공학석사학위논문

사물인터넷 애플리케이션의 적응형 동적 스케줄링 기법

Adaptive Run-time Scheduling of IoT Applications

2019년 2월

서울대학교 대학원
컴퓨터공학부
박 강 규

사물인터넷 애플리케이션의 적응형 동적 스케줄링 기법

Adaptive Run-time Scheduling of IoT Applications

지도교수 하 순 회

이 논문을 공학석사 학위논문으로 제출함

2018년 11월

서울대학교 대학원

컴퓨터공학부

박 강 규

박강규의 공학석사 학위논문을 인준함

2018년 12월

위 원 장 이 광 근 (인)

부위원장 하 순 회 (인)

위 원 이 창 건 (인)

Abstract

An IoT system can be regarded as a distributed embedded system that is composed of heterogeneous smart devices with very different performance and functions. Also many IoT applications that have different resource requirements and real-time requirements will run concurrently in the IoT system. In addition, non-functional properties such as power consumption and device lifetime are considered important. Since an IoT application can be added or removed anytime and the device status may change at run-time, the system is unprecedentedly dynamic in its configuration, which brings up a challenging scheduling problem of IoT applications onto the smart devices. To tackle this problem, we propose a novel adaptive scheduling technique that consists of two scheduling techniques, incremental and global. An incremental heuristic method is proposed to provide fast responsiveness to dynamically changing configuration. During the steady-state operation, a GA-based method is applied to perform global rescheduling of IoT applications periodically to optimize a given objective function based on non-functional properties. We use the acceptance ratio of new applications and energy consumption as two performance metrics of the proposed scheduling method. The viability of the proposed approach is verified by extensive simulations with randomly generated scenarios.

Keywords : IoT system, task graph scheduling, adaptive scheduling, IoT application scheduling

Student Number : 2017-29611

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

An IoT (Internet-of-Things) system such as a smart home and smart office consists of a variety of embedded systems, called smart devices, that have different functionalities, computing power, and resource constraints. A smart device may be added to the system anytime or removed from the system due to various reasons such as power discharge, device failure, or network disconnection. Thus, the IoT system can be regarded as a distributed and highly heterogeneous computing system with dynamically varying configuration.

There could be many IoT applications running concurrently in an IoT system. An IoT application is defined as a sequence of tasks, invoked periodically or sporadically by an event arrival, each of which is executed on a smart device. It is usually associated with a real-time requirement such as period and deadline. We consider a dynamically configurable IoT system where an IoT application may be newly added or removed from the IoT system at runtime.

Even though most existent IoT platforms do not have such capability of accepting new applications and mapping and scheduling them to the constituent smart devices at runtime, we envisage that online scheduling of IoT applications on smart devices will be a new challenging problem in the future. As a closely related work, a service-oriented IoT platform has been proposed recently [1], where a smart device is abstracted as a service provider that can serve a predefined set of services and an IoT application is specified by a directed graph, called *service graph*. In a service graph, each node represents a service while the directed edges represent the dependencies between services. Since services and smart devices can be seen as *tasks* and *processing elements* (PEs) that can support only a specific set of tasks, respectively, this platform brings forth a similar scheduling problem. However no solution has been proposed. In this thesis, we define a more general adaptive

real-time scheduling problem for future IoT systems as scheduling of dependent tasks with real-time requirements on the heterogeneous non-preemptive[1] PEs whose availability may change dynamically.

To tackle this problem, we propose a novel adaptive scheduling technique that consists of two scheduling heuristics, *incremental* and *global*. Taking into account the characteristics of an IoT system, we deal with the scheduling problem for two separate cases: *dynamic* and *steady* states.

The *dynamic* state denotes the reconfiguration of things or services, such as the new arrival of IoT applications or removal of the devices. Such events do not happen frequently, but it is crucial to respond to the request properly and promptly. Therefore, on a new arrival of an application, we schedule the application onto available smart devices *incrementally* without affecting other applications using a fast heuristic. The fast heuristic is also invoked to re-schedule the system in case of device removals.

In the *steady* state, where many IoT applications are running concurrently on the IoT system, we perform *global* scheduling *periodically* to re-schedule the entire tasks to optimize some specific objective functions such as energy consumption. As the number of tasks and PEs increases, the scheduling complexity will increase proportionally or higher. In this work, we use a GA(genetic algorithm)-based heuristic. By setting a time limit as the termination condition, the GA-based heuristic produces a sub-optimal schedule within the given time limit.

The rest of this thesis is organized as follows. In the next section, we define the adaptive scheduling problem of the future IoT system with some illustrative examples. Then, in Section 3, we show how the IoT applications are converted to independent tasks and how the timing constraints are validated with respect to a mapping. Based on this analysis, Section 4 proposes an adaptive run-time scheduling technique, which is verified

[1]Typically, services provided by things are indivisible as they are related to certain sensors/actuators or physical processes. Moreover, even if they are pure software tasks, it is common to adopt non-preemptive scheduling in embedded systems [2, 3].

with extensive evaluations of many scenarios in Section 5. Related work and concluding remarks will be drawn in Section 6 and Section 7, respectively.

2. Problem Description

2.1 Target IoT system

In this section, we first present the target IoT system that we assume in this work. When a thing, or smart device, enter into the system, a set of tasks that it can serve is registered in the IoT middleware. A simple device such as sensor and actuator is likely to serve only one task, sensing, and actuation, respectively. On the other hand, some devices may serve more than one tasks; a camera, for instance, may take a picture, record a video, perform some image processing tasks, and so forth. There may exist powerful devices like robots that can serve various types of tasks in the assumed IoT system. Moreover, a virtual device that can perform cloud services such as mailing or any AI service can be added to the system. Note that there could be more than one devices that can perform a given task and a powerful device that can perform the task faster usually consumes more power. Moreover, in a powerful device with many physical cores inside, multiple tasks can be run concurrently. We assume that how many tasks can be run concurrently is known a priori. We also assume that fixed-priority non-preemptive scheduling policy is used in each device when multiple tasks are assigned, which is a natural choice for smart devices when a task uses a shared HW resource during its execution.

An IoT application usually consists of a sequence of tasks. For instance, let us suppose that a user defines a new IoT application when leaving home, commanding "if someone enters my house, take a picture, and send it to me." This application involves at least three dependent tasks, 1) a sensor to detect someone's entrance, 2) a camera to take a picture, and 3) a cloud service to send an e-mail. This IoT application can be specified by a script or an equivalent task graph as shown in Fig. 2.1, similarly to the service graph of [1] where a node represents a task, and an arc represents the dependency between tasks. Conditional execution of tasks can be specified by a conditional node that is depicted by

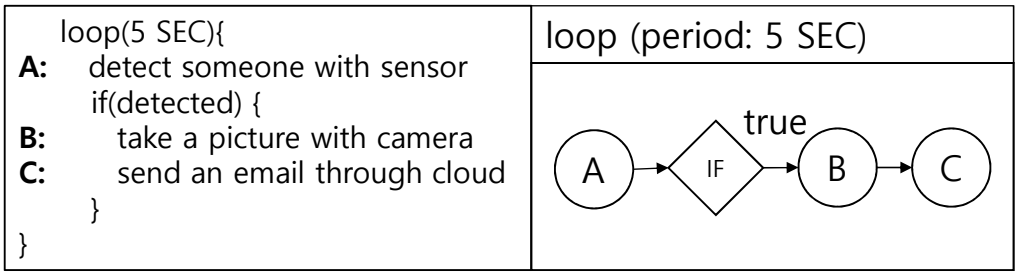


Figure 2.1: Script code description of an example IoT application and its equivalent task graph representation

a diamond symbol. In this work, we assume that the IoT applications are triggered periodically[1] and the period information is annotated to the task graph specification as shown in the figure. In this surveillance example, the task graph is invoked every five seconds, which is the implicit deadline constraint of the task.

