by each processing unit varies considerably depending on the schedule. Experimental results show that the proposed algorithm yields the lowest idle, total energy consumption, and satisfactory execution energy. The extraneous transmission energy is a trade-off for scheduling flexibility.



## 1. Introduction

Demands for energy conscious algorithms have increased in the recent years along with the fast-paced development of computational technology [1–4]. Various combined system architectures or operators were used to solve complicated problems in science and commercial including cloud computing and computer grid systems [5–9]. These systems were built with complex algorithms which required high computing power. Thus, one mandate that the supporting algorithms had to satisfy was minimizing the complexity of the computation process.

In general, when parallel code is pending for execution by a cluster of servers in a distributed system, the tasks, data, and functions are generated by the user on the client site. Only the required components are sent to the server cluster for execution. In most cases, each assigned server will only receive allotted jobs to perform. If the jobs coming from multiple sources must undergo additional security measures, a higher transferring rate to send data across networks will be required due to more processes to be finished. Therefore, ordinary server assignment algorithms are inadequate as the time complexity posts more power requirement to achieve the optimal energy consumption.

Concern of energy-aware issue calls for energy-efficient server assignment problem to be carefully studied [10–14]. Two minimum energy consumption techniques for real-time 2-level heterogeneous grid system were proposed by Terzopoulos and Karatza [15], i.e., Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM). The predominant aspect is the reduction of energy consumption with minimum performance degradation. In a typical distributed system, some data or functions are kept at a local scheduler and some are kept in specific places due to required security. The task scheduler

* Corresponding author. Address: Advanced Virtual and Intelligent Computing Research Center, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, 254 Phayathai Road, Pathumwan, Bangkok 10330, Thailand. Tel.: +66 877127360.
*E-mail addresses:* anan.niyom@gmail.com (A. Niyom), speraphon@gmail.com (P. Sophatsathit), lchidcha@gmail.com (C. Lursinsap).

1569-190X/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2013.09.005

must determine the starting time of each task and the server to which the task should be sent. To attain the optimal energy consumption, scheduling scenarios could be quite complex due to the flexibility of assigning task to proper matching execution unit. Our proposed Energy-Efficient Process Clustering Assignment (EPC) algorithm aims to handle such complex scenarios to find the optimal energy efficient assignment in a reasonable computing time. The important aspects to the development of this algorithm are matching the servers to which the tasks should be sent, determining the execution time of each specific task to be run on the designated server, and clustering tasks for execution on the same server.

The paper is organized as follows. Related works and their background theory are briefly reviewed in Section 2. Details of the problem formulation and constraints are described in Section 3. The proposed algorithm is discussed in Section 4. Sections 5 presents the experimental details and results. Some notable points are discussed and concluded in Section 6.

## 2. Related works and background

Many parallel or distributed task scheduling algorithms have been proposed, such as list scheduling, clustering, and task duplication heuristics. A brief summary of each approach is described below.

### 2.1. List scheduling heuristics (LSH)

This is perhaps the most common scheduling algorithm owing to its practicality [16–18]. Most list scheduling algorithms have two parts: task prioritization and server selection. In the first part, the algorithm assigns a score to each task according to its average computation cost. The score will then be ranked in a descending order to determine task priority. In the second part, the algorithm estimates the execution time of each task used by individual server and assigns the task to the server which has the least execution time for that task. One of the interesting algorithms is the Min-Min algorithm proposed by Li et al. in 2011 [18]. For this algorithm, the authors modified the original Min-Min algorithm which treated each task graph as independent to treating each task graph as dependent. The algorithm finds the earliest finishing time for each task in each device and then schedules each task to be computed with the earliest finishing time.

### 2.2. Clustering heuristics (CH)

Heuristic clustering algorithms are algorithms aiming to group tasks into $m$ groups according to their transmission edge, to reduce transmission cost, and then to map each group to $n$ servers [19–22]. One of the examples of heuristic clustering algorithms is the algorithm introduced by Liou and Palis in 1997 [20]. The algorithm has four steps: clustering all tasks into groups, merging clusters to a number of servers, mapping individual cluster to a server, and determining the order of task execution. It balances both the load in each cluster or server and the communication traffic among the clusters. A cluster of tasks is assigned to a server starting from the cluster which has the highest communication cost until all clusters are assigned.

### 2.3. Task duplication heuristics (TDH)

This algorithm pushes redundant tasks to different servers to reduce the communication cost between tasks [23,24]. Although it can reduce the computing time, it requires high energy consumption and execution cost as redundant tasks are executed on different servers. This approach is effective for systems with a high ratio of communication and execution cost. Unfortunately, its operation is very complex and the overhead to find an optimal solution is high.

### 2.4. Heterogeneous earliest finish time (HEFT)

The algorithm was proposed by Topcuoglu et al. in 2002 [25]. It finds the shortest time of overall finishing time of a system with insertion base. The algorithm consists of two steps, namely, task priority and server selection. The task priority calculates the priority of computation and communication for each task. The server selection determines idle time slots on each server to insert a task for execution at the earliest time.

Some predominant aspects of the algorithms can be summarized as follows. List scheduling heuristic yields the lowest computational cost, while clustering heuristic has a moderate communication cost. At any rate, HEFT gives a shorter finishing time at relatively low cost. Task duplication heuristics are strong in short finishing time but use high energy. By exploiting such strengths and weaknesses, we have derived an energy saving approach by grouping tasks in groups to be executed on the same server that requires the least energy. This in turn reduces the overall communication energy considerably. However, some tasks may carry their inherent restrictions which preclude them from being clustered or executed on the same server. Hence, the details of problem formulation and constraints of task characteristics and server competency are discussed below.

## 3. Problems formulation and constraints

Formulation of the proposed algorithm and its constraints are given in the following sections.

### 3.1. Problem formulation

Let $G = (V, E)$ be a directed graph representing a process captured in the forms of tasks and their dependencies. Graph $G$ is called dependent task graph. $V = \{v_1, \ldots, v_n\}$ is a set of tasks to be performed. $E = \{(v_i, v_k) | v_i, v_k \in V\}$ is a set dependency edges among the tasks. An edge $(v_i, v_k)$ denotes that task $v_i$ must be performed prior to task $v_k$. A set of tasks occurs on the client site, which may be a mobile phone or a PC. Each task can be performed either on a client site or a server site. Each server is capable of performing more than one task with different amount of energy consumption. A task is processed by a processing unit which can be either a client machine or a server machine. Thus, instead of distinguishing a client from a server, we will call them *processing unit*. Each processing unit is capable of processing some tasks. It is possible that there may be more than one processing unit that can process the same task. Let $e_a(v_i)$ be the amount of energy consumed to process task $v_i$ by processing unit $a$. Assuming that the number of tasks is greater than the number of processing units, the problem studied in this paper is formulated as follows.

Given a dependent task graph $G$, find a method to partition tasks $v_i$'s $\in V$ into $m$ groups of tasks $\{\mathbf{g}_1, \ldots, \mathbf{g}_m\}$ so that group $\mathbf{g}_j$ can be performed on processing unit $j$ in such a way that the total amount of energy consumed by all processing units is minimum. In other words, the objective is to minimize $\sum_{1 \leqslant i \leqslant m; v_j \in \mathbf{g}_i} e_i(v_j)$.

### 3.2. Constraints on energy consumption

The amount of energy $e_a(v_i)$ can be decomposed into three types, namely, execution energy, transmission energy, and idle energy. Execution energy depends primarily on server competency and task characteristics. Task $v_i$ requires $t_a(v_i)$ amount of time to be executed on processing unit $a$. Let $\alpha_a$ be the energy per unit time consumed by processing unit $a$ to execute a task. The execution energy to execute task $v_i$ on processing unit $a$ is computed by $\alpha_a t_a(v_i)$.

Transmission energy occurs when data are sent from one processing unit to other processing units according to their data dependencies. Let $d_a(v_i)$ be the amount of data of task $v_i$ sent from current processing unit $a$ to other processing units in the same dependency path of $G$. For each unit amount of data, the transmission energy consumed is set to $\lambda_a$. Therefore, the transmission energy is computed by $\lambda_a d_a(v_i)$.

