

# **Performance and Reliability-Driven Scheduling Approach for Efficient Execution of Parallelizable Stochastic Tasks in Heterogeneous Computing Systems**

**Ehab Y. Abdel Maksoud**

Department of Engineering Mathematics and Physics  
Faculty of Engineering, Cairo University, Egypt.  
e-mail: maksoudehab@yahoo.com

## **Abstract**

*In heterogeneous computing systems, a suite of different machines is interconnected to provide a variety of computational capabilities to execute collections of real-time application tasks that have diverse requirements. In these systems, machine failures are inevitable and can have an adverse effect on applications executing on the system. To reduce this effect, an efficient scheduling approach that is capable to assign tasks of an application to the suitable processors in these systems, to yield high performance and reliability of the system, must be devised. In this paper, we investigate a non-preemptive heuristic dynamic scheduling scheme for a set of independent stochastic tasks with parallelizable contents executing on a heterogeneous computing environment. Each stochastic task is characterized by its deadline and its time cost distribution, which is given in terms of mean and variance. A scheduled task may run as one unit on a single processor, or parallelized and its split tasks can be executed concurrently on multiple processors. We use this parallelism in tasks to meet their deadlines and also obtain better processor utilization compared to non-parallelized tasks. Non-preemptive parallelizable task scheduling combines the advantages of higher schedulability and lower scheduling overhead offered by the preemptive and non-preemptive task scheduling models, respectively. The scheduling algorithm proposed in this paper takes reliability measures into account, thereby enhancing the reliability of heterogeneous systems without any*

*additional hardware cost. To make scheduling results more realistic and precise, we exploits parallelism in tasks, if it is necessary, and distributes them among the available processors to maximize both of the reliability of the application and the performance measures of the system, such as the schedulability and the processor utilization, while maintaining lower scheduling overhead at the same time, compared to the previous task scheduling approaches.*

**Keywords:** *Heterogeneous computing environments, real-time applications, stochastic tasks, parallelizable tasks, parallel processing, stochastic scheduling, heuristics, processor utilization, reliability, schedulability.*

## 1 Introduction

Different activities of a computationally intensive real-time application often require different types of computations. However, numerous applications that have more than one type of embedded parallelism are now being considered for parallel implementation. In general, a given machine architecture with its associated compiler, operating system, and programming environment does not capable to satisfy all the requirements for all activities of an application equally well. With growing needs of building reliable real-time applications coupled with advancement of high-speed networks and high-performance machines, heterogeneous systems have been increasingly used for many real-time safety-critical applications in which the correctness of the systems depend not only on the results of a computation but also on the time at which these results are produced [1-2]. In heterogeneous computing environments, various resources have different capabilities to satisfy the requirements of varying application task mixtures. The efficient execution of applications in such environments requires effective scheduling strategies that take into account both algorithmic and architectural characteristics to achieve a good mapping of tasks to processors which is capable to maximize some required performance criterion. This means that existence of such strategies is an important component of these environments to maximize their performances.

Many real-time critical applications are composed of one or more independent tasks that need to perform their functions under strict timing constraints. These applications need to schedule dynamically their tasks with predictable performance. Thus, for predictable executions, schedulability analysis must be done before a task's execution is begun. This analysis requires a sufficient amount of information (execution time behavior and probability distribution) about the current set of tasks. Due to the real-time constraints on tasks, the proposed scheduling strategy for these tasks should be very efficient and accurate. This implies that the scheduling decisions, such as whether a task can be completed

