Runtime Resource Management with Workload Prediction

Mina Niknafs
Linköping University, Sweden
mina.niknafs@liu.se

Ivan Ukhov
Gears of Leo AB, Sweden
ivan.ukhov@leovegas.com

Petru Eles
Linköping University, Sweden
petru.eles@liu.se

Zebo Peng
Linköping University, Sweden
zebo.peng@liu.se

ABSTRACT

Modern embedded platforms need sophisticated resource managers in order to utilize the heterogeneous computational resources efficiently. Moreover, such platforms are exposed to fluctuating workloads unpredictable at design time. In such a context, predicting the incoming workload might improve the efficiency of resource management. But is this true? And, if yes, how significant is this improvement? How accurate does the prediction need to be in order to improve decisions instead of doing harm? By proposing a prediction-based resource manager aimed at minimizing energy consumption while meeting task deadlines and by running extensive experiments, we try to answer the above questions.

ACM Reference Format:
Mina Niknafs, Ivan Ukhov, Petru Eles, and Zebo Peng. 2019. Runtime Resource Management with Workload Prediction . In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3316781.3317902

1 INTRODUCTION

In modern heterogeneous architectures, multiple computational resources are brought together in order to provide high performance for different applications. Integrating multiple heterogeneous cores in a single architecture is an important technique for obtaining performance benefits; however, this can only be attained if a platform based on this architecture is equipped with an appropriate resource manager (RM) for making decisions such as task mapping, task scheduling, and voltage and frequency scaling.

Issues are made more complicated since, most of the times, platforms of this type are exposed to fluctuating workloads that are unknown at design time. In this context, considering a prediction of the future workload, in addition to the current state of the platform, should potentially improve the efficiency of the resource management decisions. While this is intuitively appealing, there are many very interesting questions that have to be answered. What would an RM that takes workload prediction into account look like? Does workload prediction actually improve the efficiency of resource management? And, if yes, how significant is this improvement? And how accurate would the prediction need to be in order to help improve decisions instead of, possibly, doing harm?

In this paper, our goal is to answer the above questions in the context of a resource management system that makes mapping and scheduling decisions for incoming tasks such that deadlines are satisfied with minimum energy consumption. The predicted parameters are the nature and the timing of incoming requests (tasks). In this paper, we do not address the issue of prediction itself; instead we rely on our previous work [12, 13], in which we show that, in real-life traces, there are patterns in the type of resource requests and in their interarrival times, and that these patterns can be used for modeling and prediction of the future workload. The proposed methods for prediction were tested on the Google cluster data set [14]. On average, the next arrival time was predictable with an error of less than 17%, and the incoming request type was predictable with an accuracy of 80%–95%. It should be noted that we selected prediction methods that have a small inference overhead in order to be applicable at runtime.

Related work and contribution: There is a vast amount of literature on resource management for embedded systems. An extensive survey can be found in [16]. Quasi-static techniques have been proposed in order to handle variable workloads [15]. This means that a set of resource allocation solutions corresponding to executed workload instances (phases) are prepared off-line. These solutions are applied at runtime, depending on the actual workload. However, such an approach cannot be applied in the context of a variable workload whose dynamic features are largely unknown at design time. The approach in [11] performs design-time profiling and optimization in order to derive a set of Pareto-optimal configurations for each application. At run-time, depending on the actual working conditions, the resources that are assigned to each active application are determined based on the configurations that were previously derived off-line. A similar approach is used in [6], where near-optimal static mappings are generated at design time for each application. The run-time system uses these near-optimal static mappings and, based on a model of the underlying architecture, determines the actual mapping and scheduling such that the properties of the static mappings are preserved. Approaches based on feedback control [5, 9] and machine learning (e.g., reinforcement learning) [4] have been proposed for particular applications in order to design adaptable resource managers. In [7], in order to maximize overall throughput, the finishing time of an application is estimated based on historical runtime information. A predictive user-level scheduler based on past performance history is proposed in [8]. Prediction techniques have been used in the context of thermal prediction as well [2]. Density predictions of global workloads at an aggregate level have been used for resource management in clouds [3, 17]. Here, the granularity level of prediction is the CPU usage or the density of arriving request streams, and the actual resource management actions are shutting down or starting up workstations. As opposed to this, prediction aided resource management with a fine granularity, at the task level, has not been addressed. *The contribution of this paper is the investigation of such resource management policies in the context of heterogeneous embedded platforms.*

2 SYSTEM MODEL AND MANAGEMENT

We consider a heterogeneous platform consisting of N computation resources r_i, $i = 1, \ldots, N$. The platform executes a fluctuating workload as the response to a stream of requests. In order to focus on the main point of interest which is prediction, we consider a relatively simple workload model. Each request req_j triggers a specific activity denoted as a task τ_j. Each task τ_j for $j = 1, \ldots, L$ is characterized by:

- arrival time s_j, the time of the arrival of request req_j;
- deadline d_j, relative to the arrival time;
- average energy consumption $e_{j,i}$, when τ_j is executed on resource r_i, for all r_i[1];
- worst case execution time (WCET) $c_{j,i}$ when τ_j is executed on resource r_i, for all r_i [1];
- energy and time overhead $em_{j,k,i}$ and $cm_{j,k,i}$ due to migration of task τ_j from resource r_k to resource r_i, for each r_i and r_k.