The idle energy of processing unit $a$ in this study is based on the assumption that if processing unit $a$ does not process any task, then it is put into an idle state. Processing unit $a$ is activated when there are some other processing units transferring the dependent data to be processed by it. Suppose the remaining jobs of task $v_i$ are transferred to processing unit $a$. When processing unit $a$ is activated, it must spends $w_a(v_i)$ unit time to receive all related data of $v_i$ to be transferred before execution. Therefore, the idle energy must involve the energy consumed during the idle state and the energy wasted during the data transferring period. Let $\psi_a(v_i)$ be the time that processing unit $a$ is in the idle state before being activated by task $v_i$. The total idle time is equal to $\psi_a(v_i) + w_a(v_i)$. We assume that each idle unit time of processing unit $a$ consumes $\beta_a$ unit of energy. Hence, the total energy is equal to $\beta_a(\psi_a(v_i) + w_a(v_i))$.

Some processing unit may not consume any idle energy and transmission energy if it executes the last task of $G$ which is a leaf vertex. To handle this situation, constants $\kappa_a \in \{0, 1\}$ and $\mu_a \in \{0, 1\}$ are introduced to indicate whether a task $v_i$ executed by processing unit $a$ is a leaf vertex or an internal vertex in $G$. $\kappa_a = 0$ if the number of out-degrees of $v_i$ is equal to zero and $\mu_a = 0$ if the number of in-degrees of $v_i$ is equal to zero. Otherwise, it is equal to 1. Hence, the total energy of $G$, denoted by $\Gamma_G$, is computed by this equation.

$$\Gamma_G = \sum_{1 \leqslant a \leqslant m; v_j \in \mathbf{g}_a} e_a(v_j) + \sum_{1 \leqslant a \leqslant m; v_i \in \mathbf{g}_a} \beta_a \psi_a(v_i) \tag{1}$$

$$= \sum_{1 \leqslant a \leqslant m; v_j \in \mathbf{g}_a} \alpha_a t_a(v_j) + \kappa_a \lambda_a d_a(v_j) + \mu_a \beta_a w_a(v_j) + \sum_{1 \leqslant a \leqslant m; v_i \in \mathbf{g}_a} \beta_a \psi_a(v_i) \tag{2}$$

## 4. Energy-Efficient Process Clustering Assignment (EPC) algorithm

We assume that, for any processing unit $a$, the values of $t_a(v_i)$, $\alpha_a$, $d_a(v_i)$, $\lambda_a$, $w_a(v_i)$, and $\beta_a$ can be estimated prior to the assignment of tasks to processing units. This assumption is feasible and practical enough for this study. Furthermore, each processing unit in this study is assumed to have the capability to process different kinds of task unless a specific capability is defined to the processing unit. This means that, in general, more than one task can be assigned to any processing unit. Let $P_{v_i}$ be a set of processing units capable of processing task $v_i$. The task assignment consists of the following steps.

1. Identify a set of candidate processing units for each task $v_i$. A candidate processing unit for $v_i$ is a processing unit whose consumed energy is less than or equal to the minimum energy estimated from every processing unit in $P_{v_i}$. This candidate processing unit is called a *primary candidate* processing unit. Any processing units that are not satisfied with the above

**Table 1**
Power consumption of benchmark desktop CPUs at peak and idle states. The values were taken from [26,27] and were used to estimate energy consumption of each processing unit in the experiment.

| Processing unit | CPU | Clock (GHz) | Power consumption (W) | |
|---|---|---|---|---|
| | | | Peak | Idle |
| *a* | Intel Core i7-975 XE | 3.33 | 240 | 105 |
| *b* | Intel Core 2 Extreme QX6850 | 3.00 | 233 | 110 |
| *c* | Intel Core 2 Extreme QX6700 | 2.66 | 229 | 106 |
| *d* | Intel Core i7-920 | 2.66 | 224 | 105 |

condition will be classified as secondary or tertiary candidate processing units, depending on the following conditions. Processing unit $a$ is called a secondary candidate processing unit if $\alpha_a t_a(v_i) \leqslant \tau$, otherwise it is called tertiary candidate processing unit. Details are given in Algorithm 1.

2. Identify a set of actual candidate processing units. For any task $v_i$ and its candidate processing unit $a$, other dependent tasks of $v_i$ executable by $a$ are virtually assigned to $a$. The energy consumption of primary candidate and secondary candidate processing units are re-computed and compared. Any candidate processing unit $a$ is replaced by a secondary candidate processing unit $b$ if the energy consumption of $a$ is higher than that of $b$. Details are given in Algorithm 2.
3. Schedule the set of tasks assigned to each processing unit. The scheduling process consists of two phases. In the first phase, a set of dependent tasks assigned to a processing unit is orderly arranged according to their dependent paths in dependent task graph $G$ in the form of processing time duration. For those independent tasks, they can be arbitrarily scheduled. Therefore, the scheduled sequence in any processing unit will consist of alternate slots of task duration and slots of idle duration. The second phase is to minimize the idle energy of the whole system. If there are some tasks possibly executed by more than one processing unit, then these tasks are re-assigned to other processing units under the condition that the new total energy must not be increased. The detail of phase 1 is given in Algorithm 3 and that of phase 2 is given in Algorithm 4.

**Algorithm 1.** Identifying Preliminary Candidate Processing Units

**Require:** $v_i, P_{v_i}$, and $G$.
1. **for** $v_i \in G$ **do**
2. Let $o$ be the number of out-degrees of $v_i$.
3. $\tau = \min_{a \in P_{v_i}}(\alpha_a t_a(v_i)) + \max_{a \in P_{v_i}}(o\kappa_a \lambda_a d_a(v_i) + \mu_a \beta_a w_a(v_i))$
4. **for** each processing unit $a \in P_{v_i}$ **do**
5. **if** $\alpha_a t_a(v_i) \leqslant \tau$ **then**
6. mark $a$ as a secondary candidate processing unit.
7. **else**
8. mark $a$ as a tertiary candidate processing unit.
9. **end if**
10. **end for**
11. **end for**

**Algorithm 2.** Identifying Actual Candidate Processing Units

**Require:** $v_i, P_{v_i}, \mathbf{g}_a$, and $G$.
1. **for** $v_i \in G$ **do**
2. **for** each secondary candidate processing unit $a \in P_{v_i}$ **do**
3. Let $\mathbf{g}_a = \phi$ for all $a$.
4. **for** all ancestors $v_j$'s of $v_i$ **do**
5. **if** $v_j$ is executable by $a$ **then**
6. Let $\mathbf{g}_a = \mathbf{g}_a \cup \{v_j\}$.
7. **end if**
8. **end for**
9. **for** all descendants $v_k$'s of $v_i$ **do**
10. **if** $v_k$ is executable by $a$ **then**
11. Let $\mathbf{g}_a = \mathbf{g}_a \cup \{v_k\}$.

```
    end if
  end for
end for
Descendingly sort |g_a|.
Mark any processing unit a ∈ P_{v_i} having maximum |g_a| and lowest execution energy as a primary candidate processing unit.
Mark the rest of processing units in P_{v_i} as a secondary candidate processing units.
end for
for v_i ∈ G do
  for each secondary candidate processing unit b ∈ P_{v_i} do
    Compute energy e_b(v_i).
    if ∃ candidate processing unit a ∈ P_{v_i} such that e_b(v_i) < e_a(v_i) then
      Mark processing unit a as a tertiary candidate processing unit.
      Mark processing unit b as a primary candidate processing unit.
    end if
  end for
end for
```

**Algorithm 3.** Phase-1 Preliminary Task Scheduling in Processing Units

```
Require: v_i, P_{v_i}, and G.
Let v_f be the latest task in its primary processing unit in P_{v_f}.
Let ζ_{v_f} be the finishing time of task v_f.
Descendingly sorted task v_i at first level of G by execution time on its primary processing unit.
for each sorted task v_i at first level of G do
  if primary processing unit P_{v_i} is an empty slot then
    Assign v_i to its primary processing unit in P_{v_i}.
  else
    Assign v_i to its primary processing unit in P_{v_i} at time ζ_{v_f} + 1.
  end if
end for
while ∃ unassigned v_i do
  Let δ be a set of v_k having all ancestor tasks already assigned to their primary processing units.
  for each task v_k ∈ δ do
    Let A_{v_k} be a set of ancestor tasks of v_k already assigned to their primary processing units.
    Find a task v_j ∈ A_{v_k} having the latest finishing time at time ζ_{v_j}.
  end for
  Find a task v_k ∈ δ having earlier finishing time ζ_{v_j} and the shortest execution time on its primary processing unit.
  if time slot ζ_{v_j} + w_a(v_k) of primary processing unit in P_{v_k} is empty then
    Assign v_k to its primary processing unit in P_{v_k} at time slot ζ_{v_j} + w_a(v_k).
else
    Assign v_k to its primary processing unit in P_{v_k} at time slot ζ_{v_f} + 1.
  end if
end while
```