before its deadline or not as well as which the most suited processor the task should be allocated, must be made efficiently. It is very difficult to estimate the actual execution time cost of different tasks in most of these applications [3-11]. Moreover, a task may have different execution times for different inputs. In addition, if we describe tasks by their arrival times, ready times, worst case computation times and deadlines, these characteristics must be taken into account during the schedulability analysis which makes the real – time scheduling problem harder than the non real–time scheduling [2-7]. On the other hand, in these applications, each task can be easily characterized as a stochastic task in terms of its time cost distribution and its deadline. The uncertainty nature of the task execution times and data transfer rates is usually neglected by traditional scheduling heuristics. These are the motivating factors to go in for a novel direction called the stochastic scheduling that investigate the problem of scheduling a set of tasks with random features [3-7, 11-13]. Common random features such as task processing times are usually modeled by specifying their probability distributions. Although a task's processing time is not known until it is complete, the probability distribution of task processing times are assumed to be known by the system a priori. A stochastic scheduling strategy which is a function of the time cost distribution and the deadline for each real-time task in the application can perform better than the previous simple heuristics [3-8, 11-13]. Knowing the time cost distributions for different application's tasks makes the scheduling decisions much faster and can guide the scheduling process efficiently to yield reasonable results [3-13]. Stochastic scheduling could be preemptive or non-preemptive, conduct on one or multiple processors, and be concerned with various optimization criteria.

Efficient application scheduling is critical for achieving high performance in heterogeneous computing environments. Because of its key importance on performance, there exists a large body of literature covering many task and parallel computer models. Many real-time heuristic scheduling algorithms have already been developed under different scenarios [3, 5-8, 13-21] which provide reasonable solutions. Most of these algorithms [3, 5, 6, 13-21] considered that each task can be executed on a single processor only. This may result in missing of task deadlines due to poor processor utilization. Moreover, tasks would miss their deadlines when their total computation time requirement is more than the deadline. On the other hand, execution of a task may be rejected, at the instant of its scheduling, since the available processing power for any processor in the systems does not enough to execute it to meet its deadline. These are the motivating factors to go in for parallelizable task scheduling strategy [2, 7, 22]. This strategy, considered in this paper, is an intermediate solution, between preemptive and non-preemptive scheduling approaches, which tries to meet the conflicting requirements of high schedulability and low scheduling overhead.

Most of existing real–time systems consider the performance measures such as schedulability as the main goal and ignore other effects such as machines failures [2-7, 11-22]. In a heterogeneous computing system, machine failure is

inevitable and can have an adverse effect on applications executing on the system [23-31]. To reduce that effect, we need an effective and efficient reliable scheduling algorithm which is capable to maximize not only the performance measures but also the probability that the system can run the entire application successfully, which is referred to as the system reliability. Recently, a wide variety of algorithms for reliable scheduling of tasks with precedence constraints on heterogeneous computing systems were developed under different scenarios [24-31]. However, since the only objective of this type of algorithms is to minimize the probability of failure of the application, it may produce task assignments that decrease the schedulability of the application [24, 29, 30]. As a result, there are usually conflicting requirements between maximizing the schedulability of the system and minimizing the probability of failure of an application, and it may not be possible to satisfy both objectives at the same time. Consequently, an effective reliable scheduling algorithm for real-time applications which take into account both schedulability and reliability must be devised. Parallelizing tasks in these applications, if it is necessary, and distributing its split tasks to run among different processors at the same time may achieve better reliability of the application, in addition of yielding higher schedulability and utilization for the system. This has great influence to make the application subject to the system failure [23, 25].

In this paper, a new approach is developed for scheduling an application represented by a set of independent stochastic tasks on a heterogeneous computing system, where the reliability measurements of different processors is incorporated into a heuristic scheduling algorithm to take failures into account while making scheduling decisions. This approach exploits parallelism in tasks, especially those with higher expected time cost or with tight laxities, to improve the resulting performance measures and the reliability of the system. The major contribution of this work is explicitly using a partitioning approach to divide given tasks as needed into parallel parts to improve the capabilities of the scheduling process. In addition, it extends the traditional formulation of the stochastic scheduling problem so that both the system performance and the reliability of applications are simultaneously accounted for. Simulation studies are used to evaluate the merits of the proposed algorithm and its effectiveness to produce a better schedulability and reliability relative to that of the other compared scheduling algorithms.

The rest of this paper is structured as follows. Section 2 describes the modeling approach and defines the problem; section 3 gives some basic definitions and presents the proposed reliable scheduling algorithm; section 4 includes the performance results and comparisons with other scheduling algorithms; and finally section 5 summarizes the paper conclusions.