Following each request req_j, at time s_j, the RM has to decide the resource to which to map the corresponding task τ_j and the moment in time at which to schedule the start of its execution. We assume that the tasks are firm real-time, which means that they have to meet their deadlines in order for their output to be of any use. If they miss their deadlines their result would be useless. Tasks are preemptable, except when executed on particular resources (e.g. GPUs), in which case they cannot be preempted and continued afterwards, but need to run to the end in order to produce results.

At each task arrival the RM considers the current context of active tasks under execution and the new task τ_j to be activated as response to the request req_j. It tries to find a mapping and schedule for task τ_j such that it satisfies its deadline. To this end, it might preempt running tasks, remap, and reschedule them - taking the involved migration overheads into consideration (with exception of e.g. GPUs - see above). If there is no solution such that all tasks meet their deadlines, τ_j will not be admitted. If prediction is employed, in addition to the current context and the arriving task τ_j, the RM also considers the task τ_p corresponding to the predicted request req_p and its predicted arrival time s_p, when deciding on mapping and scheduling of τ_j. The RM takes its decisions such that energy consumption is minimized.

In the rest of the paper we propose such a resource manager and analyze the impact of prediction quality on its performance. Given that an essential feature of the systems we consider is a fluctuating workload, unpredictable at design time, it is unavoidable that, in certain contexts, the workload demand cannot be satisfied and tasks will not be admitted. Nevertheless, there might exist applications that are safety critical and request strong guarantees regarding hard deadlines. In order to deliver such guarantees, those applications and their resource demand have to be known at design time. Thus, for the set of these safety-critical hard real-time tasks, the resource allocation decisions are taken at design time, such that the requested constraints are satisfied. Well established static or quasistatic techniques can be employed to this end which seamlessly integrate with the resource management policy discussed in this paper [16]. These decisions are stored for online use by the resource manager. At their arrival, the resource manager allocates with the highest priority the required resources to the critical applications and continues to apply the adaptive resource allocation technique, proposed in this paper, over the remaining set of resources.

3 MOTIVATIONAL EXAMPLE

Let us consider the example in Fig. 1. We have a heterogeneous architecture with two CPUs and one GPU where the RM is supposed to manage the available resources such that the energy consumption is minimized, and that all deadlines are met. Assume that τ_1 and τ_2 are submitted to the RM with the parameters given in Table 1.

Table 1: Task parameters

	s_j	d_j	WCET (ms)			Energy consumption (J)		
			CPU1	CPU2	GPU	CPU1	CPU2	GPU
τ_1	0	8	8	12	5	7.3	8.4	2
τ_2	1	5	7	8.5	3	6.2	7.5	1.5

Suppose the RM makes decisions solely based on the current state of the system and dedicates the GPU to τ_1 at time 0. Then, at time 1 when τ_2 arrives, the only chance for it to meet its deadline is to execute on the GPU which is running τ_1. Since τ_1 is running, it has to be aborted and restarted from the beginning (due to the nature of GPU) either on the GPU, after τ_2, or on the CPU1 at time 1. None of these solutions guarantees the deadlines. Therefore, the RM rejects τ_2; see (a) in Fig. 1. In this case, the acceptance rate is $1/2$. If the RM had foreseen the arrival of τ_2 at time 1, it would have mapped τ_1 to CPU1 and reserved the GPU for τ_2. This would have prevented the rejection of τ_2; see (b) in Fig. 1. In this case, the acceptance rate would be $2/2$.

Let us assume that the prediction for τ_2 to arrive at time 1 is inaccurate, and that it arrives, in fact, at time 3. Due to the (inaccurate) prediction, τ_1 is mapped to CPU1, and the GPU is available to run τ_2. Both tasks meet their deadlines with a total energy consumption of 8.8 J. The RM without prediction, however, maps τ_1 on the GPU, and at time 3 it maps τ_2 on the same GPU to be executed starting at time 5. Both tasks meet their deadlines with much less energy consumption, namely 3.5 J. The prediction accuracy has a significant effect on the efficiency of resource management, and inaccurate prediction might even become harmful.

