Partitioning Real-Time Tasks with Replications on Multiprocessor Embedded Systems

Jian (Denny) Lin, *Member, IEEE,* Albert M. K. Cheng, *Senior Member, IEEE,* and Gokhan Gercek, *Member, IEEE*

***Abstract*—Executing computing tasks with replications is an essential choice to achieve fault-tolerance in designing real-time, embedded systems. A problem of maximizing the number of real-time tasks with replications running on a multiprocessor embedded system is discussed in this paper. The partitioning problem can be modeled as a variant of the Bin-Packing problem. In the original problem, it is known that the First-Fit (FF) method has a good worst-case performance bound of 4/3. Whether or not the same bound is achievable in the variant problem remains an open question. This paper closes the question by proving that the worst-case performance bound of using the FF method approaches to 2 but it never reaches it. Then, a tight bound of asymptotic worst-case performance is shown.**

***Index Terms*—Embedded Systems, Real-Time Systems, Approximation Algorithms, Fault Tolerance.**

I. Introduction

The employing of multiple processing units has become a standard framework that is widely adopted on embedded systems to accommodate the increasing computational demand of computing intensive applications. While these systems offer extra computing power, it also opens a new range of possibilities to improve the fault-robustness of running real-time embedded applications. A scheme to enhance reliability, called replication, is to execute multiple copies of critical tasks simultaneously on different processing units in resisting late completions, core failures or computing faults.

Task partitioning or task assignment has been identified as one of the most important problems in design of multiprocessor embedded systems [1], [2]. The algorithms proposed for this problem thus far can be broadly classified as partitioned or global. While partitioned scheduling restricts each task to run only on a designated processor, global scheduling allows a task to migrate from one processor to another. The partitioning problem with replications has been studied in [3], [4], where the authors seek to reduce the makespan or cumulative utilization of scheduling a set of independent tasks.

In a hard real-time system, independent tasks can be running simultaneously and each task must complete and meet all specified timing constraints such as deadlines. This paper focuses on the partitioning problem with replications where a hard real-time system may be overloaded for executing too many tasks. If a task does not complete by its deadline, not only its execution becomes useless, it may also cause other tasks to miss deadlines. A trade-off between schedulability and the number of tasks successfully completed must be considered. To solve the problem, an approximation algorithm in [5] is used to maximize the number of tasks successfully assigned. The solution is based on an extension of the Worst-Fit (WF) method of solving the Bin-Packing problem. A worst-case performance bound of 2 is proved in the paper. It has been informed in [6] that the FF method has a much better bound of 4/3 in solving the original Bin-Packing problem. What the bound is of using a FF-based solution in the partitioning problem with replications remains an open question [5]. In this paper, we answer the question by showing that the bound of 4/3 does not exist and a new, tight bound is presented.

Jian (Denny) Lin is an Assistant Professor and Gokhan Gercek is an Associate Professor, both with the Department of Management Information Systems, University of Houston - Clear Lake, Houston, TX 77058, USA (e-mail: LinJian@uhcl.edu, Gercek@uhcl.edu)

Albert M. K. Cheng is a Full Professor with the Department of Computer Science, University of Houston, Houston, TX 77204 USA (e-mail: cheng@cs.uh.edu)

II. System Model and Problem Definition

The system model used in this paper is similar to the one in [5] which consists of M identical processors $P = \{P_1, P_2, ..., P_M\}$, and N independent, periodic tasks $T = \{T_1, T_2, ..., T_N\}$. A periodic task is specified by its Worst Case Execution Time (WCET) *t* and period *p*. Each periodic task is an infinite sequence of instances where the task is executed. There is a time interval *p* between consecutive arrivals of the instances. An instance needs to be completed before the arrival of the next instance. A utilization $u = \frac{t}{p}$ associated with each task is equal to the fraction of time of using CPU to execute the task. We use a set of $U = \{u_1, u_2, ..., u_N\}$ to denote the utilizations of tasks in T. For identical multiprocessor systems, the task properties *t*, *p*, *u* are independent on the processor on which the task runs. It is assumed that the partitioned assignment is used in solving the problem.

