Real-Time Task Replication for Fault Tolerance in Identical Multiprocessor Systems

Jian-Jia Chen, Chuan-Yue Yang, Tei-Wei Kuo
Department of Computer Science and Information Engineering
Graduate Institute of Networking and Multimedia
National Taiwan University, Taiwan.
Email: \{r90079@, r92032, ktw\}@csie.ntu.edu.tw

Shau-Yin Tseng
SoC Technology Center
Industrial Technology Research Institute
Taiwan
Email: tseng@itri.org.tw

Abstract

Multiprocessor platforms have been widely adopted in both embedded and server systems. In addition to the performance improvement, multiprocessor systems could have the flexibility in tolerating processor failures via task replication. This paper considers the replication of periodic hard real-time tasks in identical multiprocessor environments. Each task is replicated on \( K \) distinct processors, where \( K \) is a user-determined integer for fault tolerance to improve system reliability. When the objective is to minimize the maximum utilization in a system with a specified number of processors, we present a greedy algorithm with a \( 2 \)-approximation ratio, and a polynomial-time approximation scheme is developed. For the minimization of the number of processors required to derive feasible schedules with task replication, we develop greedy algorithms with a \( 2 \)-approximation ratio and an asymptotic polynomial-time approximation scheme.

Keywords: Real-Time Task Scheduling, Fault Tolerance, Task Replication, Multiprocessor Systems.

1 Introduction

Multiprocessor platforms have been widely adopted for many different applications in embedded systems and server systems, and they might become even more popular while many chip makers, such as Intel and AMD, are releasing multi-core chips. The adopting of multiprocessor platforms could improve the system performance and accommodate the growing demand of computing power and the variety of application functionality. Beside those benefits, the deployment of multiprocessor platforms also has the advantages in fault tolerance by replicating computation over processors.

Multiprocessor job/task scheduling without job/task replication has been widely explored in the literature. For the minimization of the maximum utilization of processors without task replication, the problem can be mapped to the traditional makespan problem (to minimize the maximum completion time) for parallel machines. It is done by transforming the utilization of a task as the processing time of a corresponding job. Such a strategy has been adopted in recent studies for multiprocessor real-time systems, e.g., [4, 5, 10, 12]. For the makespan problem, Graham [13] shows a \( 2 \)-approximation algorithm for identical machines by scheduling jobs one by one in an arbitrary order and assigning each job to the machine with the least amount of work so far. If jobs are assigned in a decreasing order of their processing time, the resulting algorithm is a \( \frac{3}{4} \)-approximation algorithm [13]. For systems with heterogeneous machines, Gonzales et al. [11] show that the largest-processing-time-first rule has a \( 1.5 \)-approximation ratio when each machine has a specified speed for job executions. If the machines are unrelated, Lenstra, Shmoys, and Tardos [17] develop a \( 2 \)-approximation algorithm based on Integer Linear Programming (ILP) with parametric pruning. In addition to the combinatorial approaches, researchers also develop approximation algorithms that trade the approximation ratio with the time and space complexity. For unrelated parallel machines, Horowitz and Sahni [15] develop an FPTAS for the makespan problem for systems with a constant number of processors. Hochbaum and Shmoys [14] develop a PTAS for machines with specified speeds.

This paper considers the replication of periodic hard real-time tasks in identical multiprocessor environments. Each task is replicated on \( K \) distinct processors, where \( K \) is a user-determined integer for fault tolerance. We consider two different scheduling problems for task replication: (1) the minimization problem of the maximum utilization of a set of processors and

---

\(^{1}\) An \( \alpha \)-approximation algorithm derives a solution with at most \( \alpha \) times of the objective of an optimal solution for a minimization problem of any input instance.

\(^{2}\) When an algorithm has a \((1 + \delta)\)-approximation ratio for a user-specified positive parameter \( \delta \), it is an approximation scheme [24]. If the complexity of the algorithm is polynomial by treating \( \delta \) as the input, it is referred to as a fully polynomial-time approximation scheme (FPTAS). If the complexity of the algorithm is polynomial by treating \( \delta \) as a constant, it is referred to as a polynomial-time approximation scheme (PTAS). Please refer to the textbook by Vijay [24] for the definitions.
(2) the minimization problem of the number of processors required to derive feasible schedules (i.e., a synthesis problem). Both of the problems are \( \text{NP-hard} \) in a strong sense even when no replication of tasks is required, i.e., \( K = 1 \). For the minimization of the number of processors required to derive feasible schedules, the problem can be mapped to the traditional bin packing problem if no replication is required, where a bin packing problem is to minimize the number of bins used to pack a given set of items. When the earliest-deadline-first (EDF) algorithm [19] is adopted, bin-packing-based 2-approximation algorithms exist for the minimization problem, where each bin is of a size 100% in the task utilization [24, §9]. As shown in [24, §9], the bin packing problem does not admit any PTAS. Some asymptotic approximation algorithms are proposed in [9, 16]. In the past, most multiprocessor real-time scheduling results did not consider task replication in the exploring of real-time task schedulability, e.g., [1, 3–5, 10, 22].

In the past decade, fault tolerance issues are explored for real-time systems in the contexts of uniprocessor environments, e.g., [2], and multiprocessor environments, e.g., [6, 12, 18]. While some researchers consider transient faults (recoverable on the same processor), e.g., [2, 18], some others explored task replication to resolve processor (permanent) faults, e.g., [6, 12]. Task replication issues for real-time tasks with precedence constraints are also explored in [7, 8, 23], where several heuristic algorithms are proposed. In this paper, we are interested in the replication of periodic hard real-time tasks in identical multiprocessor environments. The results closely related to this paper are those in [6, 12]. In particular, Gopalakrishnan and Caccamo [12] explored the minimization problem of the maximum utilization of processors with a constant number of heterogeneous multiprocessors. The FPTAS in [12] is extended from the FPTAS without fault tolerance in [15]. Bertossi et al. [6] explored fixed-priority-driven scheduling with the rate monotonic scheduling policy and task replication for two replicas.

The main contribution of this paper is on the development of approximation algorithms for identical multiprocessor systems with task replication, where there is an arbitrary number of processors. The time complexity of our algorithms is polynomial in the number of processors, and the number of replicas is an arbitrary number, while the time complexity of the FPTAS in [12] is exponential in the number of processors, and the number of replicas is 2 in [6]. When the objective is to minimize the maximum utilization with a specified number of processors, we present an approximation algorithm with a 2-approximation ratio. The algorithm is then extended to a polynomial-time approximation scheme. For the minimization of the number of processors to derive feasible schedules, we show that a simple modification of the first-fit, best-fit, and worst-fit strategies can also have a 2-approximation ratio. We then develop an asymptotic polynomial-time approximation scheme which guarantees that the number of processors needed to derive feasible schedules is at most \((1 + \delta)\) times the number of processors of an optimal solution plus \( K \), where \( K \) is a user-determined integer for fault tolerance, and \( \delta \) is a positive parameter. The proposed (asymptotic) polynomial-time approximation schemes allow the system designer to trade the optimality of the derived solution with the analysis time.

The rest of this paper is organized as follows: Section 2 defines the system models and the problems under considerations. Section 3 explores the minimization of the maximum utilization of processors. Section 4 focuses on the minimization of the number of processors. Section 5 concludes this work.

2 Problem Definitions

Task Model Tasks under discussions in this paper are periodic and independent in executions. A periodic task is an infinite sequence of task instances, referred to as jobs, where each job of a task comes in a regular period [19, 20]. Each task \( \tau_i \) is associated with its worst-case execution time (denoted as \( e_i \)) and its period (denoted as \( p_i \)), where the relative deadline is equal to its period. The task model considered in this paper follows that in [12] except that we explore identical multiprocessor systems. That is, a task has the same execution time on each processor.

