Overall System Value Maximization for Resource Constrained Heterogeneous Real-Time Systems

Li Wang, Zheng Li, Shangping Ren
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois, United States
email: {lwang64, zli80, ren}@iit.edu

Gang Quan
Department of Electrical and Computer Engineering
Florida International University
Miami, FL, United States
email: gang.quan@fiu.edu

***Abstract*—Nowadays, many different applications can be consolidated to the same hardware platform, especially as multi-core platforms become mainstream. These applications may have different user perceived values and may or may not share the same composing tasks. When system resources become scarce and not all applications can be executed, how to select and deploy a subset of these applications to maximize the system value is a challenging problem. In this paper, we first formulate the problem as an integer linear programming problem, and then present a computationally efficient heuristic algorithm, i.e., the *Max-Min-Min* algorithm, to appropriately select applications and map their tasks to different processing elements such that all the timing requirements of the applications are met and the overall system value is maximized. The experimental results demonstrate that the proposed heuristic approach provides a good balance between the solution quality and the solution computation cost when compared with the solutions obtained by CPLEX solver and by other commonly used heuristic algorithms, respectively.**

***Index Terms*—Application Selection, Task Deployment, Real-time Embedded System, Heterogeneous Processors**

I. Introduction

Nowadays, different applications are consolidated to the same hardware platform to meet the growing demand for diverse functionalities [1], [2]. Furthermore, for a given system, although the functionalities of the applications may be different, their composing tasks can be shared. A simple example is shown below, which is a simplified version of the example given in [3].

Example 1: Assume there are two applications in a traffic surveillance system. One is to record vehicles' information that pass by the crossroad, and the other one is to report a speeding event to the nearest policeman. To facilitate the first application, we need to perform the following tasks: (1) vehicle detection, (2) speed estimation, and (3) data storing. The second application consists of following tasks: (1) vehicle detection, (2) speed estimation, (3) speed checking, and (4) event reporting. The two applications and their composing tasks are shown in Fig. 1.

From Fig. 1, we can see that although the functionalities of these two applications are different, *vehicle detection* task and *speed estimation* task are shared among the two applications. In other words, the vehicle detection task and speed estimation task are executed only once and the results are used by both applications. □

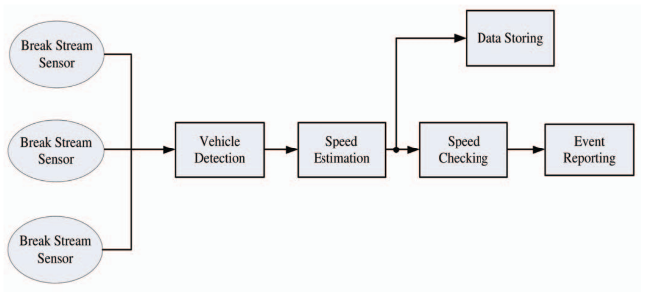

Fig. 1. Task sharing between two applications [3]

For a given system, the value that different applications contribute to the system may different at different times. For instance, in above example (Example 1), the application which reports speeding event may be more important than vehicle information recording application during the day, and recording vehicle information may be more critical at night. When system resources are limited and not all applications can be executed with guaranteed QoS, decisions have to be made as to which applications should be selected so that the overall system value is maximized. In addition, real-time systems often use a variety of heterogeneous processors, such as digital signal processing (DSP) chips [4], graphic processing units (GPUs), and general-purpose processors, due to the need for high-performance. Because of the processor heterogeneity, task execution times may be different when they are deployed on different processors.

Given a set of applications and characteristics of a hardware platform, the goal of the paper is to judiciously choose a subset of the applications such that all tasks can meet their deadlines and the system value is maximized. This is a typical NP-hard problem with its time complexity increasing exponentially with the number of tasks and processors [5]. Traditionally, this problem can be solved using methods such as the genetic algorithm [6] and simulated annealing [7]. These approaches work well during the off-line design phases when the design parameters are well-defined and timing complexity is important but not critical. However, as mentioned before, the value function for each application in our case may change from time to time. Run-time variations may also change the architecture (e.g. core wear out fault) and application characteristics (e.g.

execution times) of a system. Therefore, a computationally efficient heuristic becomes more desirable to accommodate the run-time variations and maximize the system value. In this paper, we make the following main contributions:

- We formulated the problem of system value maximization with resource constraints as an integer linear programming (ILP) problem;
- We proposed an efficient heuristic approach that can provide solution which is close to the optimal solution with polynomial time complexity;
- We conducted extensive experiments to study the performance of the proposed approach by comparing it with the theoretically optimal one (i.e. CPLEX solver [8]) and other heuristic approaches in the literature.

The reminder of this paper is organized as follows. In Section II, we discuss the related work. The system model and problem definition are presented in Section III. The ILP formulation of the problem is presented in Section IV. In Section V, we present a heuristic algorithm to select the applications and decide the processors upon which the tasks associated with the selected applications are deployed. The experimental results are discussed in Section VI. Finally, in Section VII, we conclude our work and point out the future work.

II. Related Work

The research regarding task deployment has been intensively studied from different perspectives. For example, some research focuses on deploying task set on *identical* processors [9]–[13]; while in [14]–[16], the research is to investigate the feasibility of deploying task set on a given set of *uniform* processors.