After real-time tasks are assigned to a processor, they are scheduled by a real-time scheduling algorithm. The Earliest-Deadline-First (EDF) algorithm is an optimal, preemptive algorithm which can achieve 100% utilization of using CPU time on scheduling independent, periodic tasks on uniprocessor. Using the partitioned assignment, schedulability of the whole task set can be determined by checking the total utilization of each processor in the multiprocessor system. If none of the processors has its total utilization over 1.0, the tasks assigned on the system are schedulable. The task replication requirement is represented by an integer constant K. Each task has K replicas and each replica is an identical execution of running the task. When a task is assigned, K replicas of the task are assigned to K distinct processors. If the assignment of any replica overloads a processor, the assignment of the task is not accepted. In industry, $K = 3$ is a very effective technique used for fault tolerance. While $K = 2$, an execution of a task can have two versions, primary and backup. When a primary execution gets failed, the backup version can be used to provide a correct result.

Problem Definition. We explore the same problem as in [5]. Given a set of N periodic tasks T with its utilizations defined as a set of U, a requirement of K replications and a multiprocessor system P composed of M ($K \leq M$) identical processors, our goal is to find a mapping of the tasks to the processors such that the number of successfully assigned tasks is maximized.

The partitioning problem can be equivalently converted to a variant of a traditional NP-Hard problem, the Bin-Packing problem [5]. As more general and more difficult, our problem is also a NP-Hard problem. Due to the NP-Hardness, current techniques are not possible to solve the problem within polynomial time for large-sized instances. We target at using approximation algorithms to solve the problem. An approximation algorithm is an algorithm running in polynomial time to approximate the optimal solution within a bounded error. The following definitions are used.

Definition 1. $OPT(I)$: *An optimal solution (e.g., generated by an exhaustive searching) in terms of the number of tasks successfully assigned and a mapping between the tasks and processors for a problem instance* I.

Definition 2. $A(I)$: *a sub-optimal solution of the problem generated within polynomial time by an algorithm A for a problem instance* I.

We aim to find an algorithm A where $\frac{OPT(I)}{A(I)} \leq R$ for all I. The smallest R that holds for all problem instances is called a worst-case performance bound. In some cases we can improve the bound by focusing on asymptotic behavior and finding a constant c such that $OPT(I) \leq R \times A(I) - c$.

III. A Greedy Approximation Algorithm Based on the First-Fit Approach

We propose a greedy heuristic, named as $FFIK$ (First Fit Increasing with K-replication), extended from the FF method of solving the original Bin-Packing problem. In this method, a higher indexed processor is not used until all of its lower indexed processors do not have enough space to hold any more replicas. Given a set of tasks T, the tasks are sorted by their utilizations using a non-decreasing order. We assume that all of the replicas of these tasks can be successfully assigned by using an optimal algorithm OPT.

The $FFIK$ algorithm is explained as follows to solve the partitioning problem. It assigns a set of N tasks to the processors using at most N rounds. In each round, the smallest task T_i which has the smallest index in the unassigned task set is selected. The K replicas of the task are assigned at once to K available, distinct processors that have the smallest index. If the assignment does not overload any processor, T_i is removed from the unassigned task set. The $FFIK$ algorithm repeats until all tasks are assigned or no more tasks can be assigned. The number of tasks that have been successfully assigned is returned. The time complexity of the algorithm is $O(NlogN)$ because of the sorting process.

```
Input  : $T = \{T_1, T_2, ..., T_N\}$
         $U = \{u_1, u_2, ..., u_N\}$
         $P = \{P_1, P_2, ..., P_M\}$
         $K$
Output: A successful mapping that maximizes the
         number of the tasks to the processors with their
         K replicas
1 Sort the tasks in $T$ by their utilizations non-decreasingly.
2 while $|T| > 0$ do
3     Assign the first task in $T$ with its $K$ replicas at once
      to the $K$ available processors with the lowest index.
      If it cannot find such $K$ available processors, exit the
      while loop.
4     Update the total utilization of each processor;
5     Remove the task from $T$.
6 end
```
Algorithm 1: FFIK - First Fit Increasing with K Replications

In order to obtain the performance bound for the $FFIK$ algorithm, we define a worst-case scenario of all problem instances. By the nature of using the $FFIK$ algorithm, it always uses the first $\lfloor \frac{M}{K} \rfloor \times K$ processors to assign tasks. If the number of processors M is larger than $\lfloor \frac{M}{K} \rfloor \times K$, the remaining processors are not used. A worst-case scenario is defined when the last $K-1$ processors are not used by $FFIK$ to assign tasks.