It has been shown that the earliest-deadline-first (EDF) schedule for a set of periodic real-time tasks on a processor is optimal [19]. A task set is schedulable on a processor if and only if its total utilization is no more than 100%, where the utilization of a task is defined as its worst-case execution time divided by its period. Throughout this paper, we assume that EDF schedule is applied to each processor. For task \( \tau_i \), the utilization of \( \tau_i \) is denoted as \( u_i \), in which \( u_i = \frac{e_i}{p_i} \). Each task under considerations in this paper has utilization no more than 100% since a task could not complete in time with utilization more than 100%.

Task Replication To ensure resilience to faults, each task is assigned on \( K \) distinct processors, where \( K \) is a user-specified parameter for fault tolerance. We refer to the replication factor as \( K \). These \( K \) replicas of a task are executed independently, and the output data of these \( K \) replicas might be sent to a data sink to determine the desired data. The data sink might use any scheme to select/determine the output, such as voting or averaging. For example, when \( K = 3 \), the triple modular redundancy [21] has been a very effective technique for improving system reliability for many years. If the replication factor is 2, we can treat the original task as the primary version and the replication as a backup version. If any fault is detected in the primary version, the results from the backup version are selected as the output. The task replication model here is the same as that in [12]. Please refer to [12] for details of schedules with task replications.

Problem Definitions We explore task replication for a set \( T \) of \( N \) periodic real-time tasks. Two different problems are considered in this paper. When the number \( M \) of processors is given, our objective is to derive a feasible task replication partition with the minimization of the maximum utilization of these \( M \) processors. A task replication partition is to map the real-time tasks in \( T \) to the \( M \) processors. A task replication partition is feasible if each task is assigned to exactly \( K \) distinct processors. Since a task set is schedulable if and only if its total
utilization is no more than 100%, the minimization problem of the maximum utilization is closely related to the schedulability problem of periodic real-time tasks on task replication partition. The utilization bound minimization approaches for multiprocessor systems are also studied in [4, 5, 12]. We refer to the above optimization problem as the multiprocessor task replication partition problem. Another problem, denoted as the multiprocessor task replication packing problem, is to minimize the number of processors required to derive a feasible task replication partition of $T$ without violating the timing constraints of tasks. A feasible task replication partition is said a feasible task replication packing if the total utilization on each allocated processor is no more than 100%. A feasible task replication partition is optimal if the maximum utilization of processors is the minimum among all feasible task replication partitions.

The two problems considered in this paper are equivalent in terms of decision versions. When $K = 1$, their decision versions are as the same as that of the bin packing problem. Since the bin packing problem is $\mathcal{NP}$-hard in a strong sense, the multiprocessor task replication partition problem and the multiprocessor task replication packing problem are both $\mathcal{NP}$-hard in a strong sense. Due to the $\mathcal{NP}$-hardness of the problems, we focus the study on polynomial-time approximation algorithms with worst-case guarantees. An $\alpha$-approximation algorithm derives a solution with at most $\alpha$ times of the objective of an optimal solution for the minimization problem of any input instance, where $\alpha$ is referred to as the approximation ratio of the algorithm. However, the following theorem shows the non-approximability of the multiprocessor task replication packing problem.

**Theorem 1** For any positive constant $\delta$, there does not exist any polynomial-time $(1.5 - \delta)$-approximation of the multiprocessor task replication packing problem, unless $\mathcal{P} = \mathcal{NP}$.

**Proof.** It follows directly from the hardness of the bin packing problem [24, §9, Theorem 9.2].

This paper provides combinatorial approximation algorithms for the studied problems with low time complexity. Moreover, polynomial-time approximation schemes (PTAS) are also provided to have trade-offs between the user’s tolerable approximation ratio and the complexity. An algorithm for a minimization problem is said to be a PTAS if (1) its solution is at most $(1 + \delta)$ times of the objective function of optimal solutions, and (2) its time complexity is polynomial in the input size by treating $\delta$ as a constant, where $\delta$ is a positive user-input parameter in a specified range. For a $\mathcal{NP}$-hard problem in a strong sense, PTAS is the best achievable in approximation algorithms [24]. By Theorem 1, there does not exist any PTAS for the multiprocessor task replication packing problem. However, by introducing a constant absolute error, an asymptotic polynomial-time approximation scheme (APTAS) can be achieved for the multiprocessor task replication packing problem, in which (1) its solution is at most a constant plus $(1 + \delta)$ times of the objective function of optimal solutions, and (2) its time complexity is polynomial in the input size by treating $\delta$ as a constant.

We will show that the multiprocessor task replication partition problem admits a PTAS, while an APTAS exists for the multiprocessor task replication packing problem by taking $K$ as a constant.

### 3 Multiprocessor Task Replication Partition

This section considers the multiprocessor task replication partition problem. Section 3.1 presents an efficient 2-approximation algorithm with an extension to a polynomial-time approximation scheme in Section 3.2. The time complexity of the PTAS is then improved in Section 3.3.

Since there does not exist any feasible solution for the multiprocessor task partition problem when $M < K$, we only focus our discussions on $M \geq K$ for brevity. For the rest of this section, we denote $U_{\max}(P, T)$ as the maximum utilization of the $M$ processors of a specified feasible task replication partition $P$ of the input task set $T$.

#### 3.1 A Greedy Algorithm

If the replication factor is 1, we can simply extend the greedy algorithm proposed by Graham [13] to the multiprocessor task replication partition problem as follows: Schedule tasks one by one in an arbitrary order and assign each task to the processor with the least amount of utilization so far. The algorithm provided in [13] is shown with a 2-approximation ratio. We can simply apply the above algorithm by treating the $K$ replicas of a task as $K$ different tasks. However, such an extension might assign multiple replicas of a task on a processor. Therefore, we have to be more careful on the procedure of task partition to guarantee that no two replicas of a task is assigned on a processor. The proposed algorithm here makes a slight modification of the above algorithm: Select $K$ distinct processors with the least amount of utilization so far, and assign each of the $K$ replicas of the currently considered task to these $K$ processors. The proposed algorithm is shown in Algorithm 1, denoted as Algorithm Greedy. Since Algorithm Greedy derives a task replication partition which assigns each task to $K$ distinct processors, Algorithm Greedy derives a feasible task replication partition for task set $T$ with $M \geq K$. The time complexity is $O(KN \log M)$ by applying a heap data structure in Step 5 in Algorithm 1.

We now show the approximation ratio of Algorithm Greedy. Let $u_{\max}$ be the maximum utilization of the $N$ tasks in $T$, i.e., $u_{\max} = \max_{t_i \in T} u_{i}$. Therefore, for any feasible task

---

**Algorithm 1: Greedy**

**Input:** $(T, M)$;  
1. order all the tasks in $T$ in an arbitrary order; 
2. set $U_1, U_2, \ldots, U_M$ to 0, and $T_1, T_2, \ldots, T_M$ to $\emptyset$; 
3. for $i \leftarrow 1$ to $|T|$ do 
4. for $j \leftarrow 1$ to $K$ do 
5. find the smallest $U_m$ with $\tau_i \notin T_m$: (break ties arbitrarily) 
6. $T_m \leftarrow T_m \cup \{\tau_i\}$ and $U_m \leftarrow U_m + u_i$; 
7. return the task replication partition $T_1, T_2, \ldots, T_M$;
Suppose that $U_{\max,k}$ (or $U_{\min,k}$, respectively) is the maximum (minimum, respectively) $U_m$ right after the $K$ replicas of task $\tau_i$ are assigned in Algorithm GREEDY. The following lemma shows that the difference between the maximum utilization and the minimum utilization of these $M$ processors is at most $u_{\max}$ right after assigning tasks $\tau_1, \tau_2, \ldots, \tau_i$ for any $i = 1, 2, \ldots, N$.