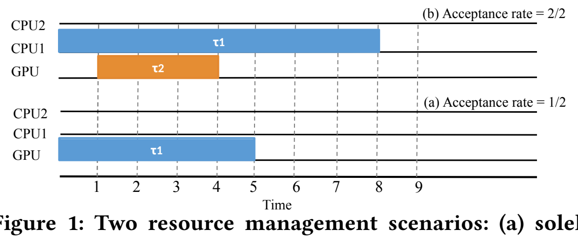


Figure 1: Two resource management scenarios: (a) solely based on the current state and (b) based on the current state and prediction.

4 RESOURCE MANAGEMENT WITH PREDICTION

In this section, we describe our approach to resource management with prediction. First, in Sec. 4.1, the notations and conventions that are used in the following sections are introduced. Then, the problem of exact optimization of task mapping and scheduling is addressed in Sec. 4.2. Finally, a heuristic solution to the problem will be introduced in Sec. 4.3.

4.1 Prerequisites

- For any activation of the RM, at a certain time t, we denote with $\overline{S}$ the set of all tasks that have been admitted before the time t and are not yet finished plus the task arrived as result of the current request and (if with prediction) the task corresponding to the predicted request.

[1]Tasks need to be executable at least on one resource. If a task is not executable on a certain resource, WCET and energy consumption for that resource are indicated as specific dummy values.

- When the resource manager is activated at a certain time t, let us consider τ_j as one of the tasks currently running on resource r_i. We remind that $c_{j,i}$ and $e_{j,i}$ are the WCET and the average energy consumption of the task on r_i. We denote by $cp_{j,i}$ and $ep_{j,i}$ the (worst case) run time not yet consumed and the average energy not yet consumed for τ_j on r_i at current time t (if the task is not yet started $cp_{j,i} = c_{j,i}$ and $ep_{j,i} = e_{j,i}$). If the RM decides to migrate τ_j to another resource r_k, then the execution time not yet executed and the energy not yet consumed are $cp_{j,k} = c_{j,k} \times (cp_{j,i}/c_{j,i})$, and $ep_{j,k} = e_{j,k} \times (cp_{j,i}/c_{j,i})$.
- The time window considered by the RM at each activation at time t is the interval between the current time t and the moment in time defined by the latest deadline of all tasks in the set $\overline{S}$. We denote the length of this time window by $\overline{K} = \max_{\tau_j \in \overline{S}}(t_{\text{left}_j})$, where $t_{\text{left}_j} = s_j + d_j - t$; here $s_j + d_j$ is the absolute deadline of task. Inside this time window $\overline{K}$ the RM will decide on the mapping and scheduling of all tasks in $\overline{S}$. On each resource the scheduling is performed according to the optimal earliest deadline first (EDF) policy. If no prediction is used there is no preemption between two activations of the RM. Thus, the RM will order the tasks on each resource according to their deadline. Here, by task we mean complete tasks (if they have not been started yet) or the pieces of tasks remained to be executed for tasks under execution at time t. If prediction is used and the predicted task has a deadline earlier than another task in the set $\overline{S}$, then the schedule produced by RM is considering the preemption caused by the predicted task. Such preemption is not applied to a GPU. Let us mention that the mapping and scheduling of the predicted task are only used as a constraint in order to find an efficient mapping and schedule for the current task, that takes the future arrival into consideration. The actual predicted task will be effectively mapped and scheduled when and if it actually arrives.

If prediction is used and a feasible mapping and scheduling are derived, the arriving task is admitted. However, if no solution is found so that all tasks meet their deadline, that does not mean that the arriving task is rejected. In this case, a mapping and scheduling solution without considering the predicted request is attempted and if it is successful the new task is admitted.

4.2 MILP formulation for exact optimization

We use Mixed Integer Linear Programming (MILP) as our exact optimization method. In order to make the equations more readable, in the following, we have used simplified formulations indicating that some of the constraints, as explained below, have to be satisfied only under certain conditions. When encoding these equations as linear constraints for the solver, the big-M method has been utilized [1]. The notation t in the equations denotes the current time t at which the RM is activated. The formulation is as follows:

$$\text{minimize} \sum_{j|\tau_j \in \overline{S}} \sum_{i=1}^{N} x_{j,i} \times (ep_{j,i} + em_{j,k,i})$$

$$\text{subject to: } \forall \tau_j \in \overline{S}: \sum_{i=1}^{N} x_{j,i} = 1 \tag{1}$$

$$\forall \tau_j \in \overline{S}: \sum_{i=1}^{N} x_{j,i} \times cpm_{j,i} \le t_{\text{left}_j} \tag{2}$$