## 2 Model Description and Problem Formulation

The proposed scheduling model consists of a target architecture heterogeneous computing environment, a real-time application, and proposed feasible performance criteria for scheduling. Our model considers a typical heterogeneous system structure consists of a set of parallel  $m$  independent different heterogeneous processors,  $P = \{P_1, P_2, \dots, P_m\}$ , having different processing powers,  $P_{P_j}$ ,  $j = 1, 2, \dots, m$ . Each of these processors has its own dispatch queue and local memory, as shown in Figure 1.

The workload of the real-time application consists of a set of  $n$  independent stochastic tasks. The number of these tasks is much greater than the number of the available processors in the system ( $n > m$ ). For the application model we assume:

1. Each task  $T_i$ ,  $i = 1, 2, \dots, n$ , is characterized by its deadline  $d_i$ , its time cost distribution and the required processing power. In the normal case, when a task  $T_i$  runs on one processor, the time cost distribution of the task is known with mean,  $\mu_i$ , and variance,  $V_i$ , and the required processing power  $\omega_i$ . Moreover, for the pessimistic case, when the task  $T_i$  is partitioned into  $k$  parallel parts running on  $k$  different processors, the worst case mean and variance  $\mu_i^k$  and  $V_i^k$ , are known a priori, where  $k$  is the degree of parallelism of the task, that varies from 2 to  $m$  (or to the max-split number ( $< m$ ) which can given as an input parameter).
2. For each task  $T_i$ , the worst case mean and variance for any  $j$  and  $k$  with  $j < k$  satisfy the sub-linear speedup assumptions where  $j * \mu_i^j \leq k * \mu_i^k$  and  $j * V_i^j \leq k * V_i^k$ , respectively [2, 7, 22]. The sub-linearity is due to the overheads associated with communication and synchronization among the split tasks of a task.
3. When a task is parallelized, all of its parallel subtasks, also called split tasks, have to start simultaneously in order to synchronize their executions.
4. Tasks are non-preemptive, i.e., once a task or a split task starts execution, it finishes to its completion.
5. The processing power for each available processor in the system  $P_j$ ,  $j = 1, 2, \dots, m$  is enough to execute any task  $T_i$ ,  $i = 1, 2, \dots, n$ , i.e.  $P_{P_j} \geq \omega_i, \forall i, j$ .

In the proposed model, all tasks arrive from a task generator through a general task queue (Q) and get fed into a central processor called the system processor or the scheduler from where they are distributed to the dispatch queues,  $Q_1, Q_2, \dots, Q_m$ , for all of the available processors in the system for execution, as shown in Figure 1. We assume that all of these tasks are synchronous, i.e. their first request arrives simultaneously at the time zero. Since the task set cannot be predicted, task allocation and scheduling must be done on-line and use up-to-date information for the scheduling activities during the system

execution. Although some tasks may miss their deadlines, the scheduler follows a task-skipping scheme [32, 33] to ensure that the firm guarantees of the new task will be met.