**Lemma 1** $U_{\max,i} \leq U_{\min,i} + u_{\max}$, for $i = 1, 2, \ldots, N$.

**Proof.** We prove this lemma by inductions. Clearly, the statement holds when $i = 1$. Suppose that $U_{\max,k} \leq U_{\min,k} + u_{\max}$, for some $k < N$. We are going to show that $U_{\max,k+1} \leq U_{\min,k+1} + u_{\max}$. If $U_{\max,k+1} = U_{\min,k}$, the induction is obvious. We only have to discuss on the case with $U_{\max,k+1} > U_{\min,k}$. Let $m^*$ ($m^1$, respectively) be the processor with $U_{m^*}$ ($U_{m^1}$, respectively) equal to $U_{\max,k+1}$ ($U_{\min,k+1}$, respectively) right after the $K$ replicas of task $\tau_{k+1}$ are assigned. (Ties are broken arbitrarily.)

Since $U_{\max,k+1} > U_{\max,k}$, one of the $K$ replicas of task $\tau_{k+1}$ must be assigned to processor $m^*$. By Algorithm GREEDY, $U_{\max,k+1}$ is no more than $U_{\max,k} + u_{k+1}$. Here are two cases: (1) $\tau_{k+1} \in T_{m^*}$, or (2) $\tau_{k+1} \notin T_{m^*}$. If $\tau_{k+1} \in T_{m^*}$, we know that $U_{\min,k+1} = U_{\min,k} + u_{k+1}$. Since $U_{\max,k} \leq U_{\min,k} + u_{\max}$, we have $U_{\max,k+1} \leq U_{\max,k} + u_{k+1} \leq U_{\min,k+1} + u_{\max}$ for the first case. If $\tau_{k+1} \notin T_{m^*}$, then $U_{\max,k+1} = U_{\max,k+1} + u_{k+1} \leq U_{\min,k+1}$, since $\tau_{k+1}$ is not assigned to processor $m^1$. The lemma is proved. □

By the definition of $U_{\min,N}$, we know that $U_m$ is no less than $U_{\min,N}$ for any $m = 1, 2, \ldots, M$ in the resulting task replication partition. Hence, for any feasible task replication partition $P^*$, we have

$$U_{\min,N} \leq \frac{1}{M} \sum_{m=1}^{M} U_m = \frac{K \sum_{i \in T} U_{i}}{M} \leq U_{\max}(P^*, T). \quad (2)$$

For brevity, let $P^G$ be the task replication partition for $T$ by applying Algorithm GREEDY. We can now show the approximation ratio of Algorithm GREEDY in the following theorem.

**Theorem 2** Algorithm GREEDY is a 2-approximation algorithm for the multiprocessor task replication partition problem.

**Proof.** This theorem comes from the combination of Lemma 1, Equation (1), and Equation (2). We have the following inequality:

$$U_{\max}(P^G, T) = U_{\max,N} \leq U_{\min,N} + u_{\max} \leq 2U_{\max}(P^*, T). \quad (3)$$

Therefore, we reach the conclusion. □

We now show that the above analysis of Algorithm GREEDY is tight. Consider the following instance for $M \geq 4$ with $\left[\frac{M}{2}\right] + 1$ tasks, where $\left[\frac{M}{2}\right]$ tasks are with utilization $\frac{1}{2M}$, and one task is with utilization $\frac{1}{2}$. Consider task replication partition for $K = 2$. An optimal task replication partition would assign the task with $\frac{1}{2}$ utilization first, as shown in Figure 1(a), and the maximum utilization of processors is $\frac{1}{2} + \left(2 \left[\frac{M}{2}\right] - (M - 2)\right) \frac{1}{2M}$. However, the solution of Algorithm GREEDY, shown in Figure 1(b), is with the maximum utilization of processors as $\left[\frac{M}{2}\right] + \frac{1}{2}$ when the $\left[\frac{M}{2}\right]$ light-utilization task are considered before the task with $\frac{1}{2}$ utilization. When $M$ is even, the ratio is $\frac{1 + 0.5}{1 + 0.5} = 2 - \frac{2}{0.5M + 1}$. When $M$ is odd, the ratio is $\frac{1 + 0.5}{1 + 0.5} = 2 - \frac{2.5}{0.5M + 1}$. Hence, the analysis of Algorithm GREEDY is tight when $M$ is sufficiently large.

### 3.2 A Polynomial-Time Approximation Scheme

This subsection presents a polynomial-time approximation scheme (PTAS) for the multiprocessor task replication partition problem. Let $B$ be $\max\{0.5U_{\max}(P^G, T), u_{\max}\}$, where $P^G$ is the task replication partition for $T$ by applying Algorithm GREEDY. With the analysis in Section 3.1, there does not exist any feasible task replication partition with the maximum utilization of processors less than $B$. In this subsection, we show how to derive optimal solutions for a special case. The proposed PTAS is based on the observation of the special case.

**3.2.1 A polynomial-time algorithm to derive optimal solutions for special cases**

In this section, we consider special cases when the number of distinct utilizations of tasks in $T$ is fixed, and the ratio of an upper bound of the optimal solution to the minimum utilization among the tasks in $T$ is a constant. For such cases, these utilizations are denoted as $v_1, v_2, \ldots, v_p$, and let $v_1 \leq v_2 < v_3 < \cdots < v_p$, where $v_p$ is the maximum utilization of tasks in $T$. (The reason why we keep the possibility that $v_1 = v_2$ will be explained when we apply the algorithm to develop our proposed PTAS.) The number of tasks in $T$ with utilization equal to $v_i$ is denoted as $n_i$. We have the following lemma for optimal solutions.

**Lemma 2** For an optimal task replication partition of $T$ with $u_i \geq v_1$ for every task $\tau_i$ in $T$, the number of tasks assigned on each processor is at most $\left\lceil \frac{C}{v_1} \right\rceil$, where $C$ is an upper bound of the maximum utilization of processors of an optimal solution.
Proof. This comes directly from the definition of $C$ and the fact $v_i \leq u_i$ for every task $\tau_i$ in $T$. □

A configuration for a processor is defined as a vector $\vec{w} = (w_1, w_2, \ldots, w_\rho)$, in which $w_i \in \{0, 1, \ldots, n_i\}$ denotes the number of tasks with utilization equal to $v_i$ in $T$ in the configuration. By Lemma 2, to derive an optimal solution, we only have to consider configurations $\vec{w}$ on processors with $\sum_{i=1}^L w_i \leq \left\lfloor \frac{C}{v_i} \right\rfloor$.

As a result, there are at most $Q = (\left\lfloor \frac{C}{v_i} \right\rfloor + \rho)$ different configurations of a processor for optimal solutions. Since $\frac{C}{v_i}$ and $\rho$ are two constants in this subsection, $Q$ is a large constant! For a feasible task replication partition, let $\vec{w}_m$ be the corresponding configuration on processor $m$, in which $w_{m,i}$ denotes the number of tasks with $v_i$ utilization in $T$ in configuration $\vec{w}_m$. As a result, we only have to consider configurations $(\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_M)$ with $\sum_{m=1}^M w_{m,i} = K \cdot n_i$ for $i = 1, 2, \ldots, \rho$, $\sum_{i=1}^\rho w_{m,i} \leq \left\lfloor \frac{C}{v_i} \right\rfloor$ for $m = 1, 2, \ldots, M$, and $w_{m,i} \leq n_i$ for $i = 1, 2, \ldots, \rho$ and $m = 1, 2, \ldots, M$.