The mapping variables are denoted by x_{ji} where $x_{ji} = 1$ if task τ_j is mapped to resource i; otherwise $x_{ji} = 0$. We denote by $cpm_{j,i}$ the total execution time of τ_j including the migration cost of the case that the task is relocated during the current time window; $cpm_{j,i} = cp_{j,i}$ if the task is not relocated and $cpm_{j,i} = cp_{j,i} + cm_{j,k,i}$ if τ_j is migrated from r_k to r_i. If τ_j is migrated from r_k to r_i during the current time window, $em_{j,k,i}$ is the energy overhead for the migration; otherwise it is zero. The constraints in (1) enforce that each task is mapped to one and only one resource. The constraints in (2) ensure that, if τ_j is mapped to r_i, its execution time on this resources is not longer than t_{left_j}; otherwise, its deadline cannot be met. The scheduling constraints (3) ensure that all tasks mapped to resource r_i meet their deadline. This constraint applies to all resources r_i, except the resource to which the predicted task τ_p is mapped. SL is the list of tasks in $\overline{S}$ sorted by their deadline. The summation is over the index k in the sorted list. We remind (see Sec. 4.1) that, according to EDF, the RM sorts the tasks mapped to each resource according to their deadline. The constraints impose that each task finishes before its deadline.

$$\forall \tau_j \in \text{SL}: \sum_{k=1}^{j} x_{k,i} \times cpm_{k,i} \le t_{\text{left}_j} \tag{3}$$

Let us now consider the predicted task τ_p and the resource r_i to which it is mapped ($x_{p,i} = 1$). If the deadline of τ_p is later than that of all other tasks in $\overline{S}$, there will be no preemption. The task will be scheduled at the time $max(s_p, q_i)$, where s_p is the arrival time of τ_p and q_i is the moment in time when all tasks mapped to r_i (except τ_p) finish their execution. One of the constraints in (4) and (5) must ensure the schedulability of τ_p:

$$q_i - t + x_{p,i} \times cp_{p,i} \le t_{\text{left}_p}; \text{ if } s_p \le q_i \tag{4}$$

$$s_{p,i} - t + x_{p,i} \times cp_{p,i} \le t_{\text{left}_p}; \text{ otherwise.} \tag{5}$$

If the deadline of the predicted task τ_p is earlier than that of some tasks in the set $\overline{S}$ then one of the tasks will be preempted. We divide the ordered list of tasks SL into two sublists: SL1 consists of those tasks whose deadline is earlier than the one of τ_p or equal. SL2 is the list of tasks with deadlines later than τ_p. The tasks in SL1 will not be preempted by τ_p and constraints (6) ensure their schedulability.

$$\forall \tau_j \in \text{SL1}: \sum_{k=1}^{j} x_{k,i} \times cpm_{k,i} \le t_{\text{left}_j} \tag{6}$$

We denote by q_i the finishing time of the last task in sublist SL1 that is mapped to resource r_i. If task τ_p arrives before this time q_i then neither tasks in SL1 nor in SL2 will be preempted. The constraints (7) ensure the schedulability of tasks in sublist SL2 for this case.

$$\forall \tau_j \in \text{SL2}: q_i - t + \sum_{k=1}^{j} x_{k,i} \times cpm_{k,i} \le t_{\text{left}_j} \tag{7}$$

The last case to be considered is if the predicted task τ_p arrives after the moment q_i. In this case, the RM has to plan for a preemption. Potentially any of the tasks in SL2 that are mapped to the same resource with τ_p could be preempted. Which one depends on the arrival time of τ_p. The preempted task τ_j is divided by the preemption point into two chunks, before and after the preemption point, respectively. We denote the start time of the execution of a chunk of task τ_j mapped to resource r_i by $sc_{j,i,k}$ (k=1 for the first chunk and 2 for the second) and the end time by $ec_{j,i,k}$. These start and end times are optimization variables. In constraints (8) we ensure that the start time of the predicted task is greater than its arrival time. Constraints (9) enforce that the end time of a chunk is after the start time of that chunk. The start of the second chunk should be after the end of the first, which is enforced by constraints (10). Constraints (11) enforce that the total execution time of the two chunks is equal with the execution time of the task. The constraints (12) or (13) must be satisfied to ensure that chunks do not overlap for each r_i. Constraint (14) guarantees that no deadlines are violated.