**Algorithm 4.** Phase-2 Minimizing Idle Energy in Task Scheduling

```
Require: a list of scheduled tasks in each g_a and G.
Let S be a set of |g_a| sorted in descending order.
for each corresponding g_a ∈ S do
  Let v_k be the last task in g_a.
  while task v_k is not the first task in g_a do
```

(*continued on next page*)

```
5.      Traverse the task list upward starting at v_k until an empty slot E is found and let v_j be the task next to this idle slot.
6.      Let A_{v_j} be a set of ancestors of v_j.
7.      for all tasks v_i ∈ A_{v_j} do
8.        if there exists a processing unit b with an empty slot and the data dependency of v_i and its ancestors in any processing units is not violated then
9.          Temporarily remove v_i from the present slot.
10.         Insert v_i to the beginning of this new empty slot.
11.         Reschedule all tasks whose starting times are after the ending time of v_i by using Algorithm 3.
12.         Let G' be the new dependent task graph after the temporary assignment of v_i.
13.         Compute the total energy Γ_{G'}.
14.         if Γ_{G'} > Γ_G then
15.           Remove v_i from processing unit b and assign it back to its original processing unit.
16.         else
17.           Permanently assign v_i to this new empty slot on processing unit b.
18.         end if
19.       end if
20.     end for
21.     if v_k is not the first task in g_a then
22.       Let v_k be a new task found after traversing list g_a upward starting from the empty slot E.
23.     end if
24.   end while
25. end for
```

To illustrate how these four algorithms work, the algorithms are applied to schedule all tasks in the dependent task graph shown in Fig. 1. Each vertex represents a task with its name. A parenthesis to the right of each vertex $v_i$ is a set of processing units capable of processing task $v_i$. For example, task $v_1$ can be processed by processing units $a$ and $c$. The values of all variables in Eq. (2) are hypothetically given in Tables 2–4. Table 2 provides the amount of processing energy per unit time ($\alpha_a$), amount of waiting energy per unit time ($\beta_a$), and transmission energy per unit amount of data ($\lambda_a$) of each processing unit $a$. In this example, there are four processing units which are $a$, $b$, $c$, and $d$. Table 3 defines the transmission speed of the link between each processing unit pair in unit amount of data per unit time. These constants are used to estimate data transmission time among processing units. Table 4 shows the amount of transmitted data of each task $v_i$ and its estimated transmission time of each processing unit. To show how the energy consumption is estimated, the activities of task $v_1$ will be used as an example.

The amount of time to process task $v_1$ on a processing unit $a$ is $t_a(v_1) = 260$ with $\alpha_a = 0.067$ as defined in Table 2. The estimated execution energy is equal to $\alpha_a t_a(v_1) = 0.067 \times 260 = 17.42$. The energy consumption of the other tasks can be similarly estimated. Table 5 summarizes the estimated energy consumption of all tasks in the given dependent task graph.

Suppose $v_1$ is processed at processing unit $a$. After finishing task $v_1$, the results must be transmitted to its descendant tasks which are $v_4$, $v_5$, $v_6$, and $v_7$. These four tasks can be executed by a set of processing units $(a, b)$ for $v_4$, $(c, d)$ for $v_5$, $(b, d)$ for $v_6$, and $(a, c)$ for $v_7$. Thus, the amount of transmission energy between processing unit pairs $\{a, b\}$, $\{a, c\}$, $\{a, d\}$ must

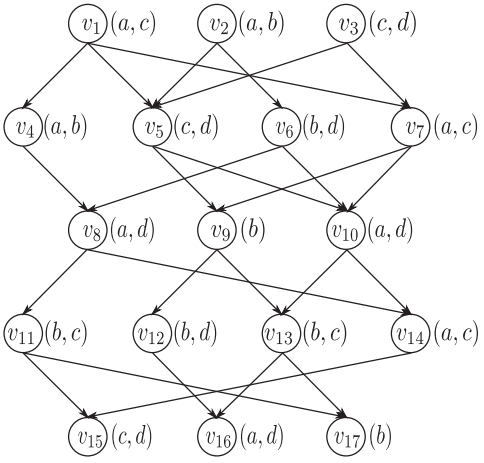


**Fig. 1.** An example of dependent task graph $G$. The parentheses to the right of each vertex denote candidate processing units to which the tasks can be executed. For example, vertex $v_1(a, c)$ denotes task $v_1$ that can be executed on either processing unit $a$ or $c$.

**Table 2**
A list of given energy consumption constants for each processing unit $i \in \{a, b, c, d\}$. $\alpha_i$ is the execution energy at peak state per unit time, $\beta_i$ is the waiting energy at idle state per unit time, $\lambda_i$ is the transmission energy per unit data.

| Energy constants | Processing units | | | |
|---|---|---|---|---|
| | $a$ | $b$ | $c$ | $d$ |
| $\alpha_{i\in\{a,\ b,\ c,\ d\}}$ | 0.067 | 0.065 | 0.064 | 0.062 |
| $\beta_{i\in\{a,\ b,\ c,\ d\}}$ | 0.029 | 0.031 | 0.029 | 0.029 |
| $\lambda_{i\in\{a,\ b,\ c,\ d\}}$ | 0.045 | 0.047 | 0.050 | 0.048 |

**Table 3**
Estimated data transmission rate $r_{a,\ b}$ between any processing unit pair $a$ and $b$ in unit amount of data per unit time.

| Processing unit | $a$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|
| $a$ | – | 1 | 0.5 | 1 |
| $b$ | 1 | – | 0.5 | 1.25 |
| $c$ | 0.5 | 0.5 | – | 1.25 |
| $d$ | 1 | 1.25 | 1.25 | – |

**Table 4**
The initial values of output data size and the estimated execution time of each task processed by each processing unit.

| Task | Amount of transmitted data | Amount of execution time | | | |
|---|---|---|---|---|---|
| $v_i$ | $d_a(v_i)$ | $t_a(v_i)$ | $t_b(v_i)$ | $t_c(v_i)$ | $t_d(v_i)$ |
| 1 | 60 | 260 | – | 760 | – |
| 2 | 80 | 350 | 160 | – | – |
| 3 | 10 | – | – | 890 | 820 |
| 4 | 60 | 420 | 150 | – | – |
| 5 | 90 | – | – | 940 | 430 |
| 6 | 30 | – | 400 | – | 560 |
| 7 | 50 | 670 | – | 850 | – |
| 8 | 40 | 300 | – | – | 630 |
| 9 | 50 | – | 470 | – | – |
| 10 | 30 | 920 | – | – | 950 |
| 11 | 80 | – | 550 | 120 | – |
| 12 | 10 | – | 200 | – | 130 |
| 13 | 90 | – | 100 | 790 | – |
| 14 | 40 | 940 | – | 580 | – |
| 15 | 50 | – | – | 350 | 310 |
| 16 | 40 | 930 | – | – | 750 |
| 17 | 70 | – | 370 | – | – |

**Table 5**
Estimated execution energy to execute each task on each processing unit for the first case.

| Task | Execution energy | | | |
|---|---|---|---|---|
| $v_i$ | $\alpha_a t_a(v_i)$ | $\alpha_b t_b(v_i)$ | $\alpha_c t_c(v_i)$ | $\alpha_d t_d(v_i)$ |
| 1 | 17.42 | – | 48.64 | – |
| 2 | 23.45 | 10.40 | – | – |
| 3 | – | – | 56.96 | 50.84 |
| 4 | 28.14 | 9.75 | – | – |
| 5 | – | – | 60.16 | 26.66 |
| 6 | – | 26.00 | – | 34.72 |
| 7 | 44.89 | – | 54.40 | – |
| 8 | 20.10 | – | – | 39.06 |
| 9 | – | 30.55 | – | – |
| 10 | 61.64 | – | – | 58.90 |
| 11 | – | 35.75 | 7.68 | – |
| 12 | – | 13.00 | – | 8.06 |
| 13 | – | 6.5 | 50.56 | – |
| 14 | 62.98 | – | 37.12 | – |
| 15 | – | – | 22.40 | 19.22 |
| 16 | 62.31 | – | – | 46.50 |
| 17 | – | 24.05 | – | – |