In the target IoT system, there is a central server that monitors the status of the registered smart devices and manages the overall system status. It also orchestrates all IoT applications by mapping and scheduling the constituent tasks onto the smart devices. When a new application is requested by a user, it first checks whether the application can be accepted or not, regarding the timing constraint, then, determine how to map the constituent tasks to the devices, if acceptable. If any device fails to meet the real-time requirement of any assigned task after adding a new task, the application should be rejected, and the user is notified of the rejection. In the case of device failures, all the IoT applications that use the device will be re-scheduled by the server. In order to enable more sophisticated service control, it is allowed to set criticality levels of the IoT applications, letting high critical applications survive longer in case all applications are not schedulable.

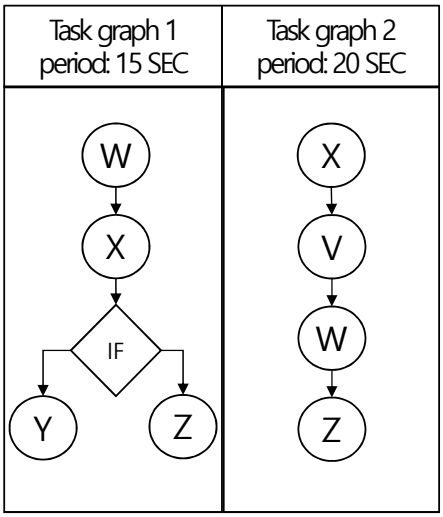
In addition to the timing requirements, it is also important to consider the non-

[1]Note that this can be extended to other triggering models without loss of generality. For instance, the period can be treated as the minimum time interval between two consecutive invocations in the event-triggered or sporadic task invocation model.

functional properties of the IoT system. For example, there may exist some devices with limited energy capacity, and it is desirable to lengthen the lifetime of them. On the other hand, saving energy consumption of the system is requested. In fact, the issue of energy consumption is emerging as an important issue in IoT systems [4]. Another issue is to increase the application acceptance ratio. We want to accommodate as many applications as possible in the system. In this thesis, we focus on maximizing the application acceptance ratio and minimizing energy consumption. However, it is worthwhile to mention that the proposed adaptive scheduling can be easily extended to other non-functional optimization criteria.

2.2 Motivational Example

In this subsection, we motivate the necessity of adaptive scheduling that enhances the application acceptance ratio. Let us suppose we have two IoT applications to be mapped on three things as shown in Fig. 2.2, whose mapping decisions are made as described in Table 2.1. In this particular mapping, *thing 1* and *thing 2* are considerably loaded in terms of utilization, while *thing 3* is relatively less occupied with fewer assigned tasks. In this configuration, let us suppose that another application that uses task *X* with a period of 10 seconds is requested. Then, this request cannot be accepted as both *thing 1* and *thing 2* do not have enough room for *X*. On the other hand, if task *W* of task graph 1 were mapped to thing 3, the new task *X* can be executed on *thing 1*. Note that such enhanced acceptance ratio possibly comes at the cost of increased energy consumption. In this case, task W consumes more energy on *thing 1* than on *thing 3*. To summarize, even with simple toy examples, it is not trivial to find a good mapping solution that co-optimizes the acceptance ratio and energy consumption at the same time with respect to the given timing constraints.



Thing	Supporting Tasks	Execution Time	Energy Consumption
Thing1	W	4	3
	X	3	7
	Y	7	5
Thing2	X	2	9
	Y	5	7
	Z	6	4
Thing3	W	3	5
	X	2	9
	V	6	2

Figure 2.2: A motivational example with two task graphs and three things

Table 2.1: A possible mapping choice of the motivational example in Fig. 2.2

Thing	Mapped tasks (Task graph)	Utilization	Energy Consumption
Thing 1	W(Graph1), X(Graph2), Y(Graph1)	0.88	15
Thing 2	X(Graph1), Z(Graph1), Z(Graph2)	0.83	17
Thing 3	V(Graph2), W(Graph2)	0.45	7

2.3 Problem Definition

The problem addressed in this thesis can be defined as follows:

1. Offline Input: All IoT applications that can be executed on the target system are given as input, each of which is specified as a task graph. All applications are assumed to be triggered periodically[2]. Thus each task graph is associated with an implicit timing constraint. That is, a task graph should be completed before the next instance is invoked. The constituent tasks are scheduled in a non-preemptive manner as stated in the previous section. As exemplified in Fig. 2.1 and 2.2, task graphs may have conditional statements (if-else clauses). In addition to the applications, all types of things, or PEs[3] that exist in the system are known in advance. As shown in Fig. 2.2, a set of information on the tasks that each PE can serve is given. To be more specific, execution time and energy consumption that a certain task takes on each PE are known.

2. Online Input: There are four types of reconfiguration events in the target IoT system, arrival/removal of applications and addition/removal of things. At runtime, we assume that one of the above reconfiguration events may occur anytime.

3. Output (Mapping/Scheduling): For all accepted applications, the mapping decisions, i.e., where to schedule each constituent task, have to be made. Once this decision is fixed, each thing schedules the assigned tasks according to the non-preemptive deadline monotonic policy. Whenever a new application arrives or a device is removed, the IoT system has to respond promptly to the event by updating the mapping decision of the tasks onto things while still satisfying the imposed timing constraints. Even without the runtime events' occurrences, the system

[2]Note that this can be extended to other triggering models without loss of generality. For instance, the period can be treated as the minimum time interval between two consecutive invocations in the event-triggered or sporadic task invocation model.

[3]A thing is a processing element. Without confusion, we use both terms interchangeably.

periodically adjusts the mapping/scheduling in order to enhance the optimization criteria gradually at runtime.

4. Objectives: Primarily, the optimization goal of the proposed mapping/scheduling technique is two-fold: to maximize the acceptance ratio and to minimize the overall energy consumption. Also, we try to keep the system as responsive as possible to the runtime reconfiguration events. In other words, the elapsed time for recalculating the mapping decision is to be minimized.

3. Schedulability Analysis

In this section, we present how to determine whether the task graphs' executions satisfy their timing constraints with respect to a given mapping. There have been several approaches proposed to analyze the worst-case latency of task graphs in distributed multi-processor systems, including analytical methods [5, 6, 7] and formal verification [8]. However, none of these is a viable solution for our problem, as the mapping decision needs to be made promptly online. Another approach to the real-time scheduling analysis of the task graph model is to transform the graph to independent periodic task sets [9, 10, 11, 12], which is adopted in the proposed technique. Even though it suffers from the risk of large over-estimation due to conservative transformation, it enables us to analyze the execution latency of task graphs fast at runtime. Since the task graph associated with an IoT application is usually a short chain of tasks or a simple DAG (directed acyclic graph), the risk of over-estimation is expected to be low. In the following subsection, we show how a task graph can be converted into a set of periodic independent tasks. Then, in Section 3.2, the schedulability analysis method we rely on is presented.

3.1 Transformation of a Task Graphs to Independent Tasks

We borrow the technique that transforms a task graph to a set of independent tasks from [9, 10]. In that approach, all the task dependencies are removed, and each task is assigned a new timing constraint that is defined by a tuple (*release offset*, *deadline*). These release offsets keep the relative ordering of the original dependent tasks. A transformed task τ on a thing θ is characterized by period T_τ, worst-case execution time $C_{\tau,\theta}$, release offset O_τ, and deadline D_τ. Now, the problem is how to assign O_τ and D_τ for each τ as T_τ and $C_{\tau,\theta}$ can be inherited from the original task graph without any modification.