$$sc_{p,i,1} \geq s_p * x_{p,i} \tag{8}$$

$$\forall \tau_j \in \text{SL2}: (\forall k \in \{1,2\}: sc_{j,i,k} - ec_{j,i,k} \leq 0) \tag{9}$$

$$\forall \tau_j \in \text{SL2}: ec_{j,i,1} - sc_{j,i,2} \leq 0 \tag{10}$$

$$\forall \tau_j \in \text{SL2}: \sum_{k=1}^{2} (ec_{i,j,k} - sc_{i,j,k}) = cp_{j,i} \times x_{j,i} \tag{11}$$

$$\forall \tau_{j_1} \in \text{SL2}: (\forall \tau_{j_2} \in \text{SL2}, j_1 \neq j_2: (\forall k_1 \in \{1,2\}: (\forall k_2 \in \{1,2\}: ec_{j_1,i,k_1} - sc_{j_2,i,k_2} \leq 0))) \tag{12}$$

$$\forall \tau_{j_1} \in \text{SL2}: (\forall \tau_{j_2} \in \text{SL2}, j_1 \neq j_2: (\forall k_1 \in \{1,2\}: (\forall k_2 \in \{1,2\}: ec_{j_2,i,k_2} - sc_{j_1,i,k_1} \leq 0))) \tag{13}$$

$$\forall \tau_j \in \text{SL2}: ec_{j,i,2} - t \leq t_{\text{left}_j} \times x_{j,i} \tag{14}$$

Due to its complexity, the MILP-based optimization described in this section is not applicable in practice. Nevertheless, we use it in our experiments in order to evaluate the efficiency of the fast heuristic proposed in the next section.

4.3 Fast heuristic

For our fast heuristic, we consider the processing resources as knapsacks with certain capacities, and the tasks are the items with certain weights. The capacity of each resource r_i is expressed in available processing time. At each activation of the RM, this capacity is equal with length of time window $\overline{K}$ introduced in Sec. 4.1. The weight of task τ_j on r_i is equal to $cpm_{j,i}$. Our proposed resource management algorithm is based on the knapsack heuristic presented in [10]. It has the worst case complexity of $O(NLlogL)$, where N is the number of resources and L is the number of tasks. The actual complexity depends on the number of tasks in set $\overline{S}$ (see Sec. 4.1), which at any activation of the resource manager is much smaller than L. The proposed heuristic is described in Algorithm 1.

Lines 1–6 initialize the algorithm. Note that $\overline{y_j} = map(\tau_j)$ is a vector, that specifies, for each task τ_j the index i of the resource that the task is mapped to. If no feasible mapping is found for certain tasks, their $\overline{y_j}$ will be zero, and these tasks will not be allowed to become active by the RM. Let $f_{j,i}$ be a measure of desirability of assigning task τ_j to resource r_i. A smaller $f_{j,i}$ means a lower level of energy consumption, so smaller values are preferred. On line 6, $M \times ((cpm_{j,i}) > t_{\text{left}_j})$ is added to $f_{j,i}$ in order to make it undesirable if τ_j is not executable on r_i; M is a sufficiently large positive number. In this algorithm, the tasks that are not yet mapped to any resource are considered iteratively (line 7), and the task τ_{j^*} that has the maximum difference between the smallest and the second smallest desirability is determined (lines 9–23). Once task τ_{j^*} is identified, one has to decide where it should be mapped (considering the schedulability constraints). The RM tries to map the task to resource r_{i^*} for which f_{j^*,i^*} is minimum (lines 25–28). If the scheduling constraint for resource i^* is violated (considering the tasks mapped to it so far), it tries to map τ_{j^*} to the resource i^* for which f_{j^*,i^*} is the next smallest (lines 29–34). The iteration is continued until the RM manages to map τ_{j^*} to a resource (fulfilling the scheduling constraints), or until all resources have been considered, and no feasible mapping has been found (lines 31–32).

The IsSchedulable function checks the schedulability of τ_{j^*} on resource r_{i^*} given the set of tasks that are mapped on r_{i^*} so far. The scheduling in function IsSchedulable is performed according to the principles outlined in Sec. 4.1 and Sec. 4.2. It is based on the EDF ordering of tasks on each resource and on checking that termination occurs before the deadline. Preemption caused by the predicted task is considered except for nonpreemptable resources, like GPUs. If all tasks are schedulable, IsSchedulable returns true. If, finally, a mapping has been produced such that all tasks meet their deadline the arriving task is admitted.