Anderson et al. take a step further to partition processors into two categories, i.e, type-1 and type-2 processors, and study the task deployment problem on 'semi-heterogeneous' processors [17]. Baruah in [5] proposed a polynomial time algorithm to decide whether a set of tasks can be deployed to a collection of heterogeneous processors. The algorithm first transforms the task deployment problem into an Integer Linear Programming (ILP) problem, and then applies the Linear Programming (LP) relaxation technique to solve this problem. The algorithm works well only when the number of processors is small as the time complexity of the linear programming relaxation algorithm is $O(m^m)$ where m is the number of processors.

Gopalakrishnan et al. also studied the task deployment problem on heterogeneous multiprocessor systems [18]. They proposed a heuristic algorithm [1] to deploy tasks in a way that the maximum utilization of all processors is minimized. Armstrong et al. presented a *Minimum Execution Time* (MET) algorithm [19] to minimize the makespan when scheduling a collection of tasks among a set of heterogeneous processors. Unlike [18], MET algorithm assigns a task to the processor on which the task's execution time is minimized.

[1]We call it utilization balanced (UB) algorithm in the paper for easy reference.

As we can see that the major differences between the work mentioned above and ours are twofolds: the objective of the work discussed above is to meet *all* task deadlines; while ours is to maximize overall system value with the constraint that all *selected* tasks must meet their deadlines; and the model of the work mentioned above is based on a set of *independent tasks*; while our model is based on a set of *applications which may share tasks*.

It is also worth pointing out that although the problem we are to address in the paper is similar to the multidimensional knapsack problem (MKP) [20], [21] on surface, the essence of these two problems are different. In MKP, the amount of resources required by different items on different knapsacks is given and fixed. In our problem, the utilization demand of different applications is unknown.

In [22], a Utility Accrual (or UA) real-time scheduling algorithm is proposed to schedule a set of tasks. Each task is associated with a time-utility function (TUF), and the scheduling goal is to maximize total accrual utility. Our work has similar concept in that we also aim to maximize overall system value. The differences lie in that the UA model assumes task values are independent; while in our model, values are applied to applications. As applications may share tasks, the simple value-monotonic approach becomes insufficient.

III. System Models and Problem Formulation

We assume that a real-time embedded system hosts a set of applications. Each application consists of a set of independent real-time periodic tasks. Some tasks may be shared among different applications. These tasks are executed on a set of heterogeneous processors. To better formulate our problem, we first introduce the following notations and definitions:

- **Processor Set** $\Pi = \{\pi_1, \pi_2, \cdots, \pi_k\}$ represents all processors (i.e. total k) in the system, where π_i denotes processor i.
- **Application Set** $\mathcal{A} = \{\alpha_1, \alpha_2, \cdots, \alpha_m\}$ represents all candidate applications in the system (i.e. total m), where α_i represents application i.
- **Application Value Vector** $\overrightarrow{V}_m = [v_1, v_2, \cdots, v_m]$ represents the value that each application can contribute to the system if all of its tasks are completed before their deadlines, where v_i is the value application for α_i.
- **Task Set** $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ represents the total number of real-time tasks (i.e. total n) in application set $\mathcal{A}$, where τ_i represents the i-th real-time task. Each real-time task τ is characterized by two parameters, i.e. $\tau = (e, p)$, where e and p represent the task execution time and period, respectively. We assume all tasks are released at the beginning of each period and the deadline is the end of its period.
- **Application-Task Matrix** $\mathbf{A}_{m\times n}$ defines the task composition for each application in $\mathcal{A}$. Specifically, $\mathbf{A}_{m\times n} = (b_{i,j})_{m\times n}$, where $b_{i,j} \in \{0, 1\}$. $b_{i,j} = 1$ indicates that

application α_i contains task τ_j. Task τ_j does not belong to application α_i if $b_{i,j} = 0$.

- **Task-Execution-Time Matrix** $\mathbf{E}_{n\times k}$ indicates the different execution times for each type of tasks running on different PEs. Specifically, $\mathbf{E}_{n\times k} = (e_{i,j})_{n\times k}$, where $e_{i,j} \in \Re^+ \cup \{+\infty\}$ represents the execution time when processor π_j only executes task τ_i. We use $e_{i,j} = +\infty$ to indicate that processor π_j cannot perform task τ_i.

We make the following assumptions:

- Assumption 1: each task can only be deployed to at most one processor.
- Assumption 2: every task that belongs to a selected application must be deployed to a processor.
- Assumption 3: on each processor, preemptive Earliest Deadline First (EDF) scheduling policy is applied.

With the above definitions and assumptions, the problem of maximizing overall system value under resource constraints can be formulated as follows:

Problem 1: Given Π, $\mathcal{A}$, $\overrightarrow{V}_m$, Γ, $\mathbf{A}_{m\times n}$, $\mathbf{E}_{n\times k}$ as defined above, select $\mathcal{A}_s \in \mathcal{A}$ and assign tasks in $\mathcal{A}_s$ (i.e. $\Gamma_s \in \Gamma$) to Π such that all tasks in Γ_s can meet deadlines and the overall value of $\mathcal{A}_s$ is maximized. □

The problem defined above is a typical resource-constrained optimization problem, which has been proved to be NP-hard [5]. In what follows, we first formulate the problem as an ILP problem. We then present a more computationally efficient heuristic to solve this problem.