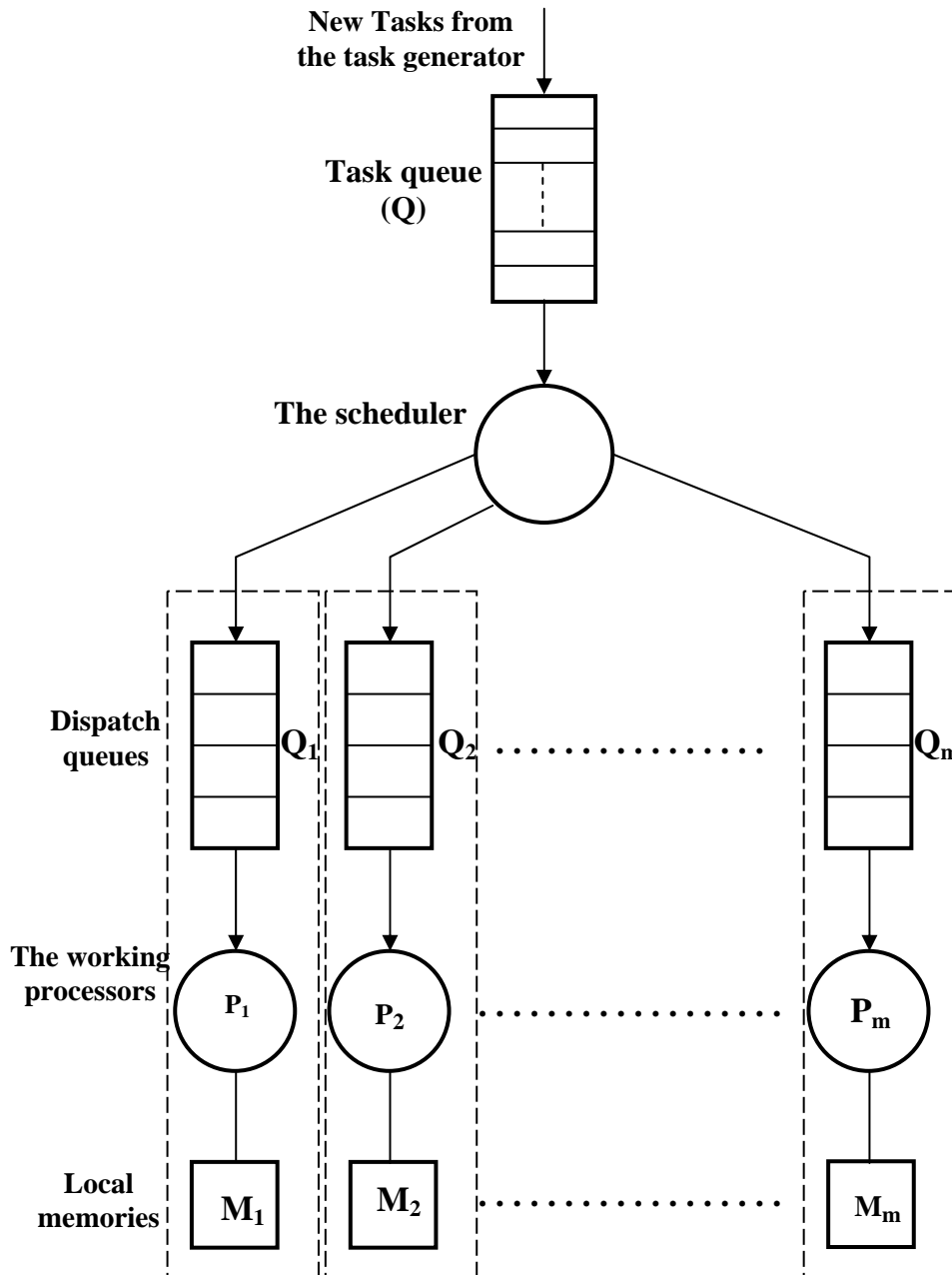


Fig. 1: The target computing environment.

For each candidate task, an acceptance test has to be carried out by the scheduler, to determine whether its real-time performance requirements can be met by executing it as one unit on a single processor or partitioned into  $k$  split tasks running on  $k$  different processors simultaneously. If so, the candidate task is

allocated to the dispatch queue(s) of the suitable application processor(s), which has been identified as having sufficient resources to schedule this task (or split tasks) without compromising the minimum performance guarantees that have been provided to the tasks which are already present. A task whose time constraints are not guaranteed can be rejected. The scheduler will run in parallel with the applications processors, scheduling the new ready tasks, from the task queue (Q), and periodically updating the dispatch queues. This organization ensures that the processors will always find some tasks in the dispatch queues when they finished their current tasks. The process of task allocation and scheduling takes (as input) the stochastic task set that represents the real-time application under consideration and the target architecture specified by the heterogeneous multiprocessor environment. It then attempts to obtain a suitable mapping of the tasks to the available processors that is capable to offer better schedulability for tasks while maintaining lower scheduling overhead.