For a configuration $\vec{w}_m$ on processor $m$, the utilization on the processor is equal to $\sum_{i=1}^\rho v_i w_{m,i}$, which can be obtained in $O(\rho)$. Since the number of $(\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_M)$ configurations to achieve $\sum_{m=1}^M w_{m,i} = K \cdot n_i$ for $i = 1, 2, \ldots, \rho$, $\sum_{i=1}^\rho w_{m,i} \leq \left\lfloor \frac{C}{v_i} \right\rfloor$ for $m = 1, 2, \ldots, M$, is at most $(M+Q)^Q = O((M)^Q)$. We can derive an optimal solution by enumerating these different configurations for $T$ on the $M$ processors and picking up the optimal configurations $(\vec{w}^*_1, \vec{w}^*_2, \ldots, \vec{w}^*_M)$ on the $M$ processors.

We now show how to assign tasks onto processors based on $(\vec{w}^*_1, \vec{w}^*_2, \ldots, \vec{w}^*_M)$ to derive a feasible task replication partition. We consider the assignment of $K$ replicas of the $n_i$ tasks with utilization equal to $v_i$ accordingly. Let these $n_i$ tasks be ordered arbitrarily and denoted as $\tau_{1,i}, \tau_{2,i}, \ldots, \tau_{n_i,i}$. Let $j^*$ be 1 initially. We consider $m$ from 1 to $M$ to assign these $n_i$ tasks as follows: Assign the $\left\lfloor \frac{j^*}{n_i} \right\rfloor$ -th replication of task $\tau_{(j^*-1) \mod n_i}+1,i$ to processor $m$ for $j = j^*, j^*+1, \ldots, j^*+w_{m,i}-1$, and update $j^*$ to $j^*+w_{m,i}$ after the above assignment. Since $\sum_{m=1}^M w_{m,i} = K \cdot n_i$ for $i = 1, 2, \ldots, \rho$, and $w_{m,i} \leq n_i$ for $i = 1, 2, \ldots, \rho$ and $m = 1, 2, \ldots, M$, the above task replication partition will assign each task in $T$ to exactly $K$ distinct processors. Thus, combining the above statements, we have proved the following theorem.

Theorem 3 An optimal task replication partition for the multi-processor task replication partition problem for task set $T$ on $M$ processors can be derived in $O((M)^Q + N)$ if there are at most $\rho$ different utilizations for tasks in $T$, and the utilization for any task in $T$ is no less than $|C|$, where $C$ is an upper bound of optimal solutions, $\rho$ and $\beta$ are two constant numbers, and $Q = (\beta + \rho)$ is also a constant.

3.2.2 A polynomial-time approximation scheme for general cases

Our polynomial-time approximation scheme for the multiprocessor task replication partition problem is based on rounding the utilization of each task so that the number of different utilizations is fixed, and the ratio of an upper bound of the optimal solution to the minimum utilization among the tasks in $T$ is a constant. After rounding the parameters of tasks, we can adopt the polynomial-time algorithm in Theorem 3 to derive an optimal schedule for the rounded input instance. A task replication partition can then be determined with near optimal performance.

For notational brevity, let $\epsilon$ be a fixed constant, which will be specified by the end of this subsection. Let $P_G$ be the task replication partition by applying Algorithm GREEDY for $T$ on $M$ processors. Let $B$ be $\max\{0.5U_{\max}(P_G, T), u_{\max}\}$. We classify tasks in $T$ into two types: For any task $\tau_i$ in $T$ with utilization no less than $\epsilon B$, i.e., $u_i \geq \epsilon B$, we call such a task as a large task. On the other hand, task $\tau_i$ in $T$ is referred to as a small task if $u_i < \epsilon B$. Let $S_1$ ($S_2$, respectively) be the task set which consists of all the large (small, respectively) tasks in $T$.

For each large task $\tau_i$ in $S_1$, let $\ell_i$ be the integer $\ell$ with

$$\epsilon B(1 + \epsilon)^{\ell} \leq u_i < \epsilon B(1 + \epsilon)^{\ell+1}.$$ 

For each large task $\tau_i$ in $S_1$, we create a rounded task $\tau_i^x$ by setting $w_i^x$ as $\epsilon B(1 + \epsilon)^{\ell_i}$. Since $u_i \leq B$ for all $\tau_i$ in $T$, we know that $0 \leq \ell_i \leq \left\lfloor \log_{1+\epsilon} \frac{1}{\epsilon} \right\rfloor$. The constructed task set is denoted by $S_1^x$. Hence, the number of distinct utilizations in $S_1^x$ is at most $\left\lfloor \log_{1+\epsilon} \frac{1}{\epsilon} \right\rfloor + 1$.

For the small tasks in $S_2$, we construct $L$ identical tasks with utilization equal to $\epsilon B$, where $L = \left\lfloor \frac{\sum_{i=1}^{n} w_{m,i}}{\epsilon B} \right\rfloor$. The constructed task set by the small tasks is denoted as $S_2^x$, where each task $\tau_i^x$ in $S_2^x$ is with $\epsilon B$ utilization.

As a result, we construct a set of tasks $T^x = S_1^x \cup S_2^x$ with at most $\left\lfloor \log_{1+\epsilon} \frac{1}{\epsilon} \right\rfloor + 1$ different values on their utilizations, and the minimum utilization of tasks in $T^x$ is $\epsilon B$. The following lemma shows that $2B + \epsilon B$ is an upper bound of an optimal task replication partition for $T^x$.

Lemma 3 The maximum utilization of the processors of an optimal task replication partition for task set $T^x$ on $M$ processors is at most $2B + \epsilon B$.

Proof. We show that any feasible task replication partition $P$ for task set $T$ can be transformed into a feasible task replication partition for task set $T^x$ with at most $\epsilon B$ increase of the utilization of each processor. For each task $\tau_i$ in $S_1$, task $\tau_i^x$ is assigned to processor $m$ if task $\tau_i$ is assigned to processor $m$ in $P$. Then, we assign $\min\{\frac{U_m}{\epsilon B}, L\}$ tasks with utilization $\epsilon B$ on processor $m$, where $U_m$ is the total utilization of small tasks assigned on processor $m$ in task replication partition $P$. As a result, the total utilization of the tasks on each processor is increased at most $\epsilon B$, compared to the task replication partition $P$. Although the number of tasks constructed from the small tasks is at least $K L$, the resulting task replication partition is feasible for $T^x$ by removing some redundant tasks so that the number of tasks constructed from the small tasks is equal to $K L$. □

We take $v_1 = \epsilon B$, $v_{i+1} = L$, $v_{i+2} = \epsilon B(1 + \epsilon)^{\ell_i}$, and $n_{i+2}$ as the number of large tasks $\tau_i^x$ with $\ell_i$ equal to $i$ for $i = 0, 1, \ldots$, $\left\lfloor \log_{1+\epsilon} \frac{1}{\epsilon} \right\rfloor$. Therefore, we have $v_1 = v_2 = \epsilon B$ in
the above setting. The reason why we keep \(v_1 \leq v_2\) in Section 3.2.1 is to know whether a task in \(T^o\) with a rounded utilization equal to \(\epsilon B\) is constructed from a large task or some small tasks. Hence, we could derive an optimal solution on scheduling tasks in \(T^o\) on \(M\) processors in polynomial time by applying the algorithm presented in Theorem 3, where \(C = 2B + \epsilon B\), \(\beta = \frac{1}{2M}\), and \(\rho = \left[\log_2(1 + \frac{1}{\epsilon B})\right] + 2\).