IV. The ILP formulation

In this section, we formulate Problem 1 as an ILP problem. The most significant advantage of the ILP formulation is that we can obtain the theoretical optimal solution to Problem 1 using available ILP solvers (such as CPLEX solver [8]).

To formulate the problem into an ILP problem, we first define two variables:

- **Application Selection Vector** $\overrightarrow{A}_m$ identifies the applications to be chosen in the system. Specifically, $\overrightarrow{A}_m = [a_1, a_2, \cdots, a_m]$, where $a_i \in \{0, 1\}$. Application α_i is selected if $a_i = 1$, or $a_i = 0$, otherwise.
- **Task-Deployment Matrix** $\mathbf{D}_{n\times k}$ determines how tasks are assigned to different processors. Specifically, $\mathbf{D}_{n\times k} = (d_{i,j})_{n\times k}$, where $d_{i,j} \in \{0, 1\}$ and $d_{i,j} = 1$ indicates that task τ_i is deployed on processor π_j, and $d_{i,j} = 0$ indicates the opposite.

For a given task-deployment matrix $\mathbf{D}_{n\times k}$, based on Assumption 1, we have

$$\sum_{j=1}^{k} d_{i,j} \leq 1 \quad i = 1, 2, \ldots, n \tag{1}$$

In addition, to ensure Assumption 2, we have

$$a_i \times b_{i,j} \leq \sum_{l=1}^{k} d_{j,l} \quad j = 1, 2, \ldots, n; \;\; i = 1, 2, \ldots, m \tag{2}$$

As preemptive EDF scheduling policy is used on each processor (Assumption 3), in order to guarantee all deployed tasks meeting their deadlines, we have to ensure that the utilization demand U_j on processor π_j $(1 \leq j \leq k)$ is within the bound given by Liu et al. [23], i.e.,

$$U_j(\Gamma_j) = \sum_{\tau_i \in \Gamma_j} \frac{e_{i,j}}{p_i} \leq 1 \quad j = 1, 2, \ldots, k \tag{3}$$

where Γ_j refers to the set of tasks deployed on processor π_j, $e_{i,j}$ and p_i are task τ_i's execution time on processor π_j and period, respectively. As such, the ILP formulation of Problem 1 can be described as follows:

$$\text{maximize} \qquad \nu = \overrightarrow{A}_m \times (\overrightarrow{V}_m)^T \tag{4}$$

Subject to:

$$\sum_{j=1}^{k} d_{i,j} \leq 1 \quad i = 1, 2, \ldots, n \tag{5}$$

$$a_i \times b_{i,j} \leq \sum_{l=1}^{k} d_{j,l} \quad j = 1, 2, \ldots, n; \;\; i = 1, 2, \ldots, m \tag{6}$$

$$\sum_{i=1}^{n} \frac{e_{i,j}}{p_i} \times d_{i,j} \leq 1 \quad j = 1, 2, \ldots, k. \tag{7}$$

where $a_i, d_{i,j}, b_{i,j} \in \{0, 1\}$.

As simple as the ILP formulation may seem to be, its computational cost grows exponentially with the size of the problem, i.e. the numbers of applications, tasks, processors, etc. Since the application values may change from time to time, and in order to accommodate the dynamics in the runtime environment, it is necessary that we solve Problem 1 on the fly. The traditional approaches, i.e., CPLEX solver [8], genetic algorithm [6] and simulated annealing [7], etc., become infeasible for large and complex system. In what follows, we present our heuristic to solve this problem with comparable results but much less computational cost.

V. A heuristic approach to maximizing overall system value

As shown in our problem formulation, to solve the problem, we need to solve two sub problems: how to judiciously select a subset of applications, and how to map the corresponding tasks to processors. In what follow, we discuss our approach to each sub problem accordingly.

A. Application Selection Criteria

Before presenting our approach to application selection, we first use a simple example to gain some intuitions.

Example 2: Assume that a system has three processors $\Pi = \{\pi_1, \pi_2, \pi_3\}$, five applications $\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$, with the value of each application given in vector $\overrightarrow{V}_5 = [v_1, v_2, v_3, v_4, v_5] = [100, 45, 70, 50, 60]$. Assume that all tasks in the application set are $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_7\}$. The relationship between the applications and the tasks is given in Application-Task matrix $\mathbf{A}_{5\times 7}$.

$$\mathbf{A}_{5\times 7} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The execution time of each task τ_i $(1 \le i \le 7)$ on processor π_j $(1 \le j \le 3)$ is shown in Task-Execution-Time matrix $\mathbf{E}_{7\times 3}$.

$$\mathbf{E}_{7\times 3} = \begin{pmatrix} +\infty & 8 & +\infty \\ 13.5 & +\infty & +\infty \\ 1 & 1.5 & 2 \\ 3.2 & +\infty & 6.4 \\ 8 & 6 & +\infty \\ +\infty & 9.6 & +\infty \\ 9 & +\infty & +\infty \end{pmatrix}$$

where $+\infty$ indicates that the task cannot be deployed to the processor.