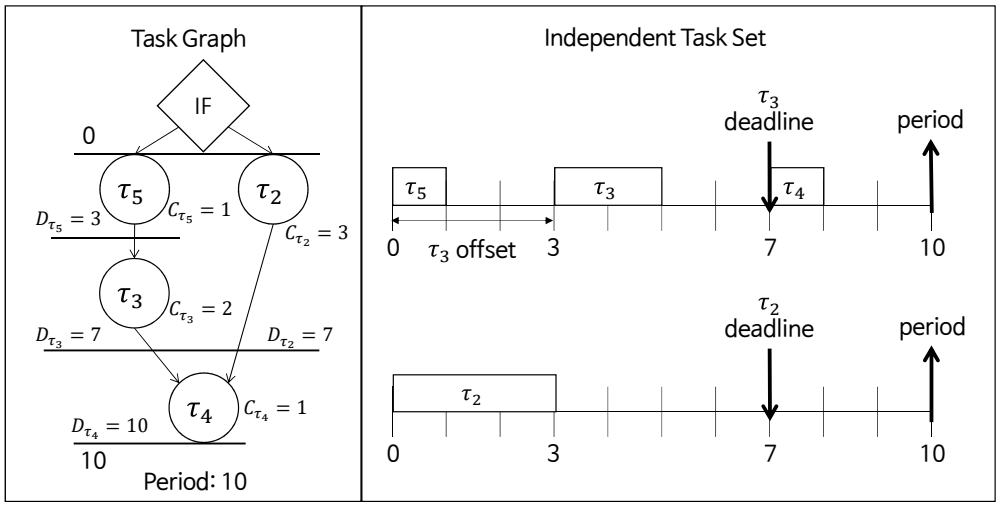


Figure 3.1: An example of the task graph transformation to an independent task set through the PURE method.

In this regard, they first find out the critical path, the sequence from the source to the destination task, the sum of whose execution time is the largest. Then, the amount of slack is quantified by calculating the difference between the period and the sum of execution times of the critical path. This slack is now distributed over the constituent tasks in that path in assigning the offset and deadline of each. Let us suppose that τ_i is the predecessor of τ_j in the critical path, the release offset of τ_j is the deadline of τ_i and the deadline is the sum of its release offset, execution time, and assigned slack. This procedure is repeated until all tasks in the graph are assigned a slack.

Depending on how the slack is distributed over the tasks in a path, there are two different approaches: PURE and NORM. In PURE, the slack is evenly distributed to all tasks, while it is distributed to the tasks proportionally to the execution times in NORM where tasks with longer execution time are assigned more slack. In the proposed technique, we used the PURE method as it has been reported that PURE exhibits better schedulability in non-preemptive scheduling [11].

Fig. 3.1 illustrates an example of the task graph transformation to independent tasks

through the PURE method. For ease of illustration, we assume that all the tasks have the same worst-case execution time on all things, i.e., $\forall\theta, C_{\tau,\theta} = C_\tau$. All the tasks in the graph are converted into independent tasks with the same period, 10, i.e., $\forall\tau, T_\tau = 10$. The critical path of the graph is $(\tau5 \rightarrow \tau3 \rightarrow \tau4)$, and the slack of the path is $10-(1+2+1)=6$. Thus, each task in the path is assigned a time slack of 2 time units. Since task $\tau5$ has no predecessor, $O_{\tau5}=0$, allocating a slack of 2 time units, its deadline is set as follows: $D_{\tau5} = O_{\tau5} + C_{\tau5} + 2 = 0+1+2 = 3$. This deadline also behaves as the release offset of the next task, $\tau3$, thus, $O_{\tau3} = 3$. With the slack of 2 time units, $D_{\tau3} = O_{\tau3} + C_{\tau3} + 2 = 3+2+2 = 7$. Similarly, we also have $O_{\tau4} = 7$ and $D_{\tau4} = 10$. Now, all tasks in the critical path are transformed. Then, the only remaining path in the graph is $(\tau2 \rightarrow \tau4)$. The same procedure is applied to the remaining tasks in the path. In this case, $\tau2$ is the only task remaining untransformed as $\tau4$ has already been handled. Considering the predecessor/successor tasks, we have $O_{\tau2} = 0$ and $D_{\tau2} = O_{\tau4} = 7$.

Note that the conditional execution information of the original task graph is obliterated by the transformation. For instance, it is trivial to tell $\tau5$ and $\tau2$ will never be invoked at the same iteration from the original specification. However, in the transformed independent task set, we assume that all tasks are instantiated in a single period. This is obviously a source of pessimism in the scheduling analysis of the proposed technique, but the inevitable cost to guarantee the schedulability at online mapping.

3.2 Schedulability Analysis

Now that all task graphs are transformed to an independent task set, the currently active applications can be represented as a set of n periodic tasks, $\Gamma = \{\tau_1, ..., \tau_n\}$. The utilization of τ is $U_\tau = C_\tau / T_\tau$. As a subset of Γ, $hp(\tau)$ denotes the set of tasks whose priority is higher than or equal to τ among the tasks that are mapped on the same thing as τ. Note that $hp(\tau)$ includes τ itself. Likewise, $lp(\tau)$ denotes the set of tasks with lower priority than τ among the tasks that are mapped on the same core. In the schedulability

analysis, we use the relative deadline D'_τ which is defined as $D_\tau - O_\tau$, instead of the absolute deadline D_τ. For instance the deadline of $\tau 3$ in Fig. 3.1 is changed to $D'_{\tau 3} = D_{\tau 3} - O_{\tau 3} = 7 - 3 = 4$ instead of 7, in order to make the analysis independent of release offset. We assume Γ is sorted in the ascending order of deadlines, i.e., $D'_{\tau_i} \leq D'_{\tau_j}, \forall i < j$.

For schedulability analysis, we rely on the time demand analysis (TDA) [3] to certify the deadline satisfaction of each task. TDA is based on the critical instant theorem [13], which says that the worst-case response time of a task is caused when all tasks are released at the same instant. Therefore, the worst-case latency can be calculated by taking into account the maximum amount of time a task τ on thing θ can be delayed by other tasks as formulated as follows:

$$\sum_{\tau_i \in hp(\tau)} \left\lceil \frac{D'_\tau}{T_{\tau_i}} \right\rceil C_{\tau_i,\theta} \leq D'_\tau \tag{3.1}$$

Note that the interference quantified in equation (3.1) only considers the higher priority tasks. As we deal with non-preemptive scheduling policy in the target IoT platform, a task may be blocked by a lower priority task as well. Thus, we modify the TDA formula in [14] as follows:

$$I_\tau + \sum_{\tau_i \in hp(\tau)} \left\lceil \frac{D'_\tau}{T_{\tau_i}} \right\rceil C_{\tau_i,\theta} \leq D'_\tau \tag{3.2}$$

where I_τ is the maximum interference from a lower priority task, $I_\tau = \max_{\tau_i \in lp(\tau)} (C_{\tau_i,\theta})$.

It is worthwhile to mention that the schedulability analysis presented in equation (3.2) is not tight, but very pessimistic, as it is deprived of execution dependencies of the original task graph. For instance, $\tau 5$ and $\tau 3$ will never interfere with each other even if they are mapped on the same thing. However, in the analysis presented in equations (3.1) and (3.2), the conflicts between them are taken into account. Such conservativeness could be alleviated by refining the definition of $hp(\tau)$ so that the dependent tasks are not included.