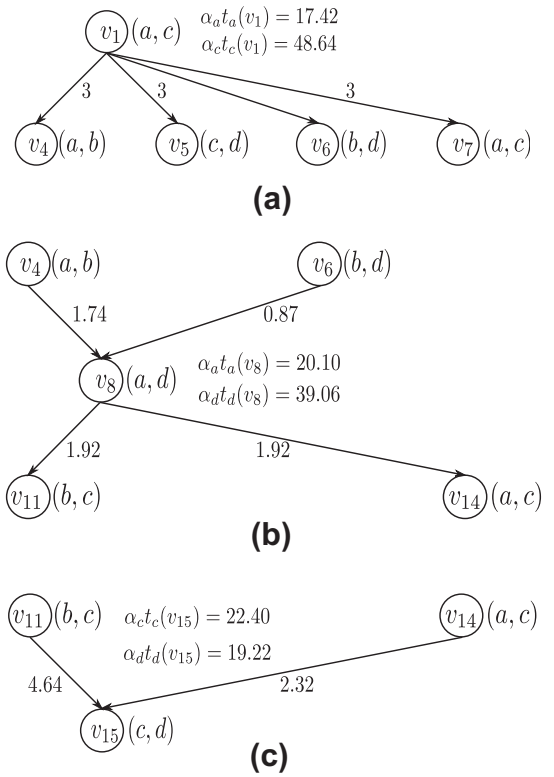


**Fig. 2.** Examples of applying the Algorithm 1 to identify preliminary candidate processing units for the tasks $v_1$, $v_8$, and $v_{15}$. The incoming edge of the considered task $v_i$ shows the waiting energy having maximum weight for each ancestor task. The outgoing edge shows the transmission energy having maximum weight for the transmission to each descendant task. The numbers next to the task denote the minimum execution energy utilized by the candidate processing units. (a) Task $v_1$, (b) Task $v_8$, and (c) Task $v_{15}$.

be estimated. Suppose we consider the pair $\{a, c\}$. From Tables 2 and 4, the amount of transmitted data is $d_a(v_1) = 60$ and the transmission energy per unit amount of data is $\lambda_a = 0.045$. Hence, the transmission energy is equal to $\lambda_a d_a(v_1) = 0.045 \times 60 = 2.7$. The transmission energy of other processing unit pairs can be similarly estimated.

The waiting energy of any processing unit depends upon the data transmission rate between the sending unit and the receiving unit. Let us consider tasks $v_6$ and $v_8$. Task $v_6$ can be processed by either processing units $b$ or $d$ and task $v_8$ can be processed by either processing units $a$ or $d$. Assume that $v_6$ is processed by $d$ and $v_8$ is by $a$. According to Table 4, the amount of data to be transmitted from $d$ to $a$ is equal to $d_d(v_6) = 30$ and the data transmission rate from $d$ to $a$ is equal to $r_{d,a} = 1$. Therefore, the data transmission time $w_a(v_8)$ is equal to $d_d(v_6)/r_{d,a} = 30/1 = 30$. The waiting energy at processing unit $a$ for $v_8$ is estimated by $\beta_a w_a(v_8) = 0.029 \times 30 = 0.87$.

The results produced by each algorithm will be explained next. From Algorithm 1, three different types of tasks will be considered to illustrate how Algorithm 1 works. The first type is a task without any incoming degree such as task $v_1$. The second type is a task with both incoming and outgoing degrees such as task $v_8$. The last type is a task having only incoming degree such as task $v_{15}$. For task $v_1$, the value of minimum energy $\tau$ estimated from every processing unit in list $P_{v_1}$ is equal to

$$\tau = \min_{k \in P_{v_1}}(\alpha_k t_k(v_1)) + \max_{k \in P_{v_1}}(o\kappa_k \lambda_k d_k(v_1) + \mu_k \beta_k w_k(v_1)) \tag{3}$$

$$= 17.42 + 3 \times 3 + 0 = 26.42 \tag{4}$$

Since $v_1$ has no incoming degree, the term $\mu_k \beta_k w_k(v_1)$ is set to zero. For task $v_8$, the value of minimum energy $\tau$ estimated from every processing unit in list $P_{v_8}$ is equal to

$$\tau = \min_{k \in P_{v_8}}(\alpha_k t_k(v_8)) + \max_{k \in P_{v_8}}(o\kappa_k \lambda_k d_k(v_8) + \mu_k \beta_k w_k(v_8)) \tag{5}$$

$$= 20.10 + 2 \times 1.92 + 1.74 = 25.68 \tag{6}$$

For task $v_{15}$, the value of minimum energy $\tau$ estimated from every processing unit in list $P_{v_{15}}$ is equal to

$$\tau = \min_{k \in P_{v_{15}}}(\alpha_k t_k(v_{15})) + \max_{k \in P_{v_{15}}}(o\kappa_k \lambda_k d_k(v_{15}) + \mu_k \beta_k w_k(v_{15})) \tag{7}$$

$$= 19.22 + 0 + 4.64 = 23.86 \tag{8}$$

From Fig. 2, processing unit $a$ is selected as a secondary candidate, while processing unit $c$ is selected as a tertiary candidate for $v_1$ with respect to $\tau$. Fig. 3 shows all secondary candidate processing units for each task after applying Algorithm 1.

In the second step, Algorithm 2 is used to identify the actual candidate processing units. The input is a dependent graph $G$ obtained from the first step as shown in Fig. 3. Algorithm 2 counts the number of ancestors as well as descendants of each task with respect to each secondary candidate processing unit. Any secondary processing unit of a task is set as a primary candidate processing unit if the task has the largest number of counts. Then the algorithm computes the actual energy consumption for each primary and secondary candidate processing units. An example is shown in Fig. 4 along with procedural computations below.

For task $v_{10}$ in Fig. 3, both $\mathbf{g}_a$ and $\mathbf{g}_d$ have the same counts of 2. As the execution energy of task $v_{10}$ on processing unit $d$ is less than the execution energy on processing unit $a$, the processing unit $d$ is marked as a primary candidate processing unit. Similarly, for task $v_{15}$ in Fig. 3, group $\mathbf{g}_c$ has the maximum counts of 3. Thus, the processing unit $c$ is marked as a primary candidate processing unit.

After identifying the primary candidate processing units, the algorithm determines the energy consumption of primary and secondary processing units of each task. For example, consider task $v_{10}$. This task has both incoming and outgoing degrees. The energy consumption of processing units $a$ and $d$ for task $v_{10}$ are shown in Fig. 4(a) and (b), respectively. The energy consumption of task $v_{10}$ by processing unit $a$ is equal to

$$e_a(v_{10}) = \alpha_a t_a(v_{10}) + \kappa_a \lambda_a d_a(v_{10}) + \mu_a \beta_a w_a(v_{10}) \tag{9}$$
$$= 61.64 + 2 \times 1.35 + 2.61 \tag{10}$$
$$= 66.95 \tag{11}$$

and by processing unit $d$ is equal to

$$e_d(v_{10}) = \alpha_d t_d(v_{10}) + \kappa_d \lambda_d d_d(v_{10}) + \mu_d \beta_d w_d(v_{10}) \tag{12}$$
$$= 58.90 + 2 \times 1.44 + 1.45 \tag{13}$$
$$= 63.23 \tag{14}$$

Since $e_d(v_{10})$ is less than $e_a(v_{10})$, the actually selected candidate processing unit for task $v_{10}$ is $d$. For a task without any outgoing degree such as $v_{15}$, a similar computation for energy consumption can be estimated. Task $v_{15}$ has two candidate processing units which are $c$ and $d$. Fig. 4(c) and 4(d) show the energy consumption of each processing unit for task $v_{15}$. The energy consumption of task $v_{15}$ by processing unit $c$ is equal to

$$e_c(v_{15}) = \alpha_c t_c(v_{15}) + \kappa_c \lambda_c d_c(v_{15}) + \mu_c \beta_c w_c(v_{15}) \tag{15}$$
$$= 22.40 + 0 + 0 \tag{16}$$
$$= 22.40 \tag{17}$$

and by processing unit $d$ is equal to

$$e_d(v_{15}) = \alpha_d t_d(v_{15}) + \kappa_d \lambda_d d_d(v_{15}) + \mu_d \beta_d w_d(v_{15}) \tag{18}$$
$$= 19.22 + 0 + 1.856 \tag{19}$$
$$= 21.076 \tag{20}$$

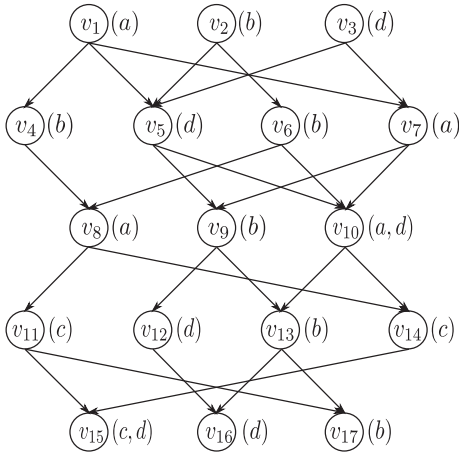


**Fig. 3.** Results from the first step after applying Algorithm 1 to the dependent task graph $G$ of Fig. 1. Note that some tasks, such as $v_{10}$ and $v_{15}$ still have more than one secondary candidate processing unit.