The periods of the tasks are given in $\overrightarrow{P}_7 = [p_1, p_2, \cdots, p_7] = [10, 15, 5, 8, 10, 12, 10]$. Based on the Task-Execution-Time matrix $\mathbf{E}_{7\times 3}$ and the period vector $\overrightarrow{P}_7$, the utilization demand of task τ_i on processor π_j, i.e., $u_{i,j} = e_{i,j}/p_i$, is shown in the Task-Utilization matrix $\mathbf{U}_{7\times 3}$ as below.

$$\mathbf{U}_{7\times 3} = \begin{pmatrix} +\infty & 0.8 & +\infty \\ 0.9 & +\infty & +\infty \\ 0.2 & 0.3 & 0.4 \\ 0.4 & +\infty & 0.8 \\ 0.8 & 0.6 & +\infty \\ +\infty & 0.8 & +\infty \\ 0.9 & +\infty & +\infty \end{pmatrix}$$

To choose the proper subset of applications, one simple and intuitive approach is to select an application with the highest value and then optimally deploy its tasks to available processors. In this example, as application α_1 has the largest value, we select application α_1 and deploy its task τ_1 and τ_2 to processor π_2 and π_1, respectively. As there are still resources remaining, application α_3 which has the largest value in the remaining applications becomes the candidate for selection. However, as not all of its tasks can be successfully deployed to the processors because of the utilization constraint (3), application α_3 cannot be chosen. Similar situation happens for the other remaining applications. Therefore, if we select applications purely based on their values, only application α_1 can be supported, and the total system value is 100.

From the above example, we can see that choosing applications solely based on their values cannot achieve good performance in terms of maximizing the total value since the application with larger value may demand more computation resources. Another heuristic is therefore to select the application based on the value-resource ratio r_i defined by (8):

$$r_i = \frac{v_i}{\sum\limits_{\tau_j \in \Gamma_i} avg(u_j)} \tag{8}$$

where Γ_i refers to the task set that application α_i contains, and $ave(u_j) = (\sum_{u_{i,j} \neq +\infty} u_{i,j})/\eta$ is the average utilization demand of task τ_j, and η is the total number of processors on which the utilization of task τ_j is not equal to $+\infty$.

Based on (8), the ratio r_i of application α_1, $\alpha_2, \cdots$, α_5 is $r_1 = 58.82$, $r_2 = 50$, $r_3 = 53.85$, $r_4 = 62.5$, and $r_5 = 66.67$, respectively. As application α_5 has the largest ratio, we select application α_5 and deploy its task τ_7 to processor π_1. Then we select application α_4 and deploy its task τ_6 to processor π_2. The remaining applications cannot be supported because of the utilization constraint (3). In this case, the overall system value is 60 + 50 = 110, which is better than the case in which we select the applications based only on their values.

In fact, the optimal solution is to select application α_2 and α_3, and deploy task τ_3, τ_4, and τ_5 to processor π_1, π_3, and π_2, respectively, and the overall system value is 45 + 70 = 115. One observation we can obtain from the optimal solution is that although both applications α_2 and α_4 have two tasks which is twice as many as application α_4 and α_5, these two applications share task τ_4. This observation suggests that if task sharing is taken into consideration, the application selection strategy may improve the solution quality.

$\square$

Based on the discussion in Example 2, we consider the following three factors when deciding application selection criteria:

1) the application value;
2) the application's *remaining* task utilization demand, i.e., tasks that have not be deployed by previous selections;
3) the potential for task sharing among unselected applications.

In particular, our selection criteria c_i is defined as following:

$$c_i = \frac{v_i}{\sum_{\tau_j \in \Gamma_i'} avg(u_j)/s_j} \tag{9}$$

where $s_j = \sum_{i=1}^{m} b_{i,j}$ is the number of applications which share the task τ_j. Γ_i' is the set of tasks that belong to application α_i, but have not been deployed by other selected applications. $ave(u_j) = (\sum_{u_{i,j} \neq +\infty} u_{i,j})/\eta$ is the average utilization demand task τ_j, and η is the total number of processors on which the utilization of task τ_j is not equal to $+\infty$.

It is worth mentioning that each time when tasks of an application are deployed, the value c_i of remaining applications may change because of task sharing. To illustrate how applications are selected by using criteria c_i, consider an example given below.

Example 3 (Example 2 revisited): Assume the number of processors, applications, tasks, relationship between applications and tasks, and the utilization matrix are the same as given in Example 2. According to formula (9), the value c_i for each application is $c_1 = 58.82, c_2 = 75, c_3 = 62$, and $c_5 = 66.67$.

As the value of c_2 for application α_2 is the largest, therefore, we select application α_2, and deploy its task τ_3 and τ_4 to processor π_1 and π_3, respectively. As application α_2 shares task τ_4 with application α_3, the value of c_3 for application α_3

becomes $c_3 = 100$, and the value of c_i for application α_1, α_4, and α_5 remains the same. Hence, application α_3 is selected, and we deploy its task τ_5 to processor π_2. As the remaining processing resources after deploying application α_2 and α_3 are not sufficient to support α_1, or α_4, or α_5. Hence, the total system value becomes 45 + 70 = 115, which is larger than the cases in which the applications are selected purely by the values or the value-resource ratios.