Let \(T^o_m\) be the set of tasks assigned on processor \(m\) for a derived optimal task replication partition of \(T^o\). We now show how to assign tasks in \(T\) based on the task assignment \(T^o_1, T^o_2, \ldots, T^o_M\). Let \(N_2\) be the number of tasks in \(S_2\). For brevity, we order tasks in \(S_2\) in an arbitrary order, and denote the \(j\)-th replication of the \(i\)-th task in \(S_2\) by \(\tau_j(\tau_{i-1})::N_2::\tau_j\) for \(i = 1, 2, \ldots, N_2\) and \(j = 1, 2, \ldots, K\). The utilization of task \(\tau_k::S_2\) is \(u_k::S_2\). Let \(i'\) be 1 initially, which represents the index of the small task to be assigned. We assign tasks onto processors from processor 1 to \(M\) accordingly. When we assign tasks onto processor \(m\), the procedures in Procedure 1 are applied.

**Procedure 1**
**Input:** \(T^o_m, S_1, S_2, S^o_1, S^o_2, i';\)

- **(A1)** For each large task \(\tau^o_j(\tau^o_j \in S^o_2)\) assigned to \(T^o_m\), we assign task \(\tau_i\) on processor \(m\), in which \(\tau_i\) is the task in \(T\) from which we construct task \(\tau^o_j\).

- **(A2)** Suppose that \(T^o_m\) contains \(R_m\) tasks in \(S^o_2\). Find the greatest index \(j\) with \(j - i' < N_2\), \(j < N_2:K\), and \(\sum_{k=i'}^{j} u_{k::S_2} < \epsilon B(R_m + 2)\), where \(N_2\) is the cardinality of task set \(S_2\). We then assign task \(\tau_i::S_2\) in \(S_2\) to processor \(m\) for \(i = i', i' + 1, \ldots, j\). The index \(i'\) is updated to \(j + 1\).

- **(A3)** Return the resulting task assignment \(T^o_m\), and index \(i'\) for the assignment of tasks on the next processor.

Let the resulting task assignment be \(T_1^o, T_2^o, \ldots, T_M^o\). The pseudo-code of the above algorithm is shown in Algorithm 2, denoted as Algorithm Rounding for brevity. The following lemma shows that the resulting task replication partition assigns each task exactly on \(K\) distinct processors, which implies the feasibility of task replication partition \((T^o_1, T^o_2, \ldots, T^o_M)\).

**Lemma 4** For each task \(\tau_i\) in \(T\), \(\tau_i\) is in exactly \(K\) distinct processors in the derived task replication partition from Algorithm Rounding.

**Proof.** Since each large task in \(S^o_2\) is assigned to \(K\) distinct processors, each large task in \(S_1\) is assigned to \(K\) distinct processors by applying procedure (A1) in Procedure 1. We only focus on showing that each small task \(\tau_i\) in \(S_2\) is assigned to \(K\) distinct processors by applying procedure (A2) in Procedure 1. Clearly, no two replicas of a small task will be assigned on the same processor. To show each small task is assigned to \(K\) distinct processors, we only have to show that \(i' < N_2:K\) after applying procedure (A2) in Procedure 1 on these \(M\) processors. Assume that \(i'\) is no more than \(N_2:K\) for contradiction. Let \(K'\) be the number of processors on which all the tasks in \(S_2\) are assigned. For each of these \(K'\) processors \(m\), \(R_m\) must be at least \(L - 1\). In addition to these \(K'\) processors, there are \(M'\) other processors assigned with some tasks in \(S_2\), where \(M' \geq K - K'\). By applying procedure (A2) in Procedure 1 with the fact that \(u_i < \epsilon B\) for any task \(\tau_i\) \(\in S_2\), since \(i' < N_2:K\), each processor \(m\) of these \(M'\) processors must assign tasks with \(\sum_{\tau_i \in (S_2 \cap T_m)} u_i > (R_m + 1) \epsilon B\). As a result,

\[
(K - K') \sum_{\tau_i \in S_2} u_i > \epsilon B(K - K' + L(K - K')).
\]

Because \(L = \left\lceil \frac{\sum_{\tau_i \in S_2} u_i}{\epsilon B} \right\rceil\), we have

\[
(K - K') \sum_{\tau_i \in S_2} u_i < \epsilon B(K - K' + L(K - K')).
\]

We reach the contradiction. \(\square\)

Now, we have shown that Algorithm Rounding is with polynomial-time complexity to derive a feasible task replication partition. For the rest of this subsection, we show the approximation ratio of Algorithm Rounding. For notational brevity, let \(P^o\) be the task replication partition \(T^o_1, T^o_2, \ldots, T^o_M\). The following lemma shows the difference between \(U_{max}(P^o, T^o)\) and any feasible task replication partition \(P^o\) of \(T^o\).

**Lemma 5** For any feasible task replication partition \(P^*\) of task set \(T^o\),

\[
U_{max}(P^*, T^o) + \epsilon B \geq U_{max}(P^o, T^o),
\]

where \(P^o\) is the task replication partition \(T^o_1, T^o_2, \ldots, T^o_M\) of task set \(T^o\).

**Proof.** The proof is as the same as that in Lemma 3. \(\square\)

The following lemma shows that the difference between the total utilization of tasks in task set \(T^o_m\) and that in the resulting task assignment \(T^e_m\) is bounded.

**Lemma 6**

\[
(1 + \epsilon)(\sum_{\tau_i \in T^e_m} u_i^e) + 2 \epsilon B \geq \sum_{\tau_i \in T^o_m} u_i^o
\]
for 

Proof. \( T_m \) consists of two parts: (1) large tasks and (2) small tasks. For each large task \( \tau_i \) in \( T_m \), we have \((1 + \epsilon)u_i^* < u_i\). As shown in procedure \((A2)\) in Procedure 1, for the small tasks, we know that \( \sum_{\tau_i \in (T_m \cap S_2)} v_i < (R_m + 2)\epsilon B \), where \( R_m \) is the number of tasks in \( S_2 \) in \( T_m \). Therefore,

\[
\sum_{\tau_i \in T_m} u_i = \sum_{\tau_i \in (T_m \cap S_1)} u_i + \sum_{\tau_i \in (T_m \cap S_2)} u_i < (1 + \epsilon) \sum_{\tau_i \in (T_m \cap S_1)} u_i^* + (R_m + 2)\epsilon B \leq (1 + \epsilon) \sum_{\tau_i \in T_m} u_i^* + 2\epsilon B.
\]

\( \square \)

We can now show that Algorithm ROUNIDDING is a PTAS.

Theorem 4 Algorithm ROUNIDDING is a polynomial-time approximation scheme for the multiprocessor task replication partition problem.