Definition 3. *A worst-case scenario defines in the worst cases the number of processors that cannot be used by the $FFIK$ algorithm. A worst-case scenario exists when* $M - \lfloor \frac{M}{K} \rfloor \times K = K - 1$.

Lemma 1. *Given a problem instance, it is sufficient to prove the worst-case performance bound of the $FFIK$ algorithm by proving it in the worst-case scenario of the problem instance.*

Proof. Given a K, suppose that a problem instance $I^{'}$ has $M^{'}$ as $M^{'} - \lfloor \frac{M^{'}}{K} \rfloor \times K = K - 1$, and a problem instance I is any instance where $M - \lfloor \frac{M}{K} \rfloor \times K = 0, 1, ..., K - 2$. Also, $M^{'} > M$.

According to the property of $FFIK$, in both instances using $FFIK$ generates the same tasks assignment except of that in $I^{'}$ it has more unused processors. Thus, $FFIK(I) = FFIK(I^{'})$. Since the instance $I^{'}$ provides more space than I, $OPT(I^{'}) \geq OPT(I)$ and thus $\frac{OPT(I^{'})}{FFIK(I^{'})} \geq \frac{OPT(I)}{FFIK(I)}$. Hence, if we find a performance bound in the worst-case scenario of using $FFIK$, the bound is the worst-case performance bound for all problem instances. $\square$

Fig. 1 and Fig. 2 demonstrate an example for the worst-case scenario. Assume that there is a task set of 7 periodic tasks and their utilizations are {0.1, 0.2, 0.35, 0.4, 0.45, 0.5, 0.5}. The replication requirement is 3. In Fig. 1 it has 6 processors and $M - \lfloor \frac{M}{K} \rfloor \times K = 0$. In Fig. 2, 8 processors are used where

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2016.2620473, IEEE Embedded Systems Letters

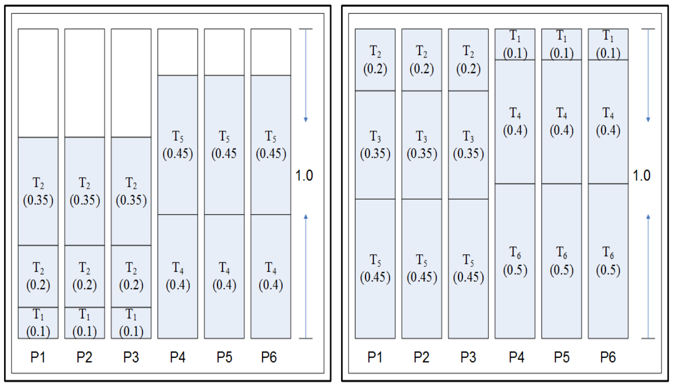

Fig. 1: A FFIK and An Optimal Assignment (6 Processors)

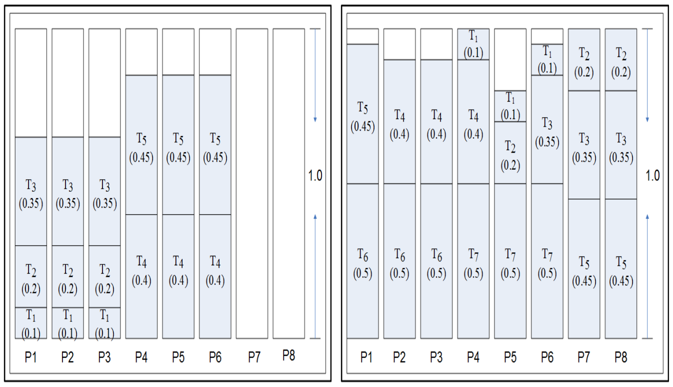

Fig. 2: A FFIK and An Optimal Assignment (8 Processors)

$M - \lfloor\frac{M}{K}\rfloor \times K = K - 1$. The case in Fig. 2 is a worst-case scenario. It can be seen that even with 2 more processors the $FFIK$'s assignments in both cases are the same. However, the optimal assignment in Fig. 2 has one more task assigned than the one in Fig. 1 because the case in Fig. 2 has more processors.