□

B. Deploying Tasks to Processors

In this section, we discuss how to deploy a set of tasks to different processors. Before presenting our task deployment algorithm, we first introduce the following definitions.

Definition 1 (Feasible Deployment): For a given task set $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$, the deployment of the tasks is feasible if and only if it satisfies the following conditions.

$$\sum_{j=1}^{k} d_{i,j} = 1 \quad i = 1, 2, \ldots, n \tag{10}$$

$$\sum_{i=1}^{n} \frac{e_{i,j}}{p_i} \times d_{i,j} \leq 1 \quad j = 1, 2, \ldots, k. \tag{11}$$

where $d_{i,j} = 1$ denotes task τ_i is deployed on processor π_j, otherwise, it is not. $e_{i,j}$ refers to the execution time of task τ_i on processor π_j, p_i is the period of task τ_i, k and n is the number of processors and tasks, respectively.

□

Constraint 10 ensures that each task $\tau_i \in \Gamma$ is deployed to one and only one processor, and Constraint 11 guarantees that all tasks on each processor meet their deadlines.

Definition 2 (Remaining Utilization): For a given processor π_j, the remaining utilization U_j' is defined as

$$U_j' = 1 - \sum_{\tau_i \in \Gamma_j} \frac{e_{i,j}}{p_i} \tag{12}$$

where Γ_j refers to the set of tasks on processor π_j.

□

Definition 3 (Available Processor): For task τ_i, processor π_j is an available processor if processor π_j satisfies (13).

$$u_{i,j} \leq U_j' \tag{13}$$

where U_j' is processor π_j's remaining utilization, and $u_{i,j}$ is the utilization demand of task τ_i on processor π_j.

□

We call the heuristic we developed as a *Max-Min-Min* approach. In particular, we first get the minimum utilization demand of each task to be deployed, then we deploy the tasks in decreasing order of their minimum utilization, i.e., the task with maximum minimum utilization demand will be deployed first, and we deploy it to the processor on which the utilization demand is minimized.

Algorithm 1 implements the Max-Min-Min heuristic for task deployment. It works as follows. From Line 3 to Line 11, task with maximum minimum utilization demand is selected and deployed to the available processor on which the task's utilization demand is minimized. The process is repeated until either all undeployed tasks have been deployed, or there exists a task that does not have an available processor to execute it. In the later case, we set the feasibility to false and remove all previously deployed tasks from the processors.

Algorithm 1 DEPLOY TASKS TO THE PROCESSORS

```
Input: The set of tasks to be deployed $\Gamma$, Processor set $\mathcal{A}$,
    Task-Utilization matrix $\mathbf{U}_{n \times k}$.
Output: Task deployment that ensures all tasks meet their
    deadlines.
 1: $feasibility \leftarrow true$
 2: while $\Gamma$ is not empty do
 3:   Choose task $\tau_i$ with the maximum minimum utilization
      demand.
 4:   if no available processor exists for task $\tau_i$ then
 5:     $feasibility \leftarrow false$
 6:     Remove the previously deployed tasks in $\Gamma$ from the
        processors.
 7:     break
 8:   else
 9:     Deploy task $\tau_i$ to the processor on which the task's
        utilization is minimized
10:     Remove task $\tau_i$ from set $\Gamma$.
11:   end if
12: end while
13: return $feasibility$
```

Complexity Analysis: In the while loop (from Line 3 to Line 11), as the time required to find the undeployed task with maximum minimum utilization demand is $O(kn)$ and the while loop itself executes at most n times. Therefore, the time complexity of Algorithm 1 is $O(kn^2)$, where n is the number of tasks to be deployed and k is the number of processors.

C. A Max-Min-Min based Application Selection and Task Deployment Approach

In Section V-A and Section V-B, we present a criteria to select applications and a heuristic approach to deploy tasks, respectively. In this subsection, we present an integrated algorithm for application selection and task deployment.

Let Γ_i denote the tasks that belong to application α_i and Γ_D denote the tasks that have been deployed before application α_i. As application α_i may share tasks with previously selected applications, hence when deploying application α_i's tasks, we only consider those which have not been deployed before, i.e., undeployed task set Γ_i':

$$\Gamma_i' = \Gamma_i \backslash \Gamma_D \tag{14}$$

When tasks in Γ_i' are to be deployed, it is possible that previously deployed tasks 'occupied' resources that should have been allocated to the tasks in current consideration. To illustrate this situation, consider a simple example given below.

Example 4: Assume a system has two processors $\Pi = \{\pi_1, \pi_2\}$ and two applications $\mathcal{A} = \{\alpha_1, \alpha_2\}$. The value of application α_1 and α_2 is 115 and 34, respectively. The two applications have four tasks in total, i.e., $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. The relationship between applications and tasks is given in Application-Task Matrix $\mathbf{A}_{2\times 4}$.

$$\mathbf{A}_{2\times 4} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

The utilization of each task on the processors is shown in Task-Utilization matrix $\mathbf{U}_{4\times 2}$ as below.

$$\mathbf{U}_{4\times 2} = \begin{pmatrix} 0.6 & 0.5 \\ 0.4 & 0.8 \\ 0.7 & 0.6 \\ 0.3 & 0.1 \end{pmatrix}$$

According to the application selection criteria (9), the value of c_i for application α_1 and α_2 is $c_1 = 100$ and $c_2 = 40$, respectively. Therefore, application α_1 is selected, and by using task deployment Algorithm 1, we deploy task τ_1 and τ_2 to processor π_2 and π_1, respectively.