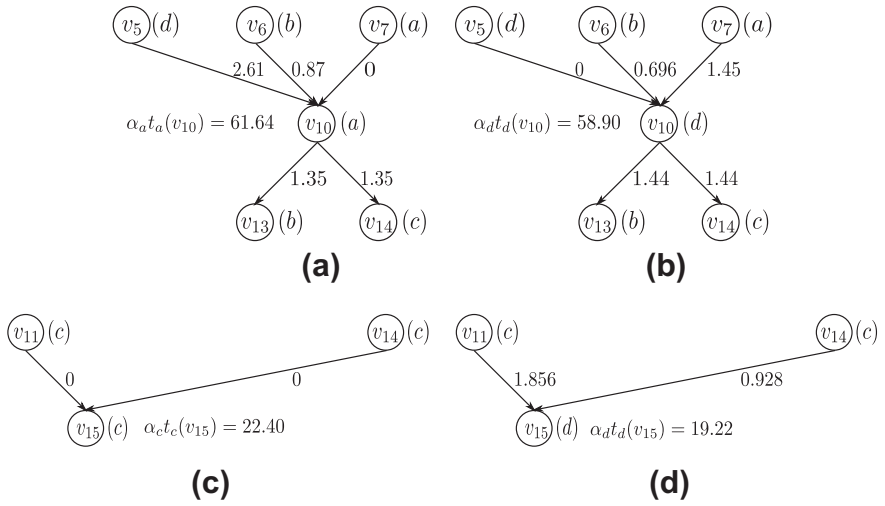


**Fig. 4.** Examples of candidate processing units for tasks $v_{10}$ and $v_{15}$. The tasks still contain secondary candidate processing units at the end of the first step and all the ancestors and descendants are primary candidate processing units. The incoming and outgoing edges are actual waiting and transmission energy, respectively. (a) Task $v_{10}$ on processing unit $a$, (b) Task $v_{10}$ on processing unit $d$, (c) Task $v_{15}$ on processing unit $c$, and (d) Task $v_{15}$ on processing unit $d$.

Because $e_d(v_{15})$ is less than $e_c(v_{15})$, the actually selected candidate processing unit for task $v_{15}$ is $d$. The actual processing unit of each task in the dependent task graph $G$ as the result of Algorithm 2 is shown in Fig. 5.

Scheduling is performed by Algorithm 3. For example, from Fig. 5, there are three tasks in the first level, namely, $v_1$, $v_2$ and $v_3$, with the assigned primary candidate processing units $a$, $b$, and $d$, having the execution time of 260, 160, and 820, respectively. Thus, task $v_2$ will be scheduled first, followed by $v_1$, and $v_3$. Next, the unassigned task $v_6$ whose ancestor task $v_2$ is considered. Task $v_6$ has the latest finishing time comparing with all of its ancestors, but has the earliest finishing time comparing with the ancestors of other tasks. Thus, it is assigned to primary candidate processing unit $b$. Task $v_4$, the descendant task of $v_1$, has the latest finishing time comparing with all of its ancestors, but has the earliest finishing time comparing with the ancestors of other tasks. Thus, it is assigned to primary candidate processing unit $b$. Task $v_8$, the descendant of $v_4$, is scheduled next. It has the latest finishing time comparing with all of its ancestors, but has the earliest finishing time comparing with the ancestors of other tasks. Therefore, it is assigned to primary candidate processing unit $a$. Tasks $v_5$ and $v_7$ have a common ancestor task $v_3$. Task $v_3$ has the latest finishing time comparing with all of its ancestors, but has the earliest finishing time comparing with the ancestors of other tasks. However, task $v_5$ has a shorter execution time. Hence, it is assigned to primary candidate processing unit $d$. The algorithms will continue until all tasks are assigned. Fig. 6(a) illustrates the list of scheduled tasks assigned by Algorithm 3.

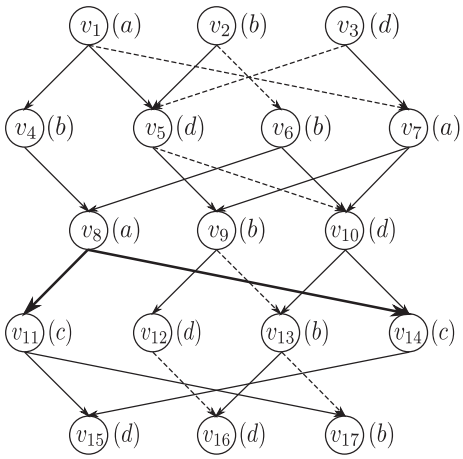


**Fig. 5.** Results of Algorithm 2. A solid line represents data transmission between different processing units. A dashed line represents data transmission within the same processing unit. A thick line represents data transmission from a task to its descendant tasks which are executed by the same processing unit, e.g., $v_8$ to $v_{11}$ and $v_{14}$.

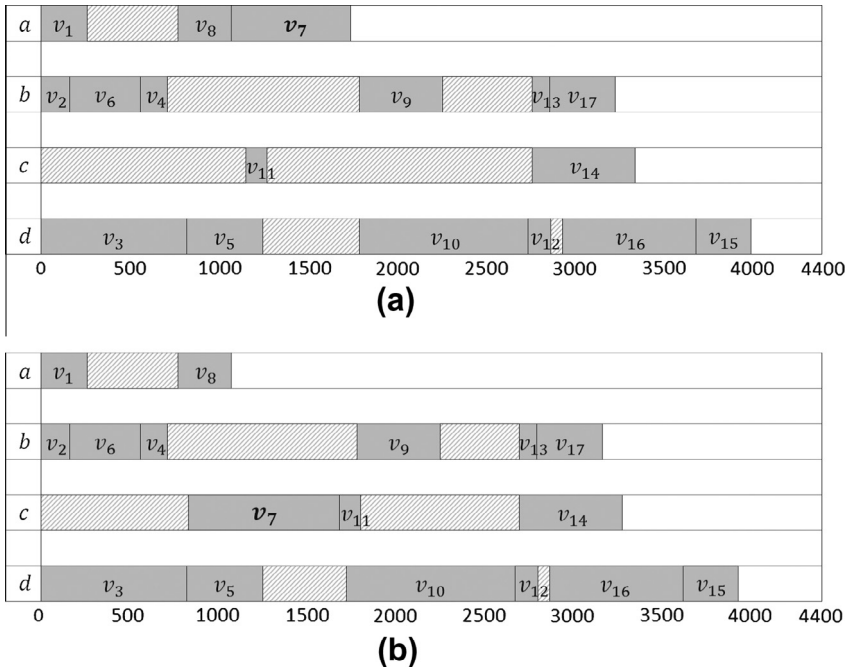


**Fig. 6.** The scheduling results of Algorithms 3 and 4. (a) The result of Algorithm 3, (b) The result of Algorithm 4. In this example, only task $v_7$ is removed from processing unit $a$ and re-assigned to processing unit $c$.

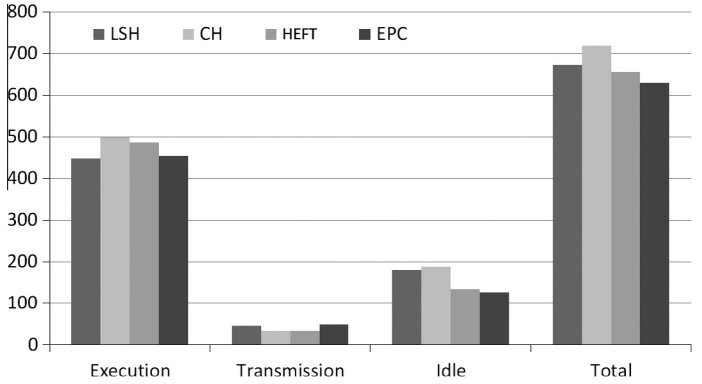


**Fig. 7.** Results of energy consumption for the first case.

Algorithm 4 is applied further to reduce the empty slots so as to minimize the idle energy of the scheduled tasks. From Fig. 6(a), the latest finished task is $v_{15}$ on processing unit $d$. The algorithm traverses the time slots of processing unit $d$ upwards until an empty slot is found. In this case, $v_{16}$ is the task next to this first idle slot and its ancestors are $v_{12}$ and $v_{13}$. A re-assignment trial of $v_{12}$ and $v_{13}$ to other processing units is proceeded. Although there are empty slots and no data dependency violation for $v_{12}$ and $v_{13}$, the re-assignment of $v_{12}$ and $v_{13}$ increases the amount of total energy. Therefore, no re-assignment for $v_{12}$ and $v_{13}$ is done. The next empty slot when traversing the time slot towards the beginning is a slot before $v_{10}$. Task $v_{10}$ has three ancestors which are $v_5$, $v_6$, and $v_7$. In this case, $v_7$ can be re-assigned to processing unit $c$ which decreases the idle energy and total energy. Other tasks in each processing unit are re-assigned in a similar fashion. Fig. 6(b) depicts the results from Algorithm 4.