4. Proposed Mapping Technique

The IoT system exhibits two different aspects at the same time in its configuration. On the one hand, like other ordinary embedded systems, it operates the same functions, that are fixed a priori, repeatedly. In this sense, it is preferred that the configuration of the system, e.g., task-to-processor mapping, is optimized and fixed at design time. On the other hand, in the IoT system, users occasionally tend to install/remove functionalities or things even during the IoT system is actively working. In this case, it is important to react to the reconfiguration events promptly while still preserving the optimality of the system. Typically, the response time and the degree of optimization are in a trade-off relationship since it takes a considerable amount of time to explore the huge design space caused by the reconfiguration events.

Therefore, in this thesis, we propose a hybrid mapping technique that combines incremental and global methods. Essentially, the mapping decision is made online by a heuristic for *incremental* mapping. That is, each time a reconfiguration event occurs, the current mapping decision is *marginally* tweaked to cope with a new environment. As each reconfiguration event affects only a small portion of the system, a new viable configuration may not differ much from the current one. This is not a bad method as a way to reduce the mapping time (and enhance the responsiveness), but not guaranteeing the optimality. To compensate for this, a *global* re-mapping is performed when the number of accumulated reconfiguration events exceeds a certain threshold since the last re-mapping and a certain period elapses. In the two following subsections, the *incremental* and *global* mapping methods are presented, respectively.

4.1 Incremental Mapping

In favor of the reduced optimization time (responsiveness), the incremental mapping strategy tries to keep the current configuration as much as possible. As stated in Section 2.3, it has to deal with four different types of re-configuration events as follows:

1. Arrival of a new application: When a user requests a new IoT application, we have to decide whether the application can be accepted or not in the system. Each task of the accepted application (or task graph) needs to be mapped on a certain thing.

2. Removal of an application: When the user withdraws a certain application from the system, the corresponding tasks are simply removed from the things and the system statistics (utilization and energy) are updated accordingly. Here, no incremental mapping is needed.

3. Installation of a new thing: If a new thing is additionally deployed in the system, the information about the thing is updated, and no mapping needs to be performed.

4. Removal of a thing: A thing can be deleted from the system due to various causes including user request, energy shortage (out of battery), communication failures, and so on. In this case, the tasks that are mapped on the things need to be re-mapped on other active things. Thus, the incremental mapping is performed.

Algorithm 1 delineates how the incremental mapping responds to the runtime re-configuration events. First, it checks out the type of the event (lines 2-12). As mentioned earlier, no incremental mapping is performed on the removal of an application or the installation of a new thing. Thus, it simply returns after updating the information in the IoT server (lines 4-9). In case of the arrival of a new application (lines 2-3), the new application (task graph) is transformed to a task set Γ_{map} derived by the transformation procedure sketched in Section 3.1 as preparation of mapping. On the other hand, if the event is caused by the removal of a thing (lines 10-11), the tasks mapped on the removed

thing need to be re-mapped and so collected in the task set Γ_{map}.

Once the tasks to be remapped are determined as Γ_{map}, the mapping decision is made for each $\tau_{cur} \in \Gamma_{map}$ (lines 14-15). At line 16, a set of feasible things that can accommodate τ_{cur} is collected in $\Theta_{\tau_{cur}}$. Now, the algorithm needs to make a decision on which thing among the feasible ones is chosen for mapping. Recall in Section 2.3 that the primary objectives we aim to optimize are 1) to maximize the acceptance ratio and 2) to minimize energy consumption. Further, we do not have enough time to systematically explore the design space as the system needs to be responsive to the events. Therefore, we solely focus on the first one (acceptance ratio) in the incremental mapping. The rationale is as follows. If we largely keep the previous mapping decision and make only marginal changes for Γ_{map}, the degree of optimality of the changed mapping would not be far away from the previous one. Moreover, this temporary mapping can later be re-optimized by the global re-mapping, which will be presented in the next subsection.

Maximizing the acceptance ratio in mapping is a challenging task since it is not known which applications will arrive in the future. We can only be able to keep the mapping in a way that the future task graph would be more likely to fit in. In line with that, we devise three different quantitative indicators, with which the set of candidate things $\Theta_{\tau_{cur}}$ is sorted (line 17). That is, the earlier a thing appears in $\Theta_{\tau_{cur}}$, the higher acceptance ratio would be expected if τ_{cur} is mapped on that thing. So, we test the candidate things in $\Theta_{\tau_{cur}}$ one by one in order and verify the schedulability using TDA (line 20). If schedulable, τ_{cur} is decided to be mapped on the thing (lines 21-23), and the procedure is continued with the next task in Γ_{map}. If not schedulable, the next thing in $\Theta_{\tau_{cur}}$ is tested. When all tasks in Γ_{map} are fixed on a certain thing through the above procedure, the mapping is actually applied to the system, and the user is notified of the acceptance (lines 32-33). On the contrary, if no feasible things are found for any element of Γ_{map}, the requested IoT application is rejected (line 29).

In what follows, we discuss, in more detail, on what basis the candidate things are

chosen for mapping (line 17). We propose the following three policies and one comparison target:

- Sum Slack: In this approach, we assume that the acceptance ratio will be higher if we maintain the sum of slacks in all things as large as possible. That is, we try to maximize

$$\sum_{\theta \in \Theta} \left(1 - \sum_{m(\tau)=\theta} C_{\tau,\theta}/T_\tau \right) \tag{4.1}$$

  where Θ denotes the set of all things in the system and $m(\tau)$ is the thing that τ is mapped on. That is, the thing that makes equation (4.1) larger is preferred in the mapping. Actually, equation (4.1) is equivalent to minimize $C_{\tau_{cur},\theta}$ over $\theta \in \Theta_{\tau_{cur}}$.

- Relative Slack: Note that faster things (smaller C_i values for τ_i) are more likely to be chosen in the sum slack approach to enlarge the slack size. This might result in an unbalanced mapping, i.e., the tasks are likely mapped on a few fast things. Thus, we propose another approach, called relative slack, that encourages the tasks to be distried more evenly over the things. Here, we choose the thing θ that maximizes the following equation:

$$\frac{1 - \sum_{m(\tau)=\theta} C_{\tau,\theta}/T_\tau}{C_{\tau_{cur},\theta}} \tag{4.2}$$

  It simply chooses the thing that has the biggest slack relative to the task's execution time on that thing.

- Period Ratio: There is a sufficient condition for schedulability, called utilization bound. If the sum of utilization of the tasks mapped on a PE is bounded by a certain value, it is *guaranteed* that a feasible schedule is existent with respect to the scheduling policy. That is, for a thing θ, if the following condition is fulfilled,

$$U_\theta = \sum_{m(\tau)=\theta} \frac{C_{\tau,\theta}}{T_\tau} \leq U_{Bound} \tag{4.3}$$

where U_{Bound} is the utilization threshold for the given scheduling policy, all the tasks mapped on thing θ are schedulable. It is well-known that U_{Bound} is 1.0 and 0.69 for the preemptive earliest-deadline-first (EDF) and the preemptive rate-monotonic (RM) policies, respectively [13]. Further, it has been proven in [2] that the utilization bound of non-preemptive RM is proportional to $\frac{\min(T_\tau)}{\max(T_\tau)}, \forall \tau, m(\tau) = \theta$. This suggests that in non-preemptive scheduling it is beneficial to keep the difference between the maximum and minimum invocation periods of the tasks on a PE as small as possible. Inspired by this, we expect that the acceptance ratio would be better if we decide the mapping in a way that every PE has a relatively balanced invocation period. Therefore, we recalculate the period ratio of each thing with the following equation and choose the thing with the smallest difference from the previous period ratio of the thing:

$$\frac{\min(T_{\tau_{cur}}, \min_{m(\tau)=\theta}(T_\tau))}{\max(T_{\tau_{cur}}, \max_{m(\tau)=\theta}(T_\tau))} \tag{4.4}$$

- First Fit: As a comparison target, we also propose another mapping approach, called first fit. That is, a thing is chosen in an arbitrary order in $\Theta_{\tau_{cur}}$ at line 17, and anyone that meets the TDA constraint for the first time is chosen for mapping.