When application α_2 is selected, we cannot deploy its task τ_3 to either of the processors because of the utilization constraint. Therefore, in this case, only application α_1 is supported by the system, and the overall system value is 115.

However, when we consider application α_2, if we allow its tasks and α_1's tasks to be re-deployed together, i.e., consider to task τ_1, τ_2, τ_3, and τ_4 together as the input task set to Algorithm 1, then task τ_3 will be the first to be deployed because it has the maximum minimum utilization demand, i.e., 0.6, and it is deployed processor π_2. Task τ_1 is then selected because it has the maximum minimum utilization demand in the remaining tasks. Similarly, task τ_2 to processor π_1 and task τ_4 to processor π_2.

In this case, all four tasks can be successfully deployed to the processors, application α_1 and α_2 are supported and the overall system value is $115 + 34 = 149$, which is larger than the previous case.

□

From Example 4, we can see that when deploying a newly selected application's tasks, if we mix new tasks with previously deployed tasks and deploy them as a whole, the chance that the selected application's tasks can be successfully deployed to the processors increases. Algorithm 2 shows the detailed procedure of application selection and task deployment.

A brief explanation of Algorithm 2 is as follows. Line 1 to Line 3 initialize the supported application set $\mathcal{A}_s$, deployed task set Γ_D, and overall system value ν. Line 3 to Line 4 calculate the value of c_i for each application, and select the application with largest c_i. When application is selected, its tasks are combined with previously deployed tasks and input to Algorithm 1 for deployment (Line 6). If the deployment is feasible, the overall system value (ν) is increased by v_i, otherwise, we restore the deployment of the previous tasks (Line 12). The application selection is stopped until all the applications have been checked.

Algorithm 2 MAX-MIN-MIN BASED APPROACH TO SELECTING APPLICATIONS AND DEPLOYING TASKS

```
Input: Processor set $\Pi$; Application set $\mathcal{A}$; Application value vector $\overrightarrow{V}_m$; Task set $\Gamma$; Task-Utilization matrix $\mathbf{U}_{n\times k}$; Application-Task matrix $\mathbf{A}_{m\times n}$.
Output: overall system value $\nu$.
 1: $\mathcal{A}_s \leftarrow \emptyset$, $\Gamma_D \leftarrow \emptyset$, $\nu \leftarrow 0$
 2: while Application set $\mathcal{A}$ is not empty do
 3:     Calculate the value of $c_i$ for each application in $\mathcal{A}$ by using (9).
 4:     Choose the application $\alpha_i$ with largest $c_i$.
 5:     $\Gamma_A \leftarrow \Gamma_D \cup \Gamma_i$
 6:     Deploy task set $\Gamma_A$ by using Algorithm 1 $(\Gamma_A, \Pi, \mathbf{U}_{n\times k})$.
 7:     if mapping is feasible then
 8:         $\nu \leftarrow \nu + v_i$
 9:         $\mathcal{A}_s \leftarrow \mathcal{A}_s + \{\alpha_i\}$
10:         $\Gamma_D \leftarrow \Gamma_D \cup \Gamma_i$
11:     else
12:         Restore the deployment of tasks of applications in $\mathcal{A}_s$.
13:     end if
14: end while
15: return $\nu$
```

Complexity Analysis: For the while loop, the time required to calculate the value of c_i for each application is $O(kmn)$. As the time complexity of Algorithm 1 is $O(kn^2)$. Therefore, the time complexity of the while loop is $O(kn^2 + kmn)$. As the while loop requires to run m times, the time complexity of Algorithm 2 is $O(kn^2m + knm^2)$.

VI. Experiment Results

In this section, we conducted three sets of experiments. The purpose of the first set of experiments is to compare the computation cost between heuristic approaches and the CPLEX solver [8]. The second set of experiments is to investigate the performance of the Max-Min-Min based approach. In particular, we compare the overall system value obtained by the Max-Min-Min based approach with the optimal value obtained by using CPLEX solver, UB [18] based approach, and Minimum Execution Time (MET) [19] based approach. The third set of experiments is to investigate how the Max-Min-Min based approach performs when the system size is large.

A. Experiment Settings

Based on the utilization constraint (3), we know that when the system size is decided, i.e., the number of processors, tasks, and applications is fixed, the feasibility that a set of tasks can be scheduled on the processors is affected by the total utilization demand of the task set. If the total utilization demand of a given application set is larger than the total computation resources that processor set can provide, some tasks have to be dropped and hence some applications may not be executed. Therefore, in order to compare the performance of different approaches, we run the test cases with different utilization demand of the task set.

In our implementation, we use UUnifast algorithm [24] to generate the utilization demand for each task. For a given total utilization demand of n tasks U, the UUnifast algorithm uniformly distributes U to task τ_i with $0 < u_i < U$ $(1 \leq i \leq n)$ and guarantees $U = \sum_{i=1}^{n} u_i$.