## 5. Experimental results and discussion

We conducted four experimental cases on LSH, CH, HEFT, and EPC algorithms to test and evaluate their performance using different scheduling scenarios. The dependent task graph in the first case was synthesized. The second case was taken directly from [25] as a validation benchmark. The third case employed the same scenario as that of the second case but

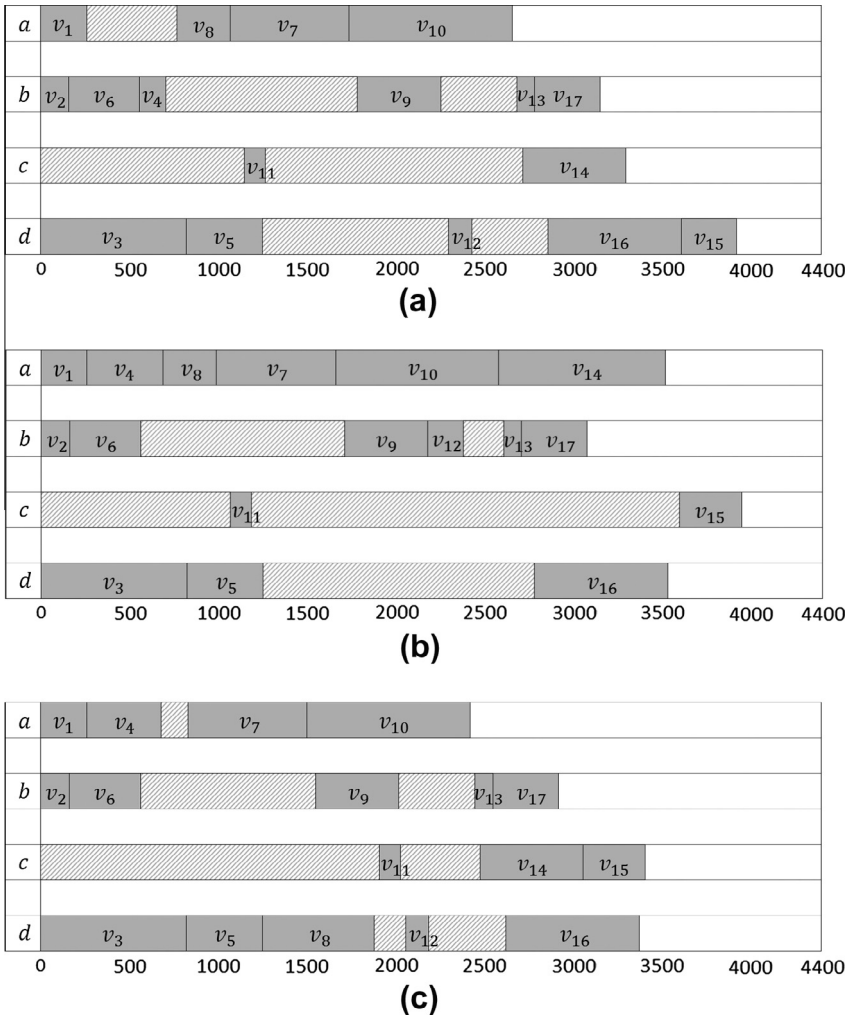


**Fig. 8.** A list of scheduled tasks in each processing unit of dependent task graph in Fig. 1. (a) List scheduling heuristic (LSH), (b) Clustering heuristic (CH), (c) Heterogeneous earliest finish time (HEFT).

**Table 6**
Energy consumption in each processing unit by LSH algorithm in the first case.

| Processing unit | Execution energy | Transmission energy | Idle energy | Total energy |
|---|---|---|---|---|
| *a* | 144.05 | 12.15 | 14.79 | 170.99 |
| *b* | 107.25 | 14.57 | 46.81 | 168.63 |
| *c* | 44.80 | 10.00 | 75.40 | 130.20 |
| *d* | 151.28 | 9.12 | 42.98 | 203.38 |
| Total | 447.38 | 45.84 | 179.98 | 673.20 |

imposed the processing unit limitation of the first case on the second case. The fourth case was extended from the first case to cover more complex task graphs and emphasize on execution intensive scenario rather than transmission load. Thus, consideration of the transmission time in the first and the fourth cases was not our main concern in this situation as the network capability is technologically increased.

The simulation environment for these experiments consists of four heterogeneous processing units. Each unit has unique competence but different energy consumption during peak or idle states. The amount of energy consumption for the four processing units were selected from the values of experimental energy consumption database for a number of CPU specifications [26,27]. The selected values are shown in Table 1. In this study, the energy consumption was converted to execution energy at peak state and waiting energy at idle state in watts per seconds. For example, for processing unit *a*, the execution

energy per unit time $\alpha_a$ = 240/3600 = 0.067 and the waiting energy per unit time $\beta_a$ = 105/3600 = 0.029. As processing units are connected to one another, the transmission energy ($\lambda$) were assigned to each pair. The values were randomly generated within the range of energy consumption during the peak state and the idle state. The obtained values for the execution energy per unit time, waiting energy per unit time, and transmission energy per unit data are shown in Table 2. In order to calculate delay time due to data transmission, the network transmission speed for all the processing unit pairs were assigned as shown in Table 3.

For the first case, the dependent task graph is shown in Fig. 1. There are 17 given tasks, i.e., $v_1$–$v_{17}$, and 4 processing units, i.e., *a*, *b*, *c*, and *d*. The tasks were created with different attributes and the processing units possessed various competency to handle individual task execution requirements. The initial assignment of processing units to their executable tasks are shown in Fig. 1. The size of output data and the estimated execution time of each task processed by each processing unit from the beginning of simulation are given in Table 4. The total execution energy consumed by each processing unit for each task is shown in Table 5.

Although other scheduling algorithms, namely, LSH, CH, HEFT, did not consider energy consumption as their cost functions, they are worthy to be deployed in performance comparison. Fig. 7 shows the comparison of energy consumption of LSH, CH, and HEFT scheduling algorithms with the proposed algorithm based on the first experimental case. This energy consumption was set up as a cost function during the scheduling process for each processing unit in terms of execution, transmission, and idle energy. The introduction of energy consumption was not deployed as a part of the cost function computation in those comparative algorithms. The resulting schedules are depicted in Fig. 8. The dark and striped slots denote execution time and idle time of each processing unit, respectively. The system completion time of LSH, CH, HEFT, and EPC were 3922, 3940, 3410, and 3924, respectively. At any rate, this number is of no concern for this study as this overhead could be allocated off-line, thus imposing no performance degradation for the proposed algorithm.

The execution, transmission, and idle energy consumption of each processing unit obtained from LSH, CH, HEFT, and EPC are shown in Tables 6–9, respectively. Note that LSH algorithm has the least execution energy usage but ranks third in the total energy usage as the algorithm concerns mainly execution energy. The least transmission energy usage belongs to CH algorithm, although the algorithm yielded the most total energy consumption as it concerns mainly transmission energy. HEFT has the least execution time, but came in the second place for the total energy usage as the algorithm concerns mainly the idle and execution time. The EPC yielded good execution and total energy usages because the algorithm selected the

**Table 7**
Energy consumption in each processing unit by CH algorithm in the first case.

| Processing unit | Execution energy | Transmission energy | Idle energy | Total energy |
|---|---|---|---|---|
| *a* | 235.17 | 9.90 | 0.00 | 245.07 |
| *b* | 110.50 | 9.87 | 42.47 | 162.84 |
| *c* | 30.08 | 4.00 | 100.63 | 134.71 |
| *d* | 124.00 | 9.12 | 44.14 | 177.26 |
| Total | 499.75 | 32.89 | 187.24 | 719.88 |

**Table 8**
Energy consumption in each processing unit by HEFT algorithm in the first case.

| Processing unit | Execution energy | Transmission energy | Idle energy | Total energy |
|---|---|---|---|---|
| *a* | 152.09 | 10.35 | 4.35 | 166.79 |
| *b* | 97.50 | 13.16 | 44.02 | 154.68 |
| *c* | 67.20 | 4.00 | 68.44 | 139.64 |
| *d* | 171.12 | 6.24 | 17.75 | 195.12 |
| Total | 487.91 | 33.75 | 134.56 | 656.22 |