Proof. We only have to show the approximation ratio of Algorithm ROUNIDDING to complete the proof of this theorem since the time and space complexity is polynomial in \( M \) and \( N \) by treating \( \epsilon \) as a constant. Let \( P^* \) be the resulting task replication partition by applying Algorithm ROUNIDDING. By applying Lemma 5 and Lemma 6 and the fact that \( B \) is a lower bound of feasible solutions, we have

\[
U_{\max}(P^*, T) \leq (1 + \epsilon) U_{\max}(P^*, T^b) + 2\epsilon B \leq (1 + \epsilon) (U_{\max}(P^*, T) + \epsilon B) + 2\epsilon B \leq (1 + \epsilon) U_{\max}(P^*, T) + 4\epsilon B \leq (1 + 5\epsilon) U_{\max}(P^*, T),
\]

where the first inequality comes from Lemma 6, the second inequality comes from Lemma 5, \( 0 < \epsilon \leq 1 \), and \( P^* \) is any feasible task replication partition of \( T \) on \( M \) processors. As a result, by taking \( \epsilon \) as \( \delta \), we have a \((1 + \delta)\) - approximation algorithm with \( 0 < \delta \leq 5 \). \( \square \)

3.3 A More Efficient PTAS

Section 3.2 provides a polynomial-time approximation scheme with very high complexity since \( Q \) is a very large constant in Theorem 3. We now show how to reduce the time complexity of Algorithm ROUNIDDING. This subsection presents an approximation algorithm to derive a \((1 + \eta\epsilon)\) - approximation of task set \( T^b \) for the multiprocessor task replication partition problem for a user-specified value \( \eta \). Suppose that the derived task replication partition for \( T^b \) on \( M \) processors is \( P^b \), where \( U_{\max}(P^b, T^b) \leq (1 + \eta\epsilon) U_{\max}(P^b, T^p) \). By performing task replication assignment based on \( P^b \) instead of \( P^b \), we have

\[
U_{\max}(P^b, T) \leq (1 + \epsilon) U_{\max}(P^b, T^b) + 2\epsilon B \leq (1 + \epsilon) (1 + \eta\epsilon) U_{\max}(P^b, T) + \epsilon B + 2\epsilon B \leq (1 + 5\epsilon + 4\eta\epsilon) U_{\max}(P^b, T),
\]

where \( 0 < \epsilon \leq 1 \).

For the rest of this section, we show how to derive task replication partition \( P^b \) of task set \( T^b \) with \( U_{\max}(P^b, T^b) \leq (1 + \eta\epsilon) U_{\max}(P^b, T^b) \). We first present a dynamic-programming algorithm to determine the minimum number of processors required to derive a feasible task replication partition without violating a given utilization bound \( C^* \). Then, the algorithm for the derivation of task replication partition \( P^b \) follows.

With the same argument in Section 3.2.1, there are at most \(( \frac{n^m}{\rho^i} + \epsilon \tau) \) different configurations, where \( \rho = \lceil \log_3 + \frac{1}{\epsilon} \rceil + 2 \) and \( \epsilon \) is \( \epsilon B \). A configuration is said to be feasible under utilization constraint \( C^* \) of a task set if its total utilization of the tasks is no more than \( C^* \) and the number of tasks with utilization \( v_i \) in the configuration is no more than \( n_i \). Let \( \tilde{W} \) be the set of the feasible configurations under utilization constraint \( C^* \) of \( T^b \). Hence, for each \( \tilde{w} \) in \( \tilde{W} \), \( w_i \leq n_i \) for \( i = 1, 2, \ldots, \rho \), and \( \sum_{i=1}^{\rho} w_i \leq C^* \).

By the definition of the multiprocessor task replication partition problem, there are \( K \cdot n_i \) tasks with utilization \( v_i \). Let \( \psi(C^*, \lambda_1, \lambda_2, \ldots, \lambda_{\rho}) \) be the minimum number \( M^\dagger \) of processors required to derive \( M^\dagger \) feasible configurations \((\tilde{w}_{1^\dagger}, \tilde{w}_{2^\dagger}, \ldots, \tilde{w}_{M^\dagger})\) under utilization constraint \( C^* \) such that \( \sum_{j=1}^{M^\dagger} w_{j,i} = \lambda_i \) for \( i = 1, 2, \ldots, \rho \). Hence, \( \psi(C^*, \lambda_1, \lambda_2, \ldots, \lambda_{\rho}) \geq \psi(C^*, \lambda_1, \lambda_2, \ldots, \lambda_{\rho}) \) for any \( \lambda_i \leq 0 \) for \( 1 \leq i \leq \rho \). By the definition of \( \psi() \), we can have the following recurrence:

\[
\psi(C^*, \lambda_1, \lambda_2, \ldots, \lambda_{\rho}) = 1 + \min_{\tilde{w} \in \tilde{W}} \psi(C^*, \lambda_1 - w_1, \lambda_2 - w_2, \ldots, \lambda_{\rho} - w_{\rho}). \tag{5}
\]

As a result, the computing of \( \psi(C^*, K \cdot n_1, K \cdot n_2, \ldots, K \cdot n_{\rho}) \) can be done by building the dynamic programming table from \( \psi(C^*, 0, 0, \ldots, 0) \) by applying the recurrence in Equation (5). There are \( O((NK)^{\rho}) \) entries, and the computing of each entry takes \( O(\tilde{W} | \rho |) \) time. As a result, the time complexity to compute \( \psi(C^*, K \cdot n_1, K \cdot n_2, \ldots, K \cdot n_{\rho}) \) is \( O((NK)^{\rho} \left( \frac{\epsilon B}{2} + \frac{\epsilon B}{\rho} \right)) \). The corresponding task replication partition \( P^b \) for \( T^b \) can be obtained by backtracking the dynamic programming table and with the same task assignment strategy in Section 3.2.1.

For notational brevity, let \( \psi(C^*) \) be the value of \( \psi(C^*, K \cdot n_1, K \cdot n_2, \ldots, K \cdot n_{\rho}) \). If we can find the minimum value \( C^1 \) of \( C^* \) such that \( \psi(C^*) \) is no more than \( M \), then \( C^1 \) is the optimal maximum utilization of processors of optimal solutions for \( T^b \). However, it might take a lot of time. We can perform a binary search of \( C^1 \) in the interval \([I_{left}, I_{right}]\), where \( I_{left} \) is initialized as \( B \) and \( I_{right} \) as \( 2B + \epsilon B \). If \( I_{right} - I_{left} \leq \eta\epsilon B \), we return the task replication partition \( P^b \) by setting \( C^* \) as \( I_{right} \). If \( \psi(C^*) \left( \frac{I_{right} - I_{left}}{2} \right) > M \), let \( I_{left} \) be updated to \( I_{right} - I_{right} + \frac{I_{left} - I_{right}}{2} \); otherwise, let \( I_{right} \) be updated to \( I_{right} + \frac{I_{left} - I_{right}}{2} \). The procedure takes \( \log_2 \frac{1 + \epsilon}{\eta \epsilon} \) iterations and then derives task replication partition \( P^b \) by backtracking the dynamic programming table.

Therefore, we can derive a feasible task replication partition for \( T^b \) on \( M \) processors with maximum utilization no more than
Table 1. Task parameters for the example illustration in Section 3.4, where $B$ is 0.605

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
<th>$\tau_5$</th>
<th>$\tau_6$</th>
<th>$\tau_7$</th>
<th>$\tau_8$</th>
<th>$\tau_9$</th>
<th>$\tau_{10}$</th>
<th>$\tau_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>0.25</td>
<td>0.24</td>
<td>0.46</td>
<td>0.55</td>
<td>0.36</td>
<td>0.37</td>
<td>0.38</td>
<td>0.16</td>
<td>0.45</td>
<td>0.02</td>
</tr>
<tr>
<td>rounded utilization ($\epsilon = 0.2$)</td>
<td>small</td>
<td>0.2091</td>
<td>0.2091</td>
<td>0.4336</td>
<td>0.5203</td>
<td>0.3011</td>
<td>0.3613</td>
<td>0.3613</td>
<td>0.1452</td>
<td>0.4336</td>
</tr>
<tr>
<td>rounded utilization ($\epsilon = 0.1$)</td>
<td>small</td>
<td>0.2297</td>
<td>0.2297</td>
<td>0.4477</td>
<td>0.5417</td>
<td>0.3364</td>
<td>0.3646</td>
<td>0.3700</td>
<td>0.1569</td>
<td>0.4477</td>
</tr>
</tbody>
</table>

Figure 2. Task replication partitions for tasks in Table 1 on 7 processors with $K = 2$: (a) an optimal task replication with maximum utilization 0.95, (b) the solution by applying Algorithm GREEDY from $\tau_1$ to $\tau_{11}$ with maximum utilization 1.21, (c) the solution by applying Algorithm GREEDY with the largest-utilization-first strategy with maximum utilization 1.04, (d) the solution by applying Algorithm Rounding with $\epsilon = 0.2$ and maximum utilization 0.97, and (e) the solution by applying Algorithm Rounding with $\epsilon = 0.1$ and maximum utilization 0.95.