When there are k processors, we first obtain the utilization demand u_i for task τ_i $(1 \leq i \leq n)$. We then apply the UUnifast algorithm again to decide the utilization demand of task τ_i on processor π_j, i.e., $u_{i,j}$. Because of processor's heterogeneity, some tasks may not be able to execute on certain processors. To reflect this, for each task τ_i, we first generate a random number k_0 in the range of $[0, \lceil k \times \varphi \rceil]$, where k is the number of processors in the system, and φ is a real number which is used to adjust the maximum number of unfeasible processors for the tasks, and $0 \leq \varphi \leq 1$. In our experiment, we set $\varphi = 0.3$. When k_0 is selected, we randomly select k_0 number of processors (Π_∞) and set the task τ_i's utilization on these processors to be $+\infty$.

For the rest $(k - k_0)$ processors, we apply the UUnifast algorithm again to decide the utilization demand of task τ_i on processor π_j where $\pi_j \notin \Pi_\infty$ within the range of $(0, u_i \times (k - k_0))$. This way, the average utilization demand of task τ_i on a single processor remains the same, i.e., u_i.

For all the sets of experiments, the value of applications is uniformly distributed from 1 to 200, the number of tasks that each application has is uniformly chosen from 1 to n (i.e., the number of tasks), and the tasks for each application are also randomly selected. In addition, for both UB based approach and MET based approach, application selection criteria (9) is used to select the applications.

B. Computational Costs

We first compare the computational costs of different approaches for solving the application selection and task deployment problem. In particular, for a given set of system parameters, we apply a commercial linear programming solver (CPLEX), the proposed Max-Min-Min based approach, the UB, and the MET based approaches to find appropriate applications and feasible task deployment, and compare the time they take to reach a solution.

Table I gives the execution times of different approaches under different sizes. From Table I we can see that the execution time of CPLEX solver and its difference from other approaches increases very quickly when the system size increases. For instance, when the number of processors is 20, number of tasks is 80, and the number of applications is 120, i.e., $k = 20$, $n = 80$, and $m = 120$, the execution time of CPLEX solver is 6312 seconds, and the execution time of MET, UB, and Max-Min-Min based approaches is only 0.028, 0.033, and 0.776 seconds, respectively.

C. Optimality of the Max-Min-Min Heuristic

In this set of experiments, we are to investigate the optimality of the Max-Min-Min based approach by comparing the overall system value obtained by the Max-Min-Min based approach with the optimal value obtained by using CPLEX solver. We set the system size to $k = 10$, $n = 40$, and $m = 60$, and the optimality of different approaches is evaluated based on the ratio between overall system value obtained by different approaches and the total application value.

TABLE I
RUNNING TIME COMPARISON UNDER DIFFERENT SYSTEM SIZE (k, n, m)

Approaches	(5, 20, 30)	(10, 40, 60)	(15,60,80)	(20,80,120)
MET	0.003	0.004	0.010	0.028
UB	0.003	0.004	0.011	0.033
Max-Min-Min	0.012	0.023	0.183	0.776
CPLEX Solver	2.57	68.4	537	6312

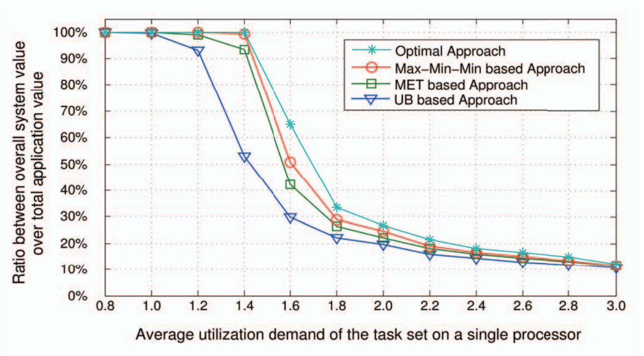

Fig. 2. Performance comparison when $k = 10, n = 40, m = 60$

Fig. 2 shows the performance of different approaches. From Fig. 2, we can see that the Max-Min-Min based approach obtains close to optimal overall system value. In addition, the Max-Min-Min based approach outperforms both UB and MET based approaches. The maximal difference between Max-Min-Min based approach and UB based approach is over 48% when the average utilization demand is 1.4, and for MET based approach, the maximal difference between Max-Min-Min based approach is 10% when the average utilization demand is 1.6.

The reason that Max-Min-Min based approach performs better than UB and MET based approaches is that for UB based approach, the task is deployed to minimize the maximum utilization of the processors. In other words, UB algorithm does not consider the utilization demand of the task but the total utilization on each processor. Therefore, the task's actual utilization demand will be higher comparing with our Max-Min-Min based approach. For MET based approach, tasks are randomly selected and allocated to the processors with minimum utilization demand, while in the Max-Min-Min based approach, tasks with maximum minimum utilization will be deployed first so that the probability that all tasks of an application can be successfully deployed to the processors increases.

D. Scalability of the Max-Min-Min Heuristic

For this set of experiments, we investigate the performance of Max-Min-Min based approach when the system size becomes large. In particular, we set the number of processors to 30, the number of tasks to 120, and number of applications to 180. That is $k = 30$, $n = 120$, and $m = 180$.

From Fig. 3, we can observe that Max-Min-Min based approach performs better than both UB based approach and MET based approach when the system size is large. For instance, when the average utilization demand of the task set on a single processor is 1.6, the overall system value ratio obtained by using Max-Min-Min algorithm is larger than MET based approach and UB based approach by 42% and 73%, respectively.