**Table 9**
Energy consumption in each processing unit by EPC algorithm in the first case.

| Processing unit | Execution energy | Transmission energy | Idle energy | Total energy |
|---|---|---|---|---|
| *a* | 37.52 | 9.90 | 14.79 | 62.21 |
| *b* | 107.25 | 15.98 | 46.87 | 170.10 |
| *c* | 99.20 | 15.00 | 49.94 | 164.14 |
| *d* | 210.18 | 7.68 | 15.49 | 233.35 |
| Total | 454.15 | 48.56 | 127.09 | 629.80 |

**Table 10**
Execution cost of dependent task graph of Fig. 9 for the second case.

| Task | $\alpha_a t_a(v_i)$ | $\alpha_b t_b(v_i)$ | $\alpha_c t_c(v_i)$ |
|---|---|---|---|
| 1 | 14 | 16 | 9 |
| 2 | 13 | 19 | 18 |
| 3 | 11 | 13 | 19 |
| 4 | 13 | 8 | 17 |
| 5 | 12 | 13 | 10 |
| 6 | 13 | 16 | 9 |
| 7 | 7 | 15 | 11 |
| 8 | 5 | 11 | 14 |
| 9 | 18 | 12 | 20 |
| 10 | 21 | 7 | 16 |

**Table 11**
Execution cost of dependent task graph of Fig. 9 for the third case.

| Task | $\alpha_a t_a(v_i)$ | $\alpha_b t_b(v_i)$ | $\alpha_c t_c(v_i)$ |
|---|---|---|---|
| 1 | 14 | – | 9 |
| 2 | 13 | 19 | – |
| 3 | – | – | 19 |
| 4 | 13 | 8 | – |
| 5 | – | – | 10 |
| 6 | – | 16 | – |
| 7 | 7 | – | 11 |
| 8 | 5 | – | – |
| 9 | – | 12 | – |
| 10 | 21 | – | – |

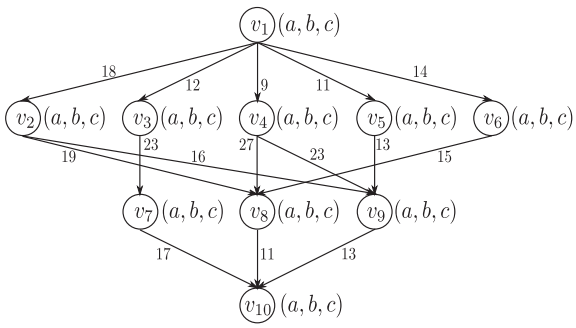


**Fig. 9.** The dependent task graph with similar transmission and execution costs from [25] used in the second case.

optimal server assignment for minimal execution and transmission energy consumption, and proceeded to reduce idle energy to attain the best total energy performance. In addition, the optimal time-line might not be reached as it is not the prime directive of EPC algorithm (see Tables 10 and 11).

The second case used a task graph taken from [25] as shown in Fig. 9. Each number at each edge of the graph denotes the transmission time. The transmission energy was determined in terms of delay, which was different for every task. The execution energy of each task depended on a processing unit. The waiting energy was set to 50% of execution energy. Scheduling of a task limited a processing unit to use only one unit of execution energy per one unit of execution time and one unit of transmission energy per one unit of transmission time. The execution energy, transmission energy, idle energy, and total energy used by each algorithm are shown in Fig. 10. The system completion time of LSH, CH, HEFT, and EPC were 80, 139, 80, and 127, respectively. The reason why LSH and HEFT ended up with the shortest computation time is because they focused primarily on execution time and did not take energy consumption issue into consideration. On the other hand, the transmission and idle energy consumption was equal to zero in EPC algorithm because all processing units were capable of executing every task and processing unit $a$ required the minimum energy.

For the third case using the task graph shown in Fig. 9, the same capabilities and conditions of the processing units as those in the second case were adopted. The results are shown in Fig. 11. The system completion time of LSH, CH, HEFT, and EPC were 102, 116, 102, and 116, respectively.

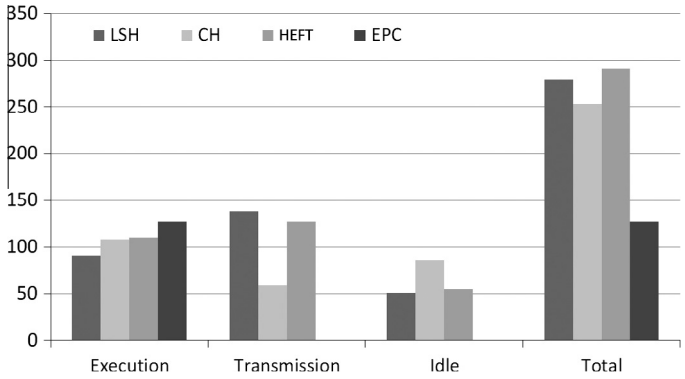


**Fig. 10.** Results of energy consumption for the second case. Note that in case of transmission and idle situations, the results from EPC were not shown because all tasks in EPC were executed only at processing unit *a*.

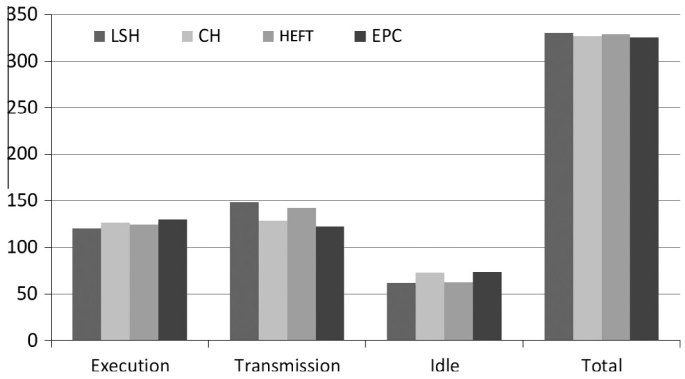


**Fig. 11.** Results of energy consumption for the third case.

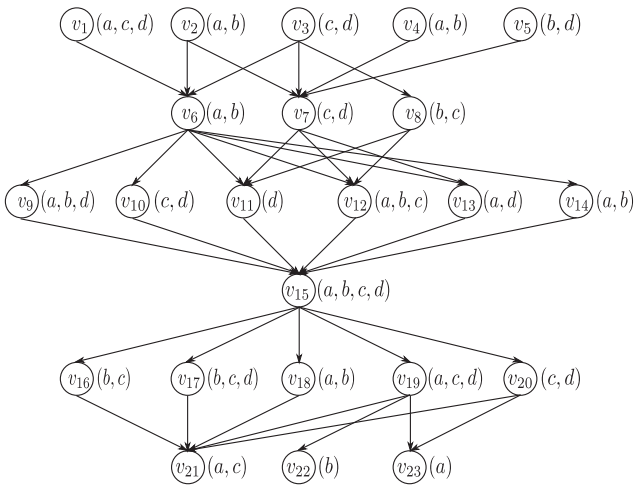


**Fig. 12.** The dependent task graph for the fourth case.

Similar results to those of the second case mean that the adopted capabilities and conditions have a little effect on the performance of each algorithm.