Under the previous assumptions, the proposed model is useful to deal with the scheduling problem for real-time applications that require predictable performance in advance. In a previous work [5-7], we presented a heuristic approach, called Performance Efficient Stochastic Scheduling (PESS) algorithm, to schedule a set of independent stochastic real-time tasks on a homogenous multiprocessor system. This work assumed that each task is characterized by its deadline and its time cost distribution which is given in the form of its mean and variance. The algorithm was proposed to yield the maximum possible probability for each task to meet its deadline. The problem being addressed in this paper is concerned with allocating a set of stochastic tasks of a parallel application onto a heterogeneous computing system. For this type of systems, we must take into account and increase the potential for system failures. Maximizing reliability of such systems is of critical importance along with the task scheduling. A system may execute an application with high reliability if the application tasks are assigned carefully onto the appropriate processors in the system taking into account the failure rates of processors. Therefore, the objective of our work is to present a simple heuristic scheduling algorithm which accounts for maximizing both the reliability of the application and yielding the required probability for each task in this application to meet its deadline, at the same time. This algorithm exploits parallelism in tasks whenever needed to satisfy the required objectives and improve the previous results.

### **3 Performances and Reliability-Driven Scheduling Algorithm for Stochastic Tasks with Parallelizable Contents**

#### **3.1 Terminology**

**The Central Limit Theorem** [34]. If  $T_1, T_2, \dots, T_n$  are  $n$  independent random variables with means,  $\mu_i$ , and variances,  $V_i$ ,  $i = 1, 2, \dots, n$ , then the random variable

$$S = \sum_{i=1}^n T_i, \text{ follows approximately a normal distribution with mean } \mu_S = \sum_{i=1}^n \mu_i, \text{ and variance } V_S = \sum_{i=1}^n V_i.$$

Based on the previous theorem and by considering the proposed parallel model, we can state the following basic definitions and important concepts which are necessary to explain the proposed scheduling approach.

**Definition 1:** Since the time cost distribution,  $T_{P_j}$ , for the current load of any processor,  $P_j$ ,  $j = 1, 2, \dots, m$ , is given by the sum of the of several independent tasks or split tasks, it follows approximately a normal distribution with mean,  $\mu_{P_j}$ , and variance,  $V_{P_j}$ , which are defined as the sum of the means and the variances for the time cost distributions of its assigned tasks or split tasks, respectively, including the current one.

**Definition 2:** The laxity,  $L_i$ , for a task,  $T_i$ , with expected time cost,  $\mu_i$ , and deadline,  $d_i$ , is defined as

$$L_i = d_i - \mu_i \quad , \quad i \in \{1, 2, \dots, n\} \quad (1)$$

**Definition 3:** A processor reliability,  $R_{P_j}$ , is the probability that the processor,  $P_j$ ,  $j \in \{1, 2, \dots, m\}$ , is operational during the execution of all tasks or split tasks assigned to it. Therefore, it can be defined in an exponential form [23, 25, 28], as follows

$$R_{P_j} = \exp(-\lambda_{P_j} \mu_{P_j}) \quad (2)$$

where  $\lambda_{P_j}$  is the failure rate of the processor  $P_j$  and  $\mu_{P_j}$  is the total expected time cost of all tasks or split tasks assigned to that processor. Without loss of generality, we can assume that the processor failure rates are constant and independent of each other. This assumption is common to other studies which deal with analyzing the reliability of real-time systems [23-31].

**Definition 4:** For a heterogeneous computing system with  $m$  processors, the reliability of an application,  $R_A$ , is defined as the probability that none of the system components (processors) fails while processing the given application tasks [23, 27, 29]. In other words,  $R_A$  is the product of the reliability for all the system



processors, since failures of these processors are assumed to be statistically independent [23, 25, 27, 29]. Hence, it can be formulated as follows

$$R_A = \prod_j R_{P_j} = \exp\left(-\sum_{j=1}^m \lambda_{P_j} \mu_{P_j}\right) \quad (3)$$

**Definition 5:** The average expected time cost,  $\bar{\mu}_{P_j}$ , and variance,  $\bar{V}_{P_j}$  of the proposed load for the processor  $P_j$ ,  $j = 1, 2, \dots, m$ , based on its processing power,  $P_{P_j}$ , is defined by

$$\bar{\mu}_{P_j} = \frac{\sum_{i=1}^n \mu_i}{m} \cdot \frac{P_{P_j}}{\sum_{j=1}^m P_{P_j}} \quad (4)$$

$$\text{and } \bar{V}_{P_j} = \frac{\sum_{i=1}^n V_i}{m} \cdot \frac{P_{P_j}}{\sum_{j=1}^m P_{P_j}} \quad (5)$$

and the corresponding proposed reliability,  $\bar{R}_{P_j}$ , for this processor is given by

$$\bar{R}_{P_j} = \exp(-\lambda_{P_j} \bar{\mu}_{P_j}) \quad (6)$$

Consequently, the proposed reliability of an application,  $\bar{R}_A$ , can be defined as follows

$$\bar{R}_A = \prod_j \bar{R}_{P_j} = \exp\left(-\sum_{j=1}^m \lambda_{P_j} \bar{\mu}_{P_j}\right) \quad (7)$$