(1 + $\eta$) $U_{\text{max}}(P^0, T^0)$ in $O\left(\log_2 \frac{1+\eta}{\rho} \cdot \rho (NK)^\rho \left(\frac{1+\eta}{\rho}\right)\right)$ time complexity, where $\rho$ is $\log_2 \frac{1}{\epsilon} + 2$. For example, by taking $\eta$ as 0.25, we can derive a $(1+6\epsilon)$-approximated solution for the multiprocessor task replication partition problem with time complexity $O\left(\log_2 \frac{1+\eta}{\epsilon} \cdot \rho (NK)^\rho \left(\frac{1+\eta}{\rho}\right)\right)$. Moreover, since $|\hat{W}|$ is also bounded by $O(N^p)$ and both $\epsilon$ and $\rho$ are constants, the time complexity is also $O((NK)^p N^p)$.

### 3.4 An Example for Illustration

To provide a clear understanding of the proposed algorithms for the multiprocessor task replication partition problem, we use an example to illustrate the solutions that can be obtained. Consider the input instance shown in Table 1 to be scheduled on 7 processors, while the replication factor $K$ is 2. Figure 2(a) shows an optimal task replication partition with maximum utilization of processors equal to 0.95. Applying Algorithm GREEDY by considering tasks from $\tau_1$ to $\tau_{11}$ leads to a task replication partition shown in Figure 2(b), in which the maximum utilization of processors is 1.21. Hence, the value of $B$ is 0.605. Figure 2(c) presents the task replication partition by applying Algorithm GREEDY with the largest-utilization-first strategy, i.e., considering tasks from high utilization to low utilization. Figure 2(d) presents the task replication partition by applying Algorithm Rounding with $\epsilon = 0.2$ down the utilization of the 10 large tasks in $S_1$, shown in Table 1. After the rounding-down procedure, there are six distinct utilizations for tasks in $S_1$, and no task with utilization equal to $\epsilon B$ is constructed. Similarly, Figure 2(e) shows the task replication partition by setting $\epsilon = 0.1$, in which the maximum utilization of processors is 0.95. Both the derived solutions from Algorithm Rounding are feasible, and the maximum utilization of processors is minimized.

### 4 Multiprocessor Task Replication Packing

This section considers the multiprocessor task replication packing problem. Section 4.1 presents greedy algorithms with approximation ratios, while an asymptotic polynomial-time approximation scheme for the problem is shown in Section 4.2.

### 4.1 Combinatorial Approximation Algorithms

It is easy to achieve 2-approximation for this problem. The framework is referred to as Framework FIT. Suppose that the tasks in $T$ are ordered in an order. The order could be arbitrary, decreasing on the utilization, or increasing on the utilization. Then, in the $i$-th step to pack task $\tau_i$, we find $K$ allocated processors that can fit the assignment of $K$ replicas of task $\tau_i$. If we cannot find $K$ allocated processors to fit task $\tau_i$, we allocate new processors so that we can find $K$ allocated processors that can fit the assignment of $K$ replicas of task $\tau_i$. The algorithm framework is shown in Framework 1. The strategy in Step 5 of Framework 1 could be first-fit, worst-fit, or best-fit. The first-fit strategy is to break ties by choosing the smallest index $m$. The worst-fit (best-fit, respectively) strategy is to break ties by choosing the processor with the smallest (largest, respectively) utilization. Suppose that $M$ is the number of processors allocated by applying Framework FIT. The time complexity is $O(NK \log NK)$. The following theorem shows that the first-fit, worst-fit, and best-fit strategies are 2-approximation algorithms.
Theorem 5 The first-fit, worst-fit, and best-fit strategies are 2-approximation algorithms for the multiprocessor task replication packing problem.

Proof. The proof is very similar to that in [24, §9] for the first-fit algorithm for the bin packing problem. \endproof

4.2 An Asymptotic Polynomial-Time Approximation Scheme

Our asymptotic polynomial-time approximation scheme for the multiprocessor task replication packing problem is based on the rounding of the utilization of each task. The approach is similar to that in Section 3.2.2, but with different rounding techniques. For notational brevity, we will override some terms defined in Section 3. Let \( \Psi(T) \) be the minimum number of processors required to derive a feasible task replication packing.

We, again, classify tasks in \( T \) into two types: For any task \( \tau_i \) in \( T \) with \( u_i > \epsilon \), such a task is referred to as a large task. On the other hand, task \( \tau_i \) in \( T \) is referred to as a small task if \( u_i \leq \epsilon \).

Let \( S_1 \) (\( S_2 \), respectively) be the task set which consists of all the large (small, respectively) tasks in \( T \). Different from Algorithm \textsc{Rounding}, we will first discard tasks in \( S_2 \) and round up the utilization of tasks in \( S_1 \) to form a task set \( T^2 \). An optimal task replication packing \( P^k \) for \( T^2 \) is then determined. Then, we pack the small tasks in \( S_2 \).

We now show how to construct task set \( T^2 \). Let tasks in \( S_1 \) be sorted in a non-decreasing order of their utilizations. \( N_1 \) is the number of tasks in \( S_1 \), and \( Z = \max\{ \lceil \frac{N_1 \epsilon^2}{1} \rceil, 1 \} \). We then partition tasks in \( S_1 \) according to the sorted order into \( Y \) groups so that each group, excluding the last one, contains exactly \( Z \) consecutive tasks in the sorted order and the \( Y \)-th group contains exactly \( N_1 - (Y - 1)Z \) tasks. As a result, \( Y \) is at most \( \lceil \frac{N_1}{Z} \rceil \). Let \( \mu_k^1 \) be the largest utilization in the \( k \)-th group. For task \( \tau_i \) in the \( k \)-th group, we construct a task \( \tau_i^2 \) with utilization \( \mu_k^1 \). Let \( T^2 \) be the set of tasks constructed from \( S_1 \).

Deriving \( \Psi(T^2) \) by applying the dynamic programming algorithm in Section 3.3 takes \( O(N_1 N K^p \lceil \frac{1}{Z^p} \rceil + 1) \) or \( O((N K^p)^p N^p) \) time, where \( \rho \) is \( \lceil \frac{2}{Z} \rceil \). The corresponding task replication partition \( P^2 \) for \( T^2 \) can be obtained by backtracking the dynamic programming table.

For each task \( \tau_i \) in \( S_1 \), we assign task \( \tau_i \) to processor \( m \) if \( \tau_i^1 \) is assigned to processor \( m \) in \( P^2 \). We now go on packing the small tasks in \( S_2 \) by applying the first-fit (worst-fit, or best-fit) strategy on replication task partition \( P^2 \). Let the resulting task replication partition be \( T_1, T_2, \ldots, T_{M^*} \), where \( T_m \) is not empty for \( m = 1, 2, \ldots, M^* \) and \( M^* \) is the number of processors used for \( T \) in the resulting solution. We denote the above algorithm as Algorithm \textsc{P-Rounding}. Since \( u_i \) is no more than \( u_i^2 \), the resulting task replication packing is feasible. For the rest of this section, we show the approximation ratio of Algorithm \textsc{P-Rounding}.