4.2 Global Re-mapping

Since the incremental heuristic method is a greedy approach that reflects dynamic changes one by one at runtime, the degree of optimality of the mapping is not as good as design time optimizations. In order to complement this drawback, we additionally apply a GA-based global re-mapping either periodically or when the number of runtime reconfiguration exceeds a certain threshold.

As the multi-processor mapping/scheduling is known to be NP-hard [15], it is common to use meta-heuristics such as GA in this class of optimization problems. Fig. 4.1

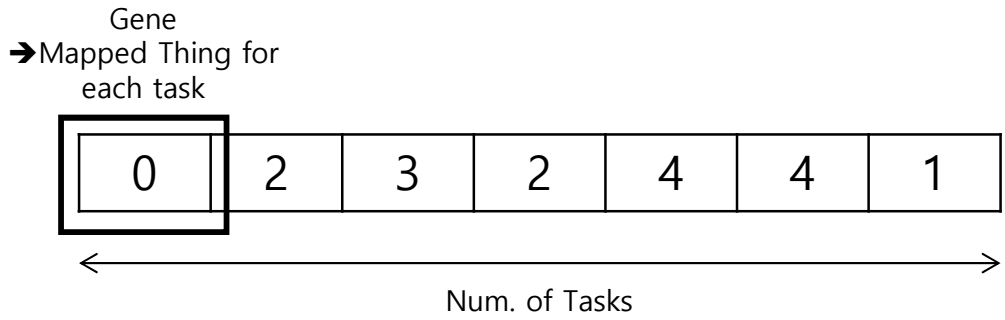


Figure 4.1: The chromosome structure of the GA-based global re-mapping solution.

illustrates the chromosome structure we used in the global re-mapping, where the slots for non-negative integer values are allocated as many as the number of tasks in the target system, i.e., $|\Gamma|$. Each value simply denotes task-to-thing mapping information. For instance, a solution shown in Fig. 4.1 stands for the mapping where τ_0, τ_1, and τ_2 are mapped on θ_0, θ_2, and θ_3, respectively. Initially, some individuals are randomly generated according to the presented gene structure. Then, as the generation goes by, they are undergone well-known genetic processes such as crossover or mutation. Each newly generated individual should be verified in terms of schedulability with inequality (3.2) in order to survive in the population. This procedure is repeated until it converges to a well-optimized mapping.

Note that, in the global re-mapping, all the applications including the ones that have already been in the system need to be completely re-mapped. In doing so, unlike the incremental mapping, we explicitly consider the minimization of energy consumption. It tries to minimize the sum of energy dissipated by each task on the mapped thing, which can be calculated as follows:

$$\sum_{\theta \in \Theta} \left(\sum_{\tau \in \Gamma, m(\tau)=\theta} E_{\tau,\theta} \right) \tag{4.5}$$

where $E_{\tau,\theta}$ denotes the energy consumption of τ on thing θ.

As the global re-mapping is a time-consuming optimization, it cannot be invoked on any reconfiguration event. Rather, this is periodically performed in the background by the IoT server, and the optimization result is occasionally reflected to the system when a more decent mapping solution than the current setup is found.

Algorithm 1 Incremental Mapping with Event e

▷ Classification of events
if e is 1) the arrival of a new application **then**
    $\Gamma_{map} \leftarrow$ transformed task set of the new application;
else if e is 2) the removal of an application **then**
    delete the tasks belonging to the deleted application;
    return Accepted;
else if e is 3) the installation of a new thing **then**
    register the new thing in the server;
    return Accepted;
else if e is 4) the removal of a thing **then**
    $\Gamma_{map} \leftarrow$ tasks mapped on the deleted thing;
end if
▷ Mapping
while $\Gamma_{map} \neq \phi$ **do** ▷ For all tasks
    $\tau_{cur} \leftarrow$ the first element of Γ_{map};
    $\Theta_{\tau_{cur}} \leftarrow$ set of things that support τ_{cur};
    sort $\Theta_{\tau_{cur}}$ according to equation (4.1) or (4.2) or (4.4);
    while $\Theta_{\tau_{cur}} \neq \phi$ **do** ▷ For all feasible things
        $\theta_{cur} \leftarrow$ the first element of $\Theta_{\tau_{cur}}$;
        if inequality (3.2) holds true **then** ▷ If schedulable
            map τ_{cur} on θ_{cur}; ▷ Mapping
            $\Gamma_{map} \leftarrow \Gamma_{map} - \{\tau_{cur}\}$;
            break;
        else ▷ If not schedulable
            $\Theta_{\tau_{cur}} \leftarrow \Theta_{\tau_{cur}} - \{\theta_{cur}\}$;
        end if
    end while
    if $\Theta_{\tau_{cur}} = \phi$ **then** ▷ If no feasible things found
        return Rejected;
    end if
end while
apply the new mapping; ▷ All tasks are mapped
return Accepted;

5. Experiments

5.1 Benchmarks

As we assume a dynamically reconfigurable IoT system, that is not matched with any existent IoT platform to the best of our knowledge; there are no open and public benchmarks for the proposed technique. Therefore, we built an in-house simulation environment in which multiple randomly generated dynamic IoT workloads are requested to run on multiple PEs (things).

A simulation environment can be characterized by some parameters that can be set by the user as summarized in Table 5.1. N is the number of PEs, and we also assume that there are N task types existing in the system. NT is the number of task types that a PE can serve, while NA is the number tasks in an IoT application. While generating the benchmarks, NT and NA are randomly chosen following a uniform distribution within the range of $[1,5]$ and $[2,10]$, respectively. The topologies of the task graphs are also randomly generated and the period and the execution time values are also randomly determined following a uniform distribution as detailed in Table 5.1. If a task can be run on multiple PEs, it is enforced that the task execution time on a more powerful PE is smaller than that on a weaker PE. Two configurations, $S1$ and $S2$, are used with different sizes, but with the same settings for the other parameters: $N = 20$ for $S1$ and $N = 50$ for $S2$.

The time values are scaled reflecting the typical characteristics of an IoT system where the period of an application is longer than the worst case latency of the associated task graph. The task types that a PE can serve are also randomly selected among the available task types.

For energy consumption, we impose two rules; 1) among multiple PEs that can serve a task, the PE that runs the task faster consumes more energy and 2) in a PE a task with longer execution time consumes more energy. The following simple formula is used to

Table 5.1: System Configuration Parameters

Symbol	Meaning	Value
N	Number of PEs = Number of task types	20 or 50
NT	Number of task types that a PE can serve	uni(1,5)
NA	Number of tasks that an application has	uni(2,10)
T_τ	Period of an application	uni(200,1000)
$C_{\tau,\theta}$	Task execution time of task τ on thing θ	uni(1,10)
P_θ	Relative computing capability of thing θ	uni(1,10)

decide the energy consumption of a task τ on thing θ: $E_{\tau,\theta} = k_1 \cdot P_\theta + k_2 \cdot C_{\tau,\theta} + k_3$ where k_1, k_2, and k_3 are configurable coefficients and P_θ is the relative computing capability of thing θ. In summary, the higher P_θ and $C_{\tau,\theta}$ are, the higher $E_{\tau,\theta}$ is.