Now an instance in the worst-case scenario is considered where it has M processors and N periodic tasks. The replication requirement is K and $M - \lfloor\frac{M}{K}\rfloor \times K = K - 1$. In what follows we prove that the worst-case performance bound of using $FFIK$ approaches to 2, but it never reaches 2.

In the $\lfloor\frac{M}{K}\rfloor \times K$ processors used by the $FFIK$ algorithm to assign tasks, we divide these processors into $\lfloor\frac{M}{K}\rfloor$ groups and each group has exactly K processors, following the order from a lower index to a higher index of the processors. The $FFIK$ algorithm assigns replicas of a task at once to all of the processors within a group, with one replica to each processor. For example in Fig. 2, the 6 utilized processors are divided into 2 groups. Three and two tasks are assigned to the processors in the first and the second group, respectively.

We begin the proofs by assuming that T_i is the first task that gets failed to be assigned by using the $FFIK$ algorithm. We denote k_c as the number of tasks that have been assigned to the processors in the last group when T_i gets failed. Please note that $\lfloor\frac{M}{K}\rfloor$ and k_c are always integers.

Lemma 2. *For any problem instance I, let $d = OPT(I) - FFIK(I)$, then $d \leq \lfloor\frac{M}{K}\rfloor + k_c$*

Proof. By the property of the $FFIK$ algorithm, only the first $\lfloor\frac{M}{K}\rfloor$ groups of processors are used to assign tasks. Also, when a task is assigned by using $FFIK$, the K replicas stay on the K processors in a group. Because T_i cannot be assigned to the processors of any group, the maximum empty space of the multiprocessor system when T_i gets failed is

$$space_{max} < \lfloor\frac{M}{K}\rfloor \times K \times u_i + (K-1) \tag{1}$$

The $K-1$ is the space from the unused $K-1$ processors in the worst-case scenario. Since the tasks are sorted in a non-decreasing order based on their utilizations, all unassigned tasks at this point have the utilizations larger than or equal to u_i. Thus, assume that at most d more tasks can be assigned by an optimal solution.

$$\begin{aligned} d &\leq \frac{space_{max}}{u_i \times K} \\ d &< \frac{\lfloor\frac{M}{K}\rfloor \times K \times u_i + (K-1)}{u_i \times K} \\ d &< \lfloor\frac{M}{K}\rfloor + \frac{1}{u_i} - \frac{1}{u_i \times K} \\ d &< \lfloor\frac{M}{K}\rfloor + \frac{1}{u_i}(1 - \frac{1}{K}) \end{aligned} \tag{2}$$

Because $K \geq 1$ and u_i is not smaller than the utilization of any task that has been assigned to the processors in the last group, we have

$$\begin{aligned} d &< \lfloor\frac{M}{K}\rfloor + \frac{1}{u_i} \\ d &\leq \lfloor\frac{M}{K}\rfloor + k_c \end{aligned} \tag{3}$$

□

Lemma 3. *For any problem instance I, $\frac{OPT(I)}{FFIK(I)} < 2$*

Proof. In the assignment of using $FFIK$, let k_1 be the number of tasks assigned to the processors in the first group, and k_2 be the number of tasks assigned to the processors in the second group, and so on. Because smaller-sized tasks are assigned to groups with lower-indexed processors, we have $k_c \leq k_{c-1}, ..., \leq k_1$ and thus $FFIK(I) \geq \lfloor\frac{M}{K}\rfloor \times k_c$. Then,

$$\begin{aligned} \frac{OPT(I)}{FFIK(I)} &= 1 + \frac{d}{FFIK(I)} \\ &\leq 1 + \frac{\lfloor\frac{M}{K}\rfloor + k_c}{\lfloor\frac{M}{K}\rfloor \times k_c} \\ &\leq 1 + \frac{1}{k_c} + \frac{1}{\lfloor\frac{M}{K}\rfloor} \end{aligned} \tag{4}$$