The fourth case uses a generated task graph shown in Fig. 12. The transmission time was reduced to 10% of the execution time. The execution time and data size of this case for each task are shown in Table 12. The same estimated energy consumption and data transmission coefficients of each processing unit used in the first case were used in this case as shown in

**Table 12**
A list of transmission data of each task $d_a(v_i)$ and execution time by each processing unit $t_a(v_i)$ for the fourth case.

| Task | Amount of transmission data | Amount of execution time | | | |
|---|---|---|---|---|---|
| $v_i$ | $d_a(v_i)$ | $t_a(v_i)$ | $t_b(v_i)$ | $t_c(v_i)$ | $t_d(v_i)$ |
| 1 | 25 | 350 | – | 570 | 430 |
| 2 | 30 | 430 | 270 | – | – |
| 3 | 20 | – | – | 510 | 450 |
| 4 | 45 | 590 | 710 | – | – |
| 5 | 60 | – | 920 | – | 790 |
| 6 | 70 | 360 | 480 | – | – |
| 7 | 30 | – | – | 840 | 720 |
| 8 | 55 | – | 710 | 380 | – |
| 9 | 20 | 390 | 670 | – | 380 |
| 10 | 25 | – | – | 550 | 580 |
| 11 | 15 | – | – | – | 620 |
| 12 | 65 | 870 | 790 | 720 | – |
| 13 | 20 | 560 | – | – | 540 |
| 14 | 10 | 430 | 410 | – | – |
| 15 | 85 | 280 | 260 | 270 | 250 |
| 16 | 35 | – | 520 | 480 | – |
| 17 | 20 | – | 380 | 390 | 420 |
| 18 | 15 | 610 | 690 | – | – |
| 19 | 30 | 250 | – | 270 | 480 |
| 20 | 10 | – | – | 620 | 710 |
| 21 | 70 | 760 | – | 890 | – |
| 22 | 30 | – | 540 | – | – |
| 23 | 35 | 580 | – | – | – |

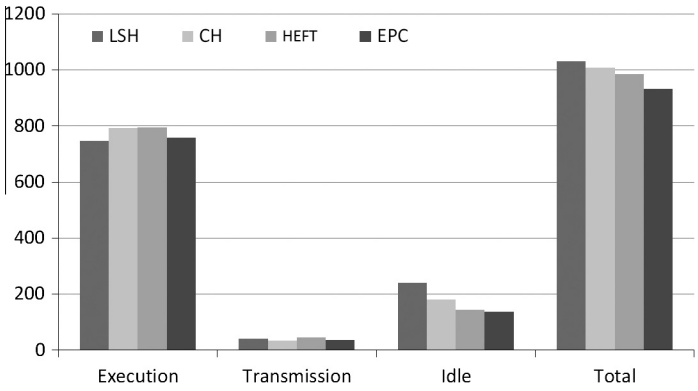


**Fig. 13.** Results of energy consumption for the fourth case.

Table 2. The execution energy, transmission energy, idle energy, and total energy used by each algorithm are shown in Fig. 13. The system completion time of LSH, CH, HEFT, and EPC were 6278, 6299, 5230, and 5450, respectively. HEFT algorithm gave the minimum scheduling length, while EPC algorithm gave the lowest total energy. As seen from Fig. 13, a significant reduction of the transmission energy from the total energy consumption was given by CH algorithm. In EPC algorithm, additional idle energy reduction helped shorten the scheduling length at the small expense of execution and transmission energy. However, EPC algorithm still yielded the minimum total energy.

## 6. Conclusion

A new set of task scheduling and assignment algorithms focusing on minimizing energy consumption, named Energy-Efficient Process Clustering Assignment algorithm or EPC algorithm, was proposed. In this study, multiple capabilities of each processing unit was involved during the task scheduling and enhance the assignment flexibility. The proposed Algorithm 4 significantly reduces the idle time and further minimizes the energy consumption which is not properly handled by the other compared algorithms. The experimental results showed that EPC algorithm spent less energy consumption than those from LSH, CH, and HEFT algorithms in different scenarios. Although the significant benefits of EPC algorithm was confirmed by the experimental results, the performance of the algorithm still depends upon the provided information on estimated transmission data of each task, execution time of each task, and the computational capability of each processing unit.

## Acknowledgments

This work is supported by the Office of the Higher Education Commission, Ministry of Education, Thailand.

## References

[1] W. Alsalih, S. Akl, H. Hassanein, Energy-aware task scheduling: toward enabling mobile computing over MANETs, in: 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE Computer Society, Denver, Colorado, USA, 2005, pp. 1–8.
[2] V. Rao, N. Navet, G. Singhal, A. Kumar, G.S. Visweswaran, Battery Aware Dynamic Scheduling for Periodic Task Graphs, 20th International Parallel and Distributed Processing Symposium (IPDPS), IEE, Rhodes Island, Greece, 2006.
[3] G. Varsamopoulos, S.K.S. Gupta, Energy proportionality and the future: metrics and directions, in: 39th International Conference on Parallel Processing Workshops (ICPPW), IEEE Computer Society, San Diego, California, USA, 2010, pp. 461–467.
[4] Y.C. Lee, A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Transactions on Parallel and Distributed Systems 22 (8) (2011) 1374–1381.
[5] R. Bajaj, D.P. Agrawal, Improving scheduling of tasks in a heterogeneous environment, IEEE Transactions on Parallel and Distributed Systems 15 (2) (2004) 107–118.
[6] X. Qin, T. Xie, An availability-aware task scheduling strategy for heterogeneous systems, IEEE Transactions on Computers 57 (2) (2008) 188–199.
[7] F. Ding, K. Li, An improved task scheduling algorithm for heterogeneous system, in: International Joint Conference on Computational Sciences and Optimization (CSO), IEEE Computer Society, Sanya, Hainan Island, China, 2009, pp. 90–94.
[8] A. Shokripour, M. Othman, H. Ibrahim, S. Subramaniam, New method for scheduling heterogeneous multi-installment systems, Future Generation Computer Systems 28 (8) (2012) 1205–1216.
[9] R. Chang, C.Y. Lin, C.F. Lin, An adaptive scoring job scheduling algorithm for grid computing, Information Sciences 207 (2012) 79–89.
[10] H.F. Sheikh, I. Ahmad, Z. Wang, S. Ranka, An overview and classification of thermal-aware scheduling techniques for multi-core processing systems, Sustainable Computing: Informatics and Systems 2 (3) (2012) 151–169.
[11] S. Albers, H. Fujiwara, Energy-efficient algorithms for flow time minimization, ACM Transactions on Algorithms 3 (4) (2007).
[12] M. Goraczko, J. Liu, D. Lymberopoulos, Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems, in: 45th ACM/IEEE Design Automation Conference (DAC), ACM/IEEE, Anaheim, California, USA, 2008, pp. 191–196.
[13] A. Bokar, M. Bozyigit, C. Sener, Energy-aware dynamic server selection and task allocation, in: 23rd International Symposium on Computer and Information Sciences (ISCIS), IEEE, Istanbul, Turkey, 2008, pp. 1–6.
[14] A. Boker, M. Bozyigit, C. Sener, Scalable energy-aware dynamic task allocation, in: International Conference on Advanced Information Networking and Applications Workshops (WAINA), IEEE Computer Society, Bradford, UK, 2009, pp. 371–376.
[15] G. Terzopoulos, H.D. Karatza, Performance evaluation and energy consumption of a real-time heterogeneous grid system using DVS and DPM, Simulation Modelling Practice and Theory 36 (2013) 33–43.
[16] S. Darbha, D.P. Agrawal, Optimal scheduling algorithm for distributed-memory machines, IEEE Transactions on Parallel and Distributed Systems 9 (1) (1998) 87–95.
[17] V. Liberatore, Multicast scheduling for list requests, in: Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), IEEE, San Francisco, California, USA, 2002, pp. 1129–1137.
[18] J. Li, M. Qiu, J. Niu, T. Chen, Battery-aware task scheduling in distributed mobile systems with lifetime constraint, in: 16th Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, Yokohama, Japan, 2011, pp. 743–748.
[19] T. Yang, A. Gerasoulis, DSC: scheduling parallel tasks on an unbounded number of processors, IEEE Transactions on Parallel and Distributed Systems 5 (9) (1994) 951–967.
[20] J. Liou, M.A. Palis, A comparison of general approaches to multiprocessor scheduling, in: Proceedings of the 11th International Parallel Processing Symposium (IPPS), IEEE Computer Society, Geneva, Switzerland, 1997, pp. 152–156.
[21] S. Zikos, H.D. Karatza, Performance and energy aware cluster-level scheduling of compute-intensive jobs with unknown service times, Simulation Modelling Practice and Theory 19 (1) (2011) 239–250.
[22] G.L. Stavrinides, H.D. Karatza, Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes, Future Generation Computer Systems 28 (7) (2012) 977–988.
[23] I. Ahmad, Y. Kwok, On exploiting task duplication in parallel program scheduling, IEEE Transactions on Parallel and Distributed Systems 9 (9) (1998) 872–892.
[24] D. Bozdag, U. Catalyurek, F. Ozguner, A task duplication based bottom-up scheduling algorithm for heterogeneous environments, in: 20th International Parallel and Distributed Processing Symposium (IPDPS), IEEE, Rhodes Island, Greece, 2006.
[25] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.
[26] Tom's Hardware, Power Consumption: System/CPU Peak. <http://www.tomshardware.com/charts/desktop-cpu-charts-2010/Power-Consumption-System-CPU-Peak,2435.html> (last visited August 2013).
[27] Tom's Hardware, Power Consumption: System Idle, <http://www.tomshardware.com/charts/desktop-cpu-charts-2010/Power-Consumption-System-Idle,2434.html> (last visited August 2013).