5.2 Experiment 1 (Incremental Mapping)

In the first set of experiments, we compare the three variations of incremental mapping (*Sum Slack*, *Relative Slack*, and *Period Ratio*) with two comparison targets (*First Fit* and *Utilization Bound*). In *Utilization Bound*, Algorithm 1 is not performed but replaced with a random mapping with a simple sufficient condition check of schedulability, DM (deadline monotonic) version of equation (4.3). We generate 3000 random IoT applications (task graphs) and inject them to the system one by one. This has been repeated ten times and how many tasks are accepted on average by each mapping policy is observed ad illustrated in Fig. 5.1. Note that the acceptance decision is actually made for each application (graph). However, for more sophisticated analysis, we plot the number of accepted tasks.

Acceptance Ratio: As shown in Fig. 5.1, *Utilization Bound* shows the worst acceptance ratio among all policies, which shows the effectiveness of the TDA-based scheduling analysis. *Relative Slack* shows the best acceptance as it tends to distribute the tasks evenly over the things, making more room for future tasks. *Period Ratio* also exhibits decent acceptance ratios, and we believe that the conservativeness of TDA is consider-

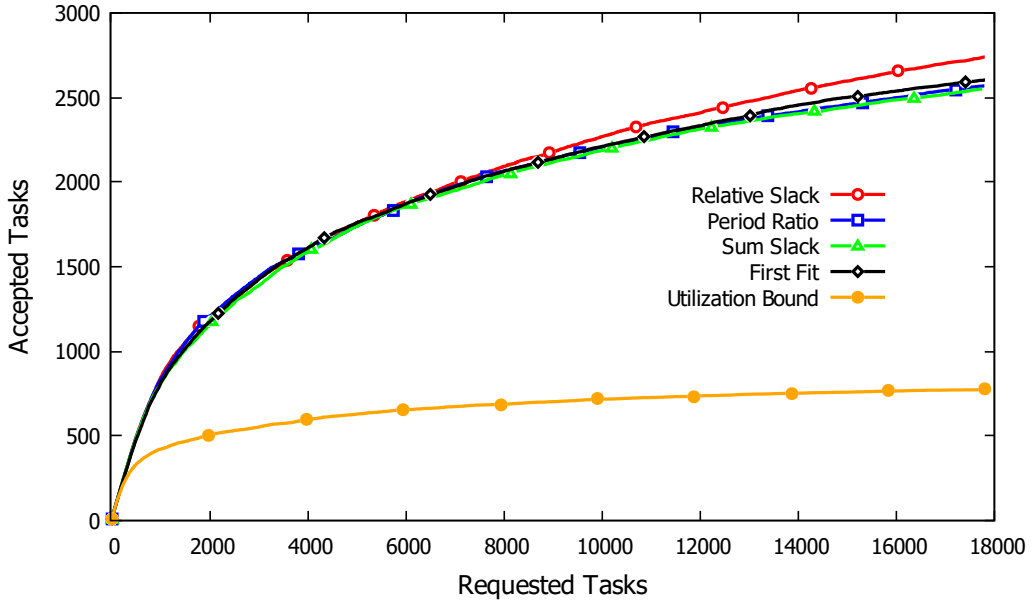


Figure 5.1: The ratio of accepted tasks over requested tasks in each mapping policy

ably alleviated by the policy. *First Fit* shows a comparable result when the number of requested tasks are not so many, but as the requested number increases, the acceptance ratio gap between the relative slack policy and the random policy becomes outstanding; about 10% difference can be observed when the number of requested tasks is 4000 in this figure.

Responsiveness: To verify the effectiveness of the incremental mapping as a runtime solution, we measure the elapsed time for the mapping decision and summarize in Table 5.2. The incremental mapping is performed in the following environment: Intel i7 CPU 3.40GHz machine with 16GB RAM, running Ubuntu 14.04. On average, all the proposed approaches take slightly more than 40 ms. The maximum mapping overhead is less than 105 ms in case of the relative slack policy, which we believe is fast enough to be used as a runtime decision maker.

Table 5.2: Elapsed Time For Incremental Mapping [milliseconds]

	Sum Slack	First Fit	Period Ratio	Relative Slack
Average	42.05	43.38	44.22	41.71
Max.	136.46	146.422	150.29	104.83

5.3 Experiment 2 (Global Re-mapping)

In the second experiment, we examine the effectiveness of the global re-mapping algorithm. For relative comparisons, we set *Relative Slack* as a baseline, which shows the best performance in the previous experiment. Like the previous experiment, we randomly generate 1000 IoT applications and inject them sequentially to the system and observe the total energy consumption in the system. Note that the requested task graph is originally mapped by the incremental mapping at the time of acceptance. Then, the global re-mapping is performed afterward. That is, in this experiment, the baseline *Incremental Mapping* denotes the proposed technique *without* the global re-mapping, while *Global Re-mapping* is the case that both two are applied together.

Energy Consumption: Fig. 5.2 compares the energy usage of the incremental method and the global re-mapping method with respect to task acceptance. The horizontal axis represents the number of tasks currently accepted by the entire system, and the vertical axis represents the sum of all the energy consumed by the PEs. In Fig. 5.2, it can be easily noticed that the energy consumption value is occasionally reduced because of the re-mapping. Since the global re-mapping is not performed every time an application is inserted, the degree of the optimality tends to decrease as the incremental mapping is continuously invoked by the reconfiguration events. Note that, in the *Incremental Mapping*, the effect of suboptimal mapping are accumulated; The difference between the two continues to be widened as the number of accepted tasks grows.

Optimization Time: The energy saving by the global re-mapping comes at the cost of increased optimization time. In order to verify the feasibility of the technique, we measure the optimization time taken by the global re-mapping. Fig. 5.3 compares the elapsed

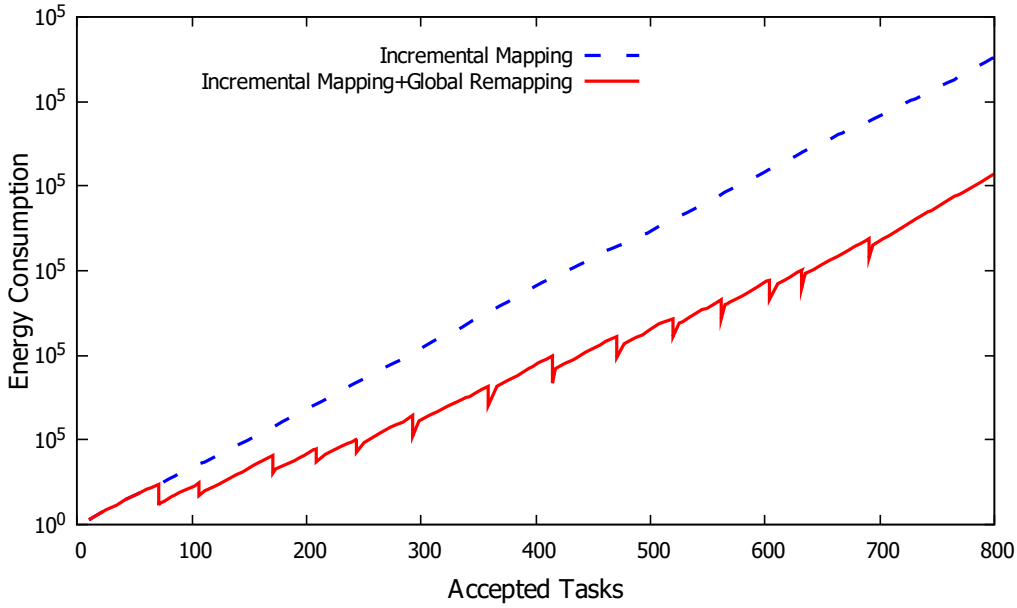


Figure 5.2: Comparison of total energy consumption between the incremental mapping and the global re-mapping