The above inequality is a function dependent on k_c and $\lfloor\frac{M}{K}\rfloor$. There are three possibilities of the values used by k_c and $\lfloor\frac{M}{K}\rfloor$ that may cause the ratio between $OPT(I)$ and $FFIK(I)$ larger than or equal to 2. We analyze these three cases and show that in every case the ratio is bounded by 2. However, it never reaches 2.

Case 1: $k_c = 1$ and $\lfloor\frac{M}{K}\rfloor$ is any value. If $k_c = 1$, it means $u_i > 0.5$ and in fact all of the remaining tasks have their utilizations larger than 0.5. Otherwise, the replicas of T_i can be assigned to the processors in the last group or the rules of using the $FFIK$ algorithm are violated. Since all of the remaining tasks have their utilizations larger than 0.5 and T_i gets failed, each replica of the unassigned tasks and the task already assigned to the last group's processors by $FFIK$ must stay on a distinct processor. Considering the worst-case scenario that $M - \lfloor \frac{M}{K} \rfloor \times K = K - 1$, at most $\lfloor \frac{M}{K} \rfloor - 1$ additional tasks can be assigned in the optimal solution. Therefore

$$\frac{OPT(I)}{FFIK(I)} \leq 1 + \frac{\lfloor \frac{M}{K} \rfloor - 1}{FFIK(I)} \tag{5}$$

Because $FFIK(I) \geq \lfloor \frac{M}{K} \rfloor$,

$$\begin{aligned} \frac{OPT(I)}{FFIK(I)} &\leq 1 + \frac{\lfloor \frac{M}{K} \rfloor - 1}{\lfloor \frac{M}{K} \rfloor} \\ \frac{OPT(I)}{FFIK(I)} &< 2 \end{aligned} \tag{6}$$

Case 2: $\lfloor \frac{M}{K} \rfloor = 1$ and k_c is any value. If $\lfloor \frac{M}{K} \rfloor = 1$, it means that in the $FFIK$'s solution only one group of K processors is used and $k_1 = k_c = FFIK(I)$. Since the k_c tasks assigned by using the $FFIK$ algorithm are the smallest ones, in an optimal solution, on any individual processor the number of replicas assigned is not larger than k_c. Because $M < 2 \times K$, in the optimal solution the number of additional tasks assigned to the M processors must be smaller than k_c. Hence,

$$\frac{OPT(I)}{FFIK(I)} < 2 \tag{7}$$

Case 3: $k_c = 2$ and $\lfloor \frac{M}{K} \rfloor = 2$. In this case, the bound calculated by (4) is equal to or less than 2. We show that it is not possible to be 2. Since $\lfloor \frac{M}{K} \rfloor = 2$, there are only two groups of processors to be used by $FFIK$ to assign tasks and thus the worst-case scenario in this case is $M = 3K - 1$. Also, because $k_c = k_2 = 2 \leq k_1$, $FFIK(I) \geq 4$. In order to have $\frac{OPT(I)}{FFIK(I)} = 2$, it needs $d \geq 4$. Since T_i cannot be assigned by $FFIK$, every task in the unassigned set has a utilization larger than $\frac{1}{3}$ and each one cannot stay together on any processor with the two tasks assigned to the second group's processors. To assign additional 4 tasks, in total $4K$ more replicas need to be assigned. We create a set of these $4K$ replicas and the $2K$ replicas assigned by $FFIK$ to the second group's processors. In this set of $6K$ replicas of the tasks, at least $3K$ processors are needed to assign them which contradicts $M = 3K - 1$. Hence,

$$\frac{OPT(I)}{FFIK(I)} < 2 \tag{8}$$

□

Because the bound of 2 is not reachable, we check the next slightly better bound which is using the asymptotic worst-case performance bound: $OPT(I) \leq 2 \times FFIK(I) - 1$. We show this bound tight by constructing a problem instance.