Assume that the task,  $T_i$  (or a split task  $T_i^k$ ),  $i \in \{1, 2, \dots, n\}$ , with deadline,  $d_i$ , is allocated to a processor,  $P_j$ ,  $j \in \{1, 2, \dots, m\}$ , and  $T_{P_j}$  be the time cost distribution for the current load of that processor, including the last task or split task, with mean,  $\mu_{P_j}$ , and variance,  $V_{P_j}$ .

**Definition 6:** Any processor in the system,  $P_j$ ,  $j = 1, 2, \dots, m$ , is said to be underloaded if  $\mu_{P_j} + \sigma_{P_j} < \bar{\mu}_{P_j} + \bar{\sigma}_{P_j}$ , where  $\sigma_{P_j} = \sqrt{V_{P_j}}$  is the standard deviation of the current load of the processor  $P_j$  and  $\bar{\sigma}_{P_j} = \sqrt{\bar{V}_{P_j}}$  is the standard deviation

of its proposed load. With equal constraint, the processor takes its full load. Otherwise, the processor is overloaded.

**Definition 7:** The **reliable processor set** is the set that contains all processors with current reliability less than their proposed reliability, i.e., all processors,  $P_j, j \in \{1, 2, \dots, m\}$ , that satisfy the condition

$$R_{P_j} \leq \bar{R}_{P_j} \quad \text{or equivalently} \quad \mu_{P_j} \leq \bar{\mu}_{P_j} \quad (8)$$

Processors in this set are ordered in a descending order of their reliabilities, i.e., in ascending order of their current expected time costs.

**Definition 8:** The probability that the processor  $P_j$  completes the execution of the task  $T_i$  (or the split task  $T_i^k$ ) before its deadline,  $d_i$ , is defined as

$$\begin{aligned} P(T_{P_j} \leq d_i) &= P\left(\frac{T_{P_j} - \mu_{P_j}}{\sigma_{P_j}} \leq \frac{d_i - \mu_{P_j}}{\sigma_{P_j}}\right) \\ &= P(Z \leq \beta) \end{aligned} \quad (9)$$

where the variable  $Z = \frac{T_{P_j} - \mu_{P_j}}{\sigma_{P_j}}$  is the standard normal variable and

$\beta = \frac{d_i - \mu_{P_j}}{\sigma_{P_j}}$  is a positive constant, since  $d_i$  must be greater than  $\mu_{P_j}$ .

If we need to make this probability at least equal to a certain value, say  $\alpha$ , then from (9) we must have

$$P(Z \leq \beta) \geq \alpha \quad (10)$$

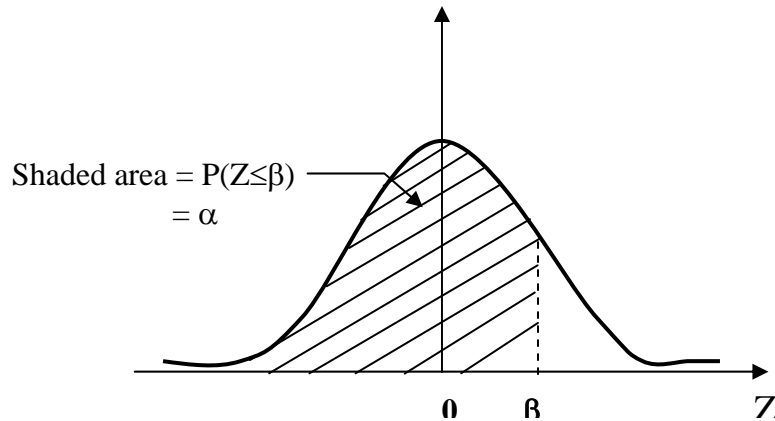
The Z-value,  $\beta$ , corresponding to the required probability,  $\alpha$ , with equal constraint, can be found directly in the body of the standard normal tables [6, 34]. The requested value is shown graphically in Figure 2. This figure shows that we can obtain the required probability if

$$\frac{d_i - \mu_{P_j}}{\sigma_{P_j}} \geq \beta \quad \text{or equivalently} \quad \left[ \frac{d_i - \mu_{P_j}}{\beta \sigma_{P_j}} \right] \geq 1 \quad (11)$$