times of the incremental mapping and global re-mapping, where the horizontal and vertical axes represent the number of tasks that are accepted and the measured elapsed time in a logarithmic scale, respectively. The measurement is performed on the same computing environment as the first experiment. While both show the exponentially growing time cost, the time taken for the global re-mapping is not tolerable in the online mapping. This justifies why the proposed technique adopts the hybrid approach in mapping. In practice, we set the time limit for global re-mapping as the system size grows, sacrificing the optimality. Still, we could achieve significant energy saving from global re-mapping.

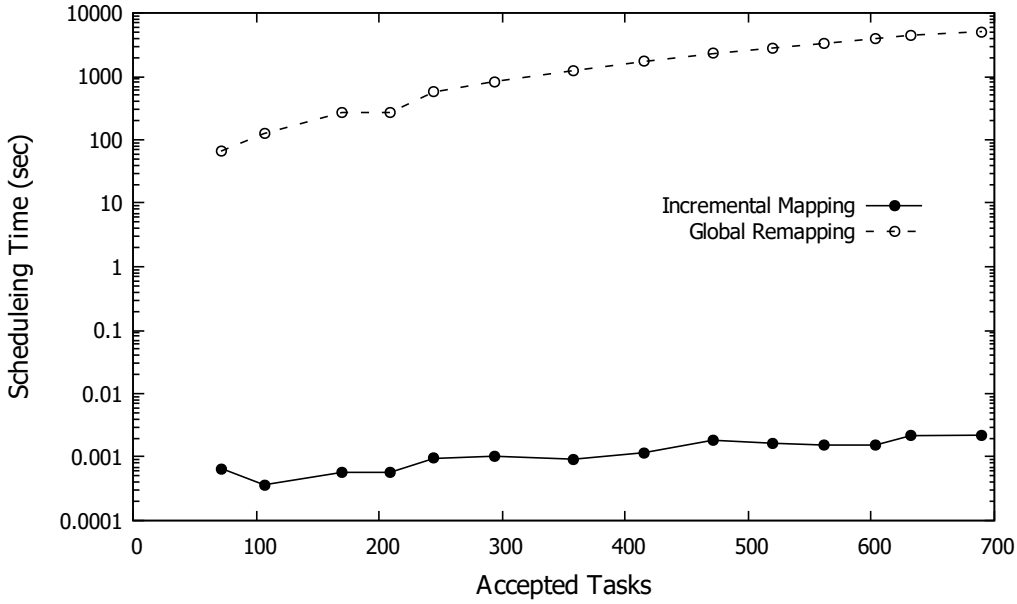


Figure 5.3: Elapsed time for the incremental mapping and global re-mapping

5.4 Experiment 3 (Sensitivity Analysis)

It is expected that the performance of the proposed mapping technique depends on the system configuration parameters. Among them, the period of application is likely the most important parameter since it affects the deadline of the constituent tasks and so the schedulability of the application. How the performance of the proposed incremental mapping is affected by this parameter, another set of experiments is devised. The upper bound of the application period, T_τ, is changed to 500 and 2000. The acceptance ratio variations are displayed in Fig. 5.4. We performed the experiment ten times and took the average. As shown in the figure, the performance gain from the proposed incremental mapping becomes more evident as the period increases. When the period is small, the number of tasks that a PE supports decreases since the deadline becomes tighter so that the relative slack policy gains little. So there is no difference in performance between the relative slack policy and the first fit policy. As the period becomes longer, more intelligent mapping decision gives more benefit. It is also noteworthy that the simple first fit policy

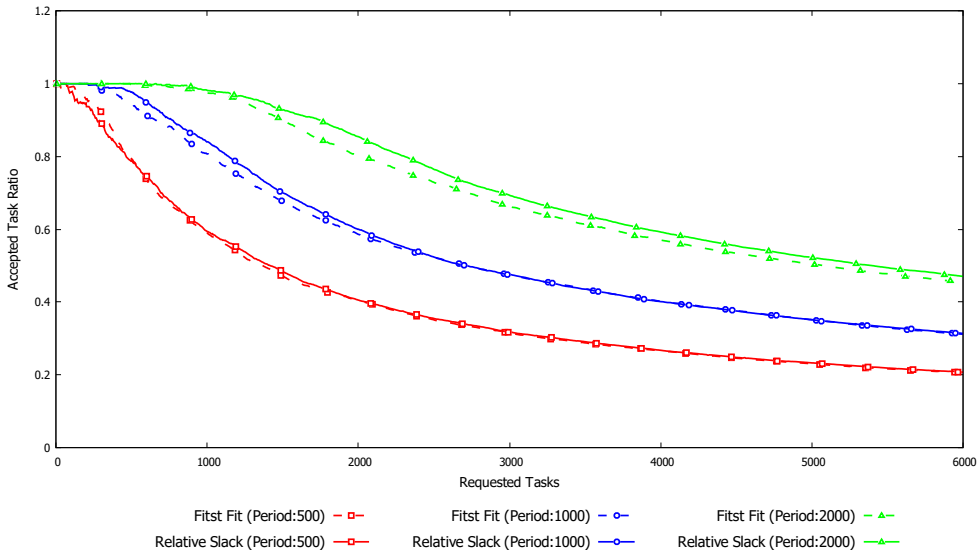


Figure 5.4: Acceptance ratio varying the period variances of the applications

may outperform when the number of applications is small. When the system is relatively idle, randomly generated dependencies of the task graphs affect the system in such an unexpected way.

6. Related Work

With increasing interest in the heterogeneous remote devices such as IoT and clouds, many researches have been devoted to the scheduling in the heterogeneous computing environments.

Lakra and Yadav [16] proposed a multi-objective scheduling algorithm for cloud computing, where two QoSes are co-optimized by means of non-dominated sorting. In their work, however, dependencies between the tasks were not considered at all, making the technique hardly applicable to the IoT use-cases. Furthermore, no real-time constraints were considered in the scheduling.

Srichandan et al. [17] proposed a multi-objective scheduling optimization for cloud computing based on genetic and bacterial foraging algorithms. While their problem is similar to the one proposed in this work in that they considered task dependencies and scheduling makespan and energy consumption are co-optimized, their solution is not applicable to the runtime mapping due to the prohibitively long optimization time.

Wang et al. [18] proposed a list scheduling algorithm for heterogeneous computing systems, where the scheduling workloads are specified in DAG. However, the merit of the technique is limited as they simply relied on a simple performance metric, such as average makespan ratio.

Recognizing an IoT system as a heterogeneous, complex, and dynamic system, Mabrouk et al. [19] addressed the problem of scheduling an incoming application onto devices. Similarly to ours, an application can be composed of multiple services and could be launched in the system at runtime. They proposed a heuristic that composes multiple services on-the-fly, co-optimizing the QoS and the energy consumption.