Theorem 1. *The asymptotic worst-case performance bound of the $FFIK$ algorithm is $OPT(I) \leq 2 \times FFIK(I) - 1$ and the bound is tight.*

Proof. Given a problem instance I such that the replication requirement is K, the number of processors $M = 2K - 1$, the number of tasks $N = 2K - 1$, and each task's utilization is $\frac{1.0}{K}$. By using the $FFIK$ algorithm, the first K processors are used and each processor of them has exactly K replicas of some tasks assigned. In this solution, $FFIK(I) = K$. The optimal solution is generated by using the algorithm $WFIK$ [5] which is based on the Worst-Fit approach. In the optimal solution, all of the tasks can be assigned. Hence, in this case, $OPT(I) = 2 \times FFIK(I) - 1$. □

IV. Discussions and Future Works

An indirect conclusion can be reached from the proof of Lemma 3 that the worst value of the bound appear when the number of processor is small and/or the tasks' sizes are relatively large. Fig. 3 shows the bound calculated by using (4) when the largest utilization in tasks is 10% ($k_c \geq 10$) and $K = 2$. It can be seen that the bound improves and converges quickly when the number of processors increases. The worst-case bound we study in this paper provides a predictable, lower bound to the performance of the solution. Due to the page limit, the average performance performed in a large set of cases is not presented. We will extend this letter to be a long, full paper, including the discussion of the worst and average performance analysis for the $FFIK$ and $WFIK$ algorithms.

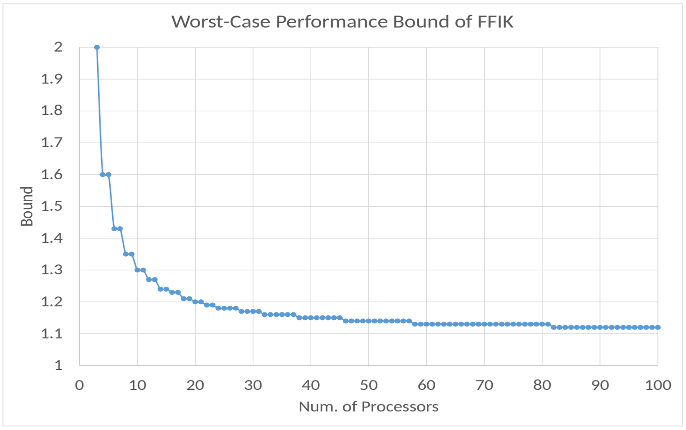

Fig. 3: Worst-Case Bound ($K = 2$ and $k_c \geq 10$)

References

[1] J. Lee, K. G. Shin, I. Shin, and A. Easwaran, *Composition of Schedulability Analyses for Real-Time Multiprocessor Systems*, IEEE Transaction on Computers, Vol. 64, No. 4, April 2015.
[2] A. K. Singh, M. Shafique, A. Kumar and J. Henkel, *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, Vol. 35, No. 1, Janurary 2016.
[3] S. Gopalakrishnan and M. Caccamo, *Task Partitioning with Replication upon Heterogeneous Multiprocessor Systems*, Proc. IEEE Real-Time and Embedded Technology and Application Symposium, May 2006.
[4] J. Chen, C. Yang, T. Kuo and S. Tseng, *Real-Time Task Replication for Fault Tolerance in Identical Multiprocessor Systems*, Proc. IEEE Real-Time and Embedded Technology and Application Symposium, 2006.
[5] J. Lin and A. K. Cheng, *Real-time Task Assignment with Replication on Multiprocessor Platforms*, Proc. IEEE 15th International Conference on Parallel and Distributed Systems, December 2009.
[6] E.G. Coffman, J. Y-T. Leung and D.W. Ting, *Bin Packing: Maximizing the Number of Pieces Packed*, Acta Informatica Vol. 9, 1978.