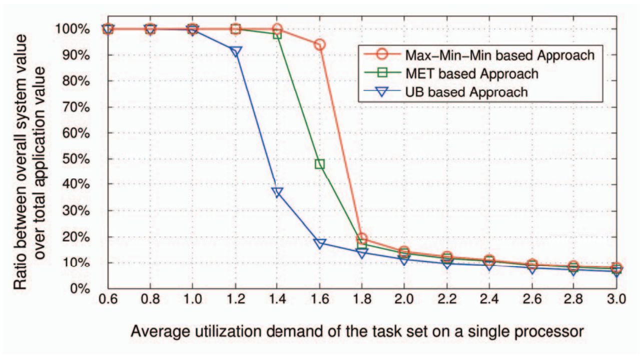

Fig. 3. Performance comparison when $k = 30, n = 120, m = 180$

From the three sets of experiments, we can conclude that the proposed Max-Min-Min based approach provides close to optimal solution for the overall system value optimization problem, but with much less computation cost and it outperforms other heuristic approaches existed in the literature.

VII. Conclusion

In this paper, we have investigated the problem of obtaining maximized overall system value under resource constraints. The uniqueness of the problem lies in that different applications may share tasks; while most research in the literature assumes that applications are independent. We have presented a heuristic approach that takes task sharing into consideration when deciding which applications to support and how to deploy the associated tasks to heterogeneous processors. The simulation results show the good performance of the proposed approach. In this paper, although we have considered the situation where applications may share tasks, we assume tasks are independent. We are to extend the work to the area when there are dependencies among tasks.

Acknowledgement

The work is supported in part by NSF CAREER CNS-0746643, NSF CNS-1018731, and NSF CNS-1035894.

References

[1] W. Wolf, A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 10, pp. 1701 –1713, Oct. 2008.

[2] J. Liu, E. Cheong, and F. Zhao, "Semantics-based optimization across uncoordinated tasks in networked embedded systems," in *Proceedings of the 5th ACM international conference on Embedded software*, ser. EMSOFT '05. New York, NY, USA: ACM, 2005, pp. 273–281.

[3] J. Liu and F. Zhao, "Composing semantic services in open sensor-rich environments," *Network, IEEE*, vol. 22, no. 4, pp. 44 – 49, july 2008.

[4] M. C. Yovits, *Advances in computers*. Academic Press, 1993.

[5] S. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *Proceedings of Real-Time and Embedded Technology and Applications Symposium*, may 2004, pp. 536 – 543.

[6] L. Wang, H. J. Siegel, V. R. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. Parallel Distrib. Comput.*, vol. 47, pp. 8–22, November 1997.

[7] M. H. Kashani and M. Jahanshahi, "Using simulated annealing for task scheduling in distributed systems," in *Proceedings of the Conference on Computational Intelligence, Modelling and Simulation*, 2009, pp. 265–269.

[8] CPLEX, "http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/," March 2012.

[9] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *27th IEEE International Real-Time Systems Symposium*, Dec. 2006, pp. 101 –110.

[10] R. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, pp. 1–40, 2011.

[11] S. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Systems*, vol. 32, pp. 9–20, 2006.

[12] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of sporadic task systems," in *Real-Time Systems Symposium*, dec. 2005, pp. 9 pp. –329.

[13] J. Lopez, M. Garcia, J. Diaz, and D. Garcia, "Worst-case utilization bound for edf scheduling on real-time multiprocessor systems," in *12th Euromicro Conference on Real-Time Systems*, 2000, pp. 25 –33.

[14] S. Baruah and J. Goossens, "The edf scheduling of sporadic task systems on uniform multiprocessors," in *Real-Time Systems Symposium, 2008*, Dec. 2008, pp. 367 –374.

[15] T. Gonzalez and S. Sahni, "Preemptive scheduling of uniform processor systems," *J. ACM*, vol. 25, pp. 92–101, January 1978.

[16] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," in *22nd IEEE Real-Time Systems Symposium*, Dec. 2001, pp. 183 – 192.

[17] B. Andersson, G. Raravi, and K. Bletsas, "Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors," in *IEEE Real-Time Systems Symposium*, Dec. 2010, pp. 239 –248.

[18] S. Gopalakrishnan and M. Caccamo, "Task partitioning with replication upon heterogeneous multiprocessor systems," in *Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 199 – 207.

[19] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Heterogeneous Computing Workshop. Proceedings of Seventh*, March 1998, pp. 79 – 87.

[20] J. Puchinger, G. R. Raidl, and U. Pferschy, "The multidimensional knapsack problem: Structure and algorithms," *INFORMS J. on Computing*, vol. 22, no. 2, pp. 250–265, Apr. 2010.

[21] A. Freville, "The multidimensional 0-1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1 – 21, 2004.

[22] H. Cho, B. Ravindran, and E. D. Jensen, "Utility accrual real-time scheduling for multiprocessor embedded systems," *J. Parallel Distrib. Comput.*, vol. 70, pp. 101–110, February 2010.

[23] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, January 1973.

[24] E. Bini and G. C. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, 2004, pp. 196–203.