Algorithm 1 Mapping Heuristic

```
Require: $\overline{S}$ , $N$ , $ep_{j,i}$, $cp_{j,i}$, $\overline{K}$, $em_{j,k,i}$, $cm_{j,k,i}$
Ensure: $\overline{y_j} = map(\tau_j)$
1:  $U = \{1 \dots |\overline{S}|\}$                          ▷ index of active tasks
2:  $R = \{r_1 \dots r_N\}$                                  ▷ index of resources
3:  for each $r_i$ in $R$ do
4:      $\overline{K}_i = K$
5:      for each $\tau_j$ in $\overline{S}$ do
6:          $f_{j,i} = ep_{j,i} + em_{j,k,i} + M \times ((cpm_{j,i}) > t_{\text{left}_j})$
7:  while $U \neq \emptyset$ do
8:      $d^* = -\infty$
9:      for each $j \in U$ do
10:         $F_j = \{r_i \in R | cpm_{j,i} \leq \overline{K}_i\}$
11:         if $F_j \neq \emptyset$ then
12:             $i^* = argmin\{f_{j,i} | i \in F_j\}$
13:             if $F_j \setminus \{i^*\} = \emptyset$ then
14:                 $d = +\infty$
15:             else
16:                 $i' = argmin\{f_{j,i} | i \in F_j \setminus \{i^*\}\}$
17:                 $d = f_{j,i'} - f_{j,i^*}$
18:                 if $d > d^*$ then
19:                     $d^* = d$
20:                     $j^* = j$
21:                     $i^* = i$
22:         else                                         ▷ there is no solution
23:             exit
24:     while $y_{j^*} = 0$ do
25:         if IsSchedulable($j^*$, $i^*$) then
26:             $y_{j^*} = i^*$
27:             $\overline{K}_{i^*} = \overline{K}_{i^*} - cpm_{j^*,i^*}$
28:             $U = U \setminus \{j^*\}$
29:         else               ▷ $\tau_{j^*}$ cannot be scheduled on resource $i^*$
30:             $F_{j^*} = F_{j^*} \setminus \{i^*\}$
31:             if $F_{j^*} = \emptyset$ then                 ▷ no more resources
32:                 exit
33:             else
34:                 $i^* = argmin\{f_{j^*,i} | i \in F_{j^*}\}$     ▷ pick next best $r_i$
```

5 EXPERIMENTAL RESULTS

The generation of experimental workload traces and the system configuration are presented in Sec. 5.1. The first set of experiments, Sec. 5.2 and Sec. 5.3, is performed with the goal to compare resource management with and without prediction. In these experiments, the prediction is considered accurate. The experiments in Sec. 5.4 explore the impact of prediction accuracy on the efficiency of resource management. Finally, in Sec. 5.5, the effect of prediction overhead on the efficiency of the RM is investigated. For all experiments, the performance of our heuristic is compared with that of a hypothetical resource manager that runs the optimal MILP formulation without overhead.

5.1 Trace generation and system configuration

In our experiments, we assume that we have a heterogeneous architecture with five CPUs and one GPU.

Each arriving request includes the following three fields: arrival time, request (task) type, and deadline. In order to create such traces, we create 100 different tasks. For each task, random numbers from $Gaussian(40, 9^2)$ are generated as its WCET on CPUs. Similarly random numbers for $Gaussian(15, 3^2)$ are generated as the energy consumptions on the CPUs. For the GPU, the average execution time on CPUs and the average the energy consumption, respectively are divided by a random number in range 2–10.

After creating the task sets, 500 traces with a length of 500 requests each are created as follows. In order to create the first field of each trace (arrival time), we started from time 0, and for the next arrival, we add the arrival of the pervious task with a sampled number from $Gaussian(1.2, 0.4^2)$. Then, in order to assign a task to each arrival, a task is selected randomly for each sequence of task arrivals. Finally, in order to set a deadline (relative to the arrival time) for each pair of a task and an arrival time we do as follows. First, we select randomly one of its WCETs on the different resources, and we call it $RWCET$. Then, if we assume the arrival time is s_j, the relative deadline is set as follows: $d_j = RWCET \times C$ where C is a coefficient. For having very tight deadlines, we choose randomly smaller coefficients in the range 1.5–2; in order to have less tight deadlines, larger coefficients in the range 2–6 are chosen randomly. To recapitulate, we create two categories of workload traces, and the only difference between them is their deadline field. The first category of 500 traces is the one with very tight deadlines (referred to as the VT group), and the second category of 500 traces is the one with less tight deadlines (referred to as the LT group).

For the migration overhead of a task, in terms of time and energy, we assumed that it is related to the complexity of the task as captured by its WCET and energy consumption. We considered this overhead to be between 0.1 and 0.2 of the average WCET and energy consumption (over all resources) respectively.

5.2 MILP versus heuristic without prediction

Our first goal is to compare the performance of the exact optimization and that of the heuristic, without prediction, in terms of acceptance rate. On average, for the total number of 1000 traces run (VT+LT), the rejection percentage without prediction for the MILP and heuristic are 24.5% and 31%, respectively. We found that out of 1000 traces run, for 88% of them the acceptance percentage with the MILP based model was higher than the one with the heuristic. The fact that this not 100% might be counterintuitive at the first sight. The explanation is that at any instant of its activation the RM takes a locally optimal decision based on the received request and the current state of the system. Nevertheless, it is not excluded that a suboptimal decision taken by the heuristic might turn out more efficient on the long run, in the context of future arriving requests.