Unlike our model, however, they did not consider the service (task) dependencies and the real-time constraints. Narman et al. [20] proposed a scheduling method tailored

to the cloud-based IoT system, in which the requests are classified and mapped onto server groups. Like our approach, they proposed to periodically update the request-server mapping. However, they were not aware of the dependencies between services.

Li et al. [21] attempted to solve the scheduling problem in service-oriented IoT systems. They modeled the IoT system as a composition of application, network, and sensing layers, then, for each layer, an associated QoS is determined and optimized in the scheduling optimization. While the rapid service deployment was enabled for the suggested IoT environment, the dependencies between the IoT services were not considered.

Pham et al. [22] proposed a scheduling algorithm of DAG-specified applications for heterogeneous fog/cloud computing systems, considering makespan and cloud cost. During the scheduling, task prioritizing, node selection, and task reassignment are performed. While the task reassignment deals with the user-specified deadline constraint, however, no guarantee is provided in terms of schedulability.

To summarize, the mapping/scheduling optimization has been actively studied recently in the domain of heterogeneous distributed computing such as cloud and IoT. However, to the best of our knowledge, none of the existing solutions guarantees the schedulability of the IoT or cloud services within the given stringent timing-constraints.

In this regard, the proposed technique is distinguishable from the related work. We considered the dependencies between the IoT services (tasks) and performed the schedulability analysis using the task graph transformation and TDA. Based on this, a hybrid runtime mapping optimization technique has also been proposed.

7. Conclusion

In this thesis, we presented a new mapping/scheduling problem of IoT applications that can be added or removed anytime to/from the system on distributed heterogeneous things. We proposed a novel adaptive mapping/scheduling technique that consists of *Incremental Mapping* mapping and *Global Re-mapping*. The incremental mapping enables the fast responsiveness to dynamically changing configurations and still maintains the high application acceptance ratio. When the system is in a steady state, the GA-based global re-mapping is periodically performed to enhanced the optimality of the system, which is the energy consumption in this thesis. The effectiveness of the proposed method was verified by extensive simulations with randomly generated scenarios in terms of acceptance ratio, energy consumption, and responsiveness.

References

[1] HyunJae Lee, EunJin Jeong, Donghyun Kang, Jinmyeong Kim, and Soonhoi Ha. A novel service-oriented platform for the internet of things. In *Proceedings of the Seventh International Conference on the Internet of Things*, IoT '17, pages 18:1–18:8, New York, NY, USA, 2017. ACM.

[2] Moonju Park. Non-preemptive fixed priority scheduling of hard real-time periodic tasks. In Yong Shi, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *Computational Science – ICCS 2007*, pages 881–888, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[3] G. Brüggen, J. Chen, and W. Huang. Schedulability and optimization analysis for non-preemptive static priority scheduling based on task utilization and blocking factors. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 90–101, July 2015.

[4] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, Apr 2015.

[5] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using modular performance analysis: a case study. *International Journal on Software Tools for Technology Transfer*, 8(6):649–667, 2006.

[6] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis–the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

[7] Kai Lampka, Simon Perathoner, and Lothar Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 107–116. ACM, 2009.

[8] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal—a tool suite for automatic verification of real-time systems. In *International Hybrid Systems Workshop*, pages 232–243. Springer, 1995.

[9] H. I. Ali, B. Akesson, and L. M. Pinho. Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs. In *2015 23rd Euromicro*

International Conference on Parallel, Distributed, and Network-Based Processing, pages 701–710, March 2015.

[10] H. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. In *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*, pages 428–437, May 1993.

[11] Di Natale and Stankovic. Dynamic end-to-end guarantees in distributed real time systems. In *1994 Proceedings Real-Time Systems Symposium*, pages 216–227, Dec 1994.

[12] Mohamed Bamakhrama and Todor Stefanov. Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 195–204. ACM, 2011.

[13] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[14] R. I. Davis, A. Thekkilakattil, O. Gettings, R. Dobrin, and S. Punnekkat. Quantifying the exact sub-optimality of non-preemptive scheduling. In *2015 IEEE Real-Time Systems Symposium*, pages 96–106, Dec 2015.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[16] Atul Vikas Lakra and Dharmendra Kumar Yadav. Multi-objective tasks scheduling algorithm for cloud computing throughput optimization. *Procedia Computer Science*, 48:107 – 113, 2015. International Conference on Computer, Communication and Convergence (ICCC 2015).

[17] Sobhanayak Srichandan, Turuk Ashok Kumar, and Sahoo Bibhudatta. Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm. *Future Computing and Informatics Journal*, 2018.

[18] Guan Wang, Yuxin Wang, Hui Liu, and He Guo. Hsip: A novel task scheduling algorithm for heterogeneous computing. *Scientific Programming*, 2016:11, March 2016.

[19] Nebil Ben Mabrouk, Sandrine Beauche, Elena Kuznetsova, Nikolaos Georgantas, and Valérie Issarny. Qos-aware service composition in dynamic service oriented

environments. In Jean M. Bacon and Brian F. Cooper, editors, *Middleware 2009*, pages 123–142, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[20] Husnu S. Narman, Md. Shohrab Hossain, Mohammed Atiquzzaman, and Haiying Shen. Scheduling internet of things applications in cloud computing. *Annals of Telecommunications*, 72(1):79–93, Feb 2017.

[21] L. Li, S. Li, and S. Zhao. Qos-aware scheduling of services-oriented internet of things. *IEEE Transactions on Industrial Informatics*, 10(2):1497–1505, May 2014.

[22] Xuan-Qui Pham, Nguyen Doan Man, Nguyen Dao Tan Tri, Ngo Quang Thai, and Eui-Nam Huh. A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. *International Journal of Distributed Sensor Networks*, 13(11):1–16, 2017.

요 약

IoT 시스템은 매우 다른 성능과 기능을 가진 이기종 스마트 장치로 구성된 분산 임베디드 시스템이다. IoT 시스템에서 일반적으로 리소스 요구 사항과 실시간 요구 사항이 서로 다른 많은 IoT 애플리케이션들이 동시에 실행된다. 또한, 전력 소비 및 장치 수명과 같은 비 기능적 특성이 중요하게 고려된다. IoT 애플리케이션은 언제든지 추가되거나 제거 될 수 있으며 런타임에 디바이스 상태가 변경 될 수 있다. 이 같이 시스템은 동적 특성을 갖기 때문에 IoT 애플리케이션을 스마트 디바이스에 매핑/스케줄링 하는 것은 매우 까다로운 문제이다. 이 문제를 해결하기 위해 점진적 매핑 및 글로벌 재 매핑의 두 가지 스케줄링 기법으로 구성된 새로운 적응적 스케줄링 기법을 제안한다. 동적 환경 변화에 대한 빠른 응답을 제공하기 위해 점진적 매핑 방법을 제안하며, 정적 상태에서 비 기능적 특성에 기초하여 주어진 목적 함수를 최적화하기 위해 주기적으로 IoT 애플리케이션의 전체 태스크를 모두 다시 스케줄링 하는 유전 알고리즘 기반 글로벌 재 매핑 방법은 제안한다. 제안 된 스케줄링 방법의 두 가지 성능 지표로 애플리케이션 수용 비율 및 에너지 소비량을 사용하였으며, 성능 및 실용성은 무작위로 생성 된 시나리오를 사용한 시뮬레이션 환경을 통해 검증한다.

주요어 : 사물인터넷 시스템, 태스크 그래프 스케줄링, 동적 스케줄링, 사물인터넷 애플리케이션 스케줄링

학번 : 2017-29611

감사의 글

감사