Therefore, executing the task  $T_i$  (or the split task  $T_i^k$ ) on the processor  $P_j$ , that

satisfies the condition  $\left[ \frac{d_i - \mu_{P_j}}{\beta \sigma_{P_j}} \right] \geq 1$ , is a sufficient condition to guarantee the

required probability of this task to meet its deadline.



**Fig. 2: Determining the z-value to meet a specified probability.**

**Definition 9:** The **candidate processor set** is a subset of the reliable processor set that contains all processors,  $P_j$ ,  $j \in \{1, 2, \dots, m\}$ , which satisfy the condition,  $\mu_{P_j} \leq d_i$ .

**Definition 10:**  $FTP_j$  is the finishing time for the last task (or split task) in the processor  $P_j$ ,  $j=1, 2, \dots, m$ . The distribution of this time has the mean  $\mu_{FTP_j}$  and the variance  $V_{FTP_j}$ , which are defined respectively, as the sum of the means and the variances for the time cost distributions of all tasks or split tasks that are scheduled on that processor in the proposed scheduling.

**Definition 11:**  $F_{TS}$  is the finishing time of a schedule (application) which is the time required for the last processor to finish executing its assigned task (or split task). The distribution of this time is equal to the finishing time distribution of the processor, say  $P_k$ , that has maximum mean. This time cost has the mean  $\mu_{F_{TS}}$  and the variance  $V_{F_{TS}}$  where

$$\mu_{F_{TS}} = \max_{P_j} \mu_{FTP_j} = \mu_{FTP_k} \quad \text{and} \quad V_{F_{TS}} = V_{FTP_k} \quad (12)$$

### 3.2 The proposed algorithm

In this section, we introduce a performance and reliability-driven approach for stochastic tasks with parallelizable contents to allocate and schedule a set of independent stochastic tasks comprising a real-time application onto a heterogeneous computing system. In the context of this paper, the proposed algorithm has complete knowledge about the currently active set of tasks, but not about the new tasks that may arrive while scheduling the current set. The design of a real-time scheduling scheme requires simultaneous consideration of the

multiple contradictory objectives. Due to the nature of the real-time applications and under the assumption that tasks are stochastic [5-7, 11-13, 25], our scheduling approach must satisfy the following requirements:

1. Yielding a certain required probability for each task to meet its deadline.
2. Providing a high level reliability while executing the application.
3. Maximizing the utilization for the available processing power of different processors to increase the rate of applications admitted to the system without any delay and avoiding (reducing) the fragmentation of this power, as much as possible.
4. Minimizing the inter-communication among the sequential tasks or split tasks.
5. Producing a well-balanced load to each of the available processor in the system, relative to its associated processing power, to minimize the total expected time cost of the schedule (application).
6. Keeping the degree of parallelism as specified in the application's task graphs and choosing the number of split tasks as small as possible, to decrease scheduling overheads.