5.3 The impact of prediction

Fig. 2a and Fig. 2b depict the average rejection percentage for the MILP and heuristic with respect to the LT and VT group, respectively. The results are shown for the case with accurate prediction (predictor status "on") and without prediction (predictor status "off"). Prediction reduces the rejection percentage by 1% (LT) and 9.17% (VT) for the MILP based RM and by 2.6% (LT) and 10.2% (VT) with the heuristic. We can observe that improvements due to prediction are more significant in the case of tight deadlines, which is expected since the problem becomes easier with loose deadlines. We can also observe that the results with the heuristic are only 4% below those with the MILP (VT) with prediction and 5.5% without prediction.

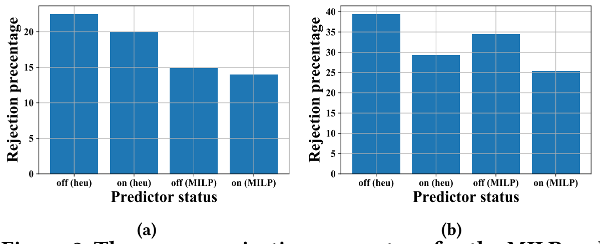


Figure 2: The average rejection percentage for the MILP and heuristic with respect to (a) the LT group and (b) the VT group.

Fig. 3a and Fig. 3b depict the average normalized energy consumption for the MILP and heuristic for the VT and LT traces, respectively. The energy levels closely follow those of the acceptance percentage; smaller rejection percentage results in higher energy consumption since more workload is executed. For the case of tight deadlines (VT) it is also visible that for the optimal MILP solution the relation between reduced rejection percentage and increased energy is more favorable than for a fast and affordable heuristic.

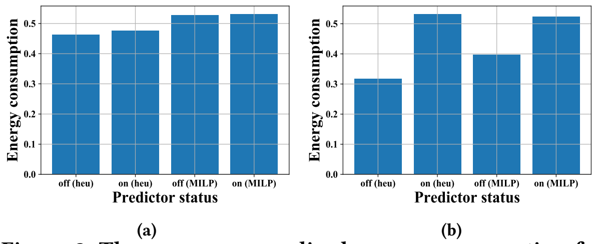


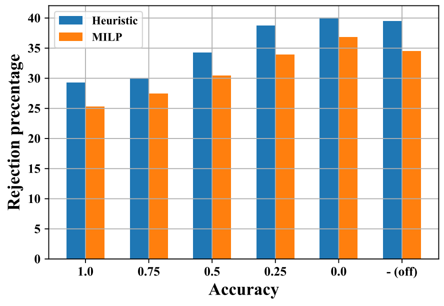
Figure 3: The average normalized energy consumption for the MILP and heuristic with respect to (a) the LT group and (b) the VT group.

5.4 The impact of prediction accuracy levels

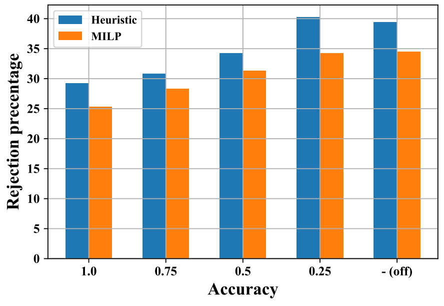
In this experiment, the goal is to explore how different accuracy levels affect the performance of the RM in terms of the rejection percentage. Since the predictor provides both the identity and the arrival time of incoming tasks, errors might be present in any of these two quantities. Fig. 4 depicts the average rejection percentage for different levels of accuracy with respect to the task type and the arrival time, respectively, in the case of the VT traces. For example, an accuracy of 0.75 in Fig. 4a means that the task (request) identity is predicted incorrectly with a probability of 25% at each prediction step but the arrival time is accurate.

In Fig. 4b, 0.75 accuracy value means that the normalized average error (the normalized root mean square error) for the arrival time prediction over the corresponding trace is 0.25.

As expected, by decreasing the prediction accuracy, the rejection percentage increases for both the MILP and heuristic, and it gets close to the rejection percentage in the scenario when the predictor is disabled. The average level of 0.25 already does not offer any sensible benefit compared to no prediction, and this in the context in which prediction overhead is neglected. Of course in concrete cases, the threshold at which using prediction is justified has to be



(a)



(b)

Figure 4: The average rejection percentage for different accuracy levels with respect to (a) the task type and (b) the arrival time for the VT group.