**Lemma 7** If \( M^* > \Psi(T^2) \), then \( M^* \leq (1 + 2\epsilon) \Psi(T) + K \), where \( 0 < \epsilon \leq 0.5 \) and \( M^* \) is the number of processors in the solution derived from Algorithm \textsc{P-Rounding} of \( T \).

**Framework 1:** \textsc{Fit}

**Input:** \( T \);

1: order all the tasks in \( T \) in any specified order;
2: set \( M \) as 0;
3: for \( i \leftarrow 1 \) to |\( T \)| do
   4: for \( j \leftarrow 1 \) to \( K \) do
      5: find a processor \( m \) with \( m \leq M \), \( \tau_i \notin T_m \), and \( U_m + u_i \leq 1 \); (break ties by a specified strategy)
      6: if \( m \) exists then
         7: \( T_m \leftarrow T_m \cup \{ \tau_i \} \) and \( U_m \leftarrow U_m + u_i \);
      8: else
         9: \( M \leftarrow M + 1 \), \( T_M \leftarrow \{ \tau_i \} \) and \( U_M \leftarrow u_i \);
   10: return the task replication partition \( T_1, T_2, \ldots, T_M \);

**Proof.** Since \( M^* > \Psi(T^2) \) and \( u_i \leq \epsilon \) for every task \( \tau_i \) in \( S_2 \), there are at least \( M^* - K \) allocated processors with utilization of tasks greater than \((1 - \epsilon)\). Therefore, \( \Psi(T) \geq \sum_{\tau_i \in T} u_i \geq (M^* - K)(1 - \epsilon) \), and
\[
M^* \leq \frac{\psi(T)}{1 - \epsilon} + K \leq (1 + 2\epsilon)\Psi(T) + K,
\]
where the last inequality comes from \( 0 < \epsilon \leq 0.5 \). \endproof

**Lemma 8** If \( M^* = \Psi(T^2) \), \( M^* \leq (1 + \epsilon) \Psi(T) \), where \( M^* \) is the number of processors in the solution derived from Algorithm \textsc{P-Rounding} of \( T \).

**Proof.** This lemma can be proved by showing that \( \Psi(T^2) \leq (1 + \epsilon) \Psi(T) \). Recall the partition of \( T \) into \( Y \) groups. If each group contains at most 1 task, i.e., \( Z = 1 \), we know that \( \Psi(T^2) \leq \Psi(T) \), since no utilization rounding of tasks is performed in \( T^2 \). We only consider \( Z \geq 2 \) for the rest of this proof. To show this, we construct a task set \( T^0 \) with \( \Psi(T^0) \leq (1 + \epsilon)\Psi(T^2) \) and \( \Psi(T^0) \leq \Psi(T) \).

Let \( \hat{\mu}_k \) be the smallest utilization of the tasks in the \( k \)-th group. Hence, \( \hat{\mu}_k \geq \mu_k^1 - 1 \) for \( k \geq 2 \). For task \( \tau_i \) in \( S_1 \) in the \( k \)-th group, we construct a task \( \tau_i^2 \) with utilization \( \hat{\mu}_k \). Let \( T^0 \) be the set of tasks constructed from \( S_1 \). Figure 3 provides an illustrative example to show the difference between task sets \( T^2 \) and \( T^0 \) constructed above. Clearly, \( \Psi(T^0) \leq \Psi(T) \). We now show that \( \Psi(T^0) \leq (1 + \epsilon)\Psi(T^2) \). Since tasks in \( T^2 \) are with utilization greater than \( \epsilon \), we have \( \Psi(T^0) \geq K\epsilon N_1 \).

For an optimal task replication packing \( P^{*2} \) of \( T^2 \), we can derive a feasible task replication packing for \( T^2 \) with at most \( \Psi(T^2) + K \lceil \epsilon^2 N_1 \rceil \) processors. If the rounding-down task constructed in \( T^2 \) from the \( i \)-th task in the sorted order is assigned on processor \( m \) in \( P^{*2} \) with \( i > \frac{\epsilon^2 N_1}{1} \), we assign the rounding-up task constructed in \( T^0 \) from the \( (i - \frac{\epsilon^2 N_1}{1}) \)-th task on processor \( m \). Since \( \hat{\mu}_k \geq \mu_k^1 - 1 \) for \( k \geq 2 \), the above task replication packing is feasible for \( T^0 \), excluding the \( \lceil \epsilon^2 N_1 \rceil \) largest tasks in \( T^2 \). Allocating a processor for each of the \( K \) replicas of the \( \lceil \epsilon^2 N_1 \rceil \) largest tasks in \( T^2 \) leads to a solution using \( \Psi(T^0) + K \lceil \epsilon^2 N_1 \rceil \) processors. As a result, we have \( \Psi(T^0) \leq \Psi(T^2) + \epsilon^2 K N_1 \). Since \( \Psi(T^2) \geq K\epsilon N_1 \), we know \( \Psi(T^0) \leq (1 + \epsilon)\Psi(T^0) \leq (1 + \epsilon)\Psi(T) \). \endproof
Figure 3. An illustrative example for the proof in Lemma 8 with $Z = 3$, while the $Y$-th group has 2 tasks only. From left to right, tasks are ordered from low utilization to high utilization. The largest task in the first group and the smallest task in the second group have the same utilization.

We conclude this section with the following theorem.

Theorem 6 Algorithm P-ROUNDING is an asymptotic polynomial-time approximation scheme for the multiprocessor task replication packing problem if $K$ is a constant.

Proof. It comes from Lemma 7 and Lemma 8 by taking $\epsilon$ as $0.5\delta$ for any user-specified positive value $\delta$ with $0 < \delta \leq 1$ to derive a solution with at most $(1 + \delta)\psi(T) + K$ processors. \qed

5 Conclusion

This paper develops approximation algorithms for task replication in identical multiprocessor environments. When the objective is to minimize the maximum utilization with a specified number of processors, we present an approximation algorithm with a 2-approximation ratio. The algorithm is then extended to a polynomial-time approximation scheme. For the minimization of the number of processors in the deriving of feasible schedules, a simple modification to the first-fit, best-fit, and worst-fit strategies are shown to have a 2-approximation ratio. An asymptotic polynomial-time approximation scheme is also developed. It guarantees that the number of processors used for the derivation of feasible schedules is at most $(1 + \delta)$ times the number of processors of an optimal solution plus $K$, where $K$ is a user-determined integer for fault tolerance, and $\delta$ is a positive parameter. The proposed (asymmetric) polynomial-time approximation schemes allow the system designer to trade the optimality of the derived solution with the analysis time.

We show in the example of Figure 2(c) that Algorithm GREEDY has better performance in the minimization of the maximum utilization of processors when tasks are ordered from a high utilization to a low utilization. The largest-utilization-first strategy is a $\frac{4}{3}$-approximation algorithm when no task replication is required [13]. However, it is still open whether the approximation ratio of the largest-utilization-first strategy can be better than that of Algorithm GREEDY when task replication is required. The proposed combinatorial algorithms can also be simply extended to consider tasks with different replication factors with the same worst-case guarantees. How to derive (asymmetric) polynomial-time approximation schemes for tasks with different replication factors is still unknown. For future research, we will further explore the capability of the largest-utilization-first strategy. We are also interested in the exploring of task replication strategy for heterogeneous multiprocessor systems with an arbitrary number of processors.

References