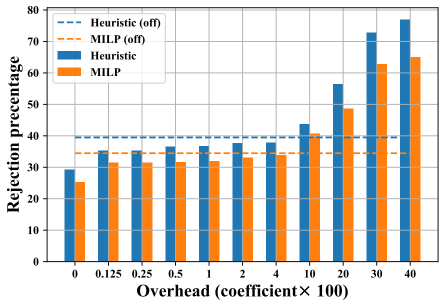


Figure 5: The average rejection percentage for different prediction overheads in the case of the VT group.

considered by the designer taking the overhead into consideration (see the next section).

5.5 The impact of prediction overhead

Various methods for interarrival time and workload modeling and prediction can be employed. These methods imply different runtime overheads. The prediction overhead at runtime is a very important aspect when considering the efficiency of prediction based resource management, in particular in the context of time-sensitive applications. The goal of this experiment is to show how the prediction overhead affects the performance of the MILP and heuristic in terms of the rejection percentage. We assume that predictions are 100% accurate. We impose different prediction overhead according to the following formula: time overhead = coefficient × average interarrival time of the tasks. The result can be seen in Fig. 5 where the horizontal axis corresponds to the aforementioned coefficient multiplied by 100. If the overhead is larger than 2–4 percent of the average interarrival times, the rejection rate with perfectly accurate prediction becomes worse than the one in the case when the predictor is off. Therefore, in order to be able to utilize predictions, the runtime overhead for the selected prediction method should be carefully investigated.

6 CONCLUSION AND FUTURE WORK

In this paper, we have investigated the following questions: does predicting the incoming workload improve the efficiency of resource management? How significant is this improvement? We have shown that, in many cases (for instance, in scenarios with tight deadlines and with high prediction accuracy), the efficiency improves significantly. In cases when prediction accuracy is low or prediction overhead is high the efficiency declines. In cases when constraints e.g. on deadlines are loose the improvements are less significant.

In addition, we have considered the following question: how accurate does the prediction need to be in order to improve decision-making instead of obstructing it? For our traces, we conclude that the accuracy should be at least 50% in order to have a reasonable improvement. Another aspect to note is that the benefit also depends on the prediction overhead; if the overhead is too high, even perfect accuracy reduces the efficiency of resource management.

We have also presented a fast efficient heuristic for prediction aided resource management. Moreover, as we mentioned our previous work has shown that prediction with accuracy of 83% for the next arrival time and 80%–95% for the request type is possible on real-life streams with low overhead.

REFERENCES

[1] Der-San Chen et al. 2011. *Applied integer programming: modeling and solution.* John Wiley & Sons.
[2] Ryan Cochran and Sherief Reda. 2010. Consistent runtime thermal prediction and control through workload phase detection. In *DAC*. 62–67.
[3] Eli Cortez et al. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 153–167.
[4] Anup Das and et al. 2016. Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 2 (2016), 24.
[5] Antonio Filieri et al. 2015. Automated multi-objective control for self-adaptive software design. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 13–24.
[6] Andrés Goens et al. 2017. Tetris: A multi-application run-time system for predictable execution of static mappings. In *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems*. ACM, 11–20.
[7] Chris Gregg et al. 2011. Dynamic heterogeneous scheduling decisions using historical runtime data. In *Workshop on Applications for Multi-and Many-Core Processors (A4MMC)*.
[8] Víctor J Jiménez et al. 2009. Predictive runtime code scheduling for heterogeneous architectures. In *International Conference on High-Performance Embedded Architectures and Compilers*. Springer, 19–33.
[9] Alberto Leva et al. 2018. Event-based power/performance-aware thermal management for high-density microprocessors. *IEEE Transactions on Control Systems Technology* 26, 2 (2018), 535–550.
[10] Silvano Martello. 1990. *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons Ltd.
[11] Giuseppe Massari et al. 2014. Combining application adaptivity and system-wide Resource Management on multi-core platforms. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*. IEEE, 26–33.
[12] Mina Niknafs et al. 2017. Two-phase interarrival time prediction for runtime resource management. In *Digital System Design (DSD)*. IEEE, 524–528.
[13] Mina Niknafs et al. 2017. Workload prediction for runtime resource management. In *Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. IEEE, 1–5.
[14] Charles Reiss et al. 2011. *Google cluster-usage traces: format + schema.* Technical Report. Google Inc., Mountain View, CA, USA.
[15] Amit Singh et al. 2016. Resource and throughput aware execution trace analysis for efficient run-time mapping on MPSoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 1 (2016), 72–85.
[16] Amit Kumar Singh et al. 2013. Mapping on multi/many-core systems: survey of current and emerging trends. In *DAC*. 1–10.
[17] Chu-Fu Wang et al. 2014. A prediction based energy conserving resources allocation scheme for cloud computing. In *Granular Computing*. IEEE, 320–324.