For real – time systems, we must take into consideration the system failure. A system may execute an application with high reliability if the application tasks are assigned carefully onto the appropriate processors in the system taking into account the failure rates of processors. The proposed algorithm parallelizes a task whenever its deadline cannot be met with the required probability. The degree of parallelization,  $k$  (i.e., the number of split tasks) of a task is chosen in such a way that the task's deadline is just met with this probability. Knowing the time cost distributions of different tasks, in both normal and pessimistic cases, can be used as a good tool to guide the scheduling process efficiently. The statistical properties and the probability rules can help to find the sufficient conditions that maintain the required probability for each task. Before applying this algorithm, we must evaluate the expected time cost,  $\bar{\mu}_{P_j}$ , and the variance,  $\bar{V}_{P_j}$ , of the proposed load for each available processor in the system,  $P_j$ ,  $j = 1, 2, \dots, m$ , and its corresponding proposed reliability,  $\bar{R}_{P_j}$ , to guide selection of suitable candidate processor(s) during the scheduling process. In addition, evaluating  $\bar{\mu}_{P_j}$  and  $\bar{V}_{P_j}$  can be used to check the current load of any processor, after allocating a task or a split task to it, in order to know whether that processor takes its proposed load or remains under loaded, to maintain the required load balancing. If the processor takes its proposed load, the scheduler should stop assigning tasks to this processor. On the other hand, the under loaded processors can continue accepting more tasks as assigned by the scheduling process. In the task queue (Q), the current tasks are sorted in ascending order of their deadlines. Tasks with the same deadlines are sorted in ascending order of their laxities, i.e., in a descending order

of their expected time costs. In the case of several possibilities, tasks are sorted in a descending order of their variances. Executing the task with the maximum expected time cost and the maximum variance, out of tasks with the same deadline may increase its probability to meet its deadline and has great influence on maximizing the overall reliability of the system to execute the application. After arranging tasks, we allocate the first  $m$  tasks in the resulting list, one to a different available processor in the system. For the remainder tasks,  $T_i, i \in \{m, m+1, \dots, n\}$ , the proposed algorithm builds their allocation and scheduling step by step by extending the previous partial schedule by one task at a time. At each step, the candidate task in the current task queue is selected and a decision is made on which that task needs parallelization or not, and on to which processor(s) the task (or its split tasks) should be mapped. Our main mission in taking this decision is to select the most reliable processor that guarantees the required probability for the candidate task to meet its deadline. It is important to notice that using the available processors efficiently, especially when parallelization for the candidate task is necessary, has a great influence to maximize this probability. On the other hand, poor processor utilization may result in missing of task deadlines.

The proper choice of the candidate processor is made as follows. The candidate task  $T_i$ , with deadline,  $d_i, i \in \{m, m+1, \dots, n\}$ , is added to the current load of each under loaded processor in the system,  $P_j, j \in \{1, 2, \dots, m\}$ . For the resultant time cost distribution of each processor, we evaluate the expected time cost,  $\mu_{P_j}$ , and the variance,  $V_{P_j}$ . Then, we can construct the reliable processor set which contains all processors that is capable to execute this task and satisfy the condition  $R_{P_j} \leq \bar{R}_{P_j}$  or equivalently  $\mu_{P_j} \leq \bar{\mu}_{P_j}, j \in \{1, 2, \dots, m\}$ . Processors in this set are ordered in ascending order of their current expected time costs, i.e., in a descending order of their reliabilities. Finally, we construct the candidate processor set that contains all processors, in the reliable processor set, which satisfy the condition:  $\mu_{P_j} \leq d_i$ . Let the first processor in the candidate processor

list is  $P_f, f \in \{1, 2, \dots, m\}$ . If the ratio  $\left[ \frac{d_i - \mu_{P_f}}{\beta \sigma_{P_f}} \right] \geq 1$ , the candidate task  $T_i$  do not

need to be parallelized, as illustrated in definition 8, and scheduling it on the processor  $P_f$  can produce the required probability of this task to meet its deadline. If the previous ratio is less than one, the candidate task  $T_i$  must be parallelized. In this case, the number of split tasks,  $k$ , must be chosen as small as possible, to avoid the idle times of the candidate processors and decrease scheduling overheads. Therefore, we search for the smallest number of processors,  $k$  processors,  $k \in \{2, 3, \dots, \max-split\}$ , such that the  $k^{\text{th}}$  processor in the current

candidate processor list, say  $P_r$ ,  $r \in \{1, 2, \dots, m\}$ , satisfy the condition  $\left[ \frac{d_i - \mu_{P_r}}{\beta \sigma_{P_r}} \right] \geq 1$ ,

when the  $k^{\text{th}}$  split task  $T_i^k$  is added to the current load of this processor. Then, the split tasks for the candidate task  $T_i$  are scheduled on the first  $k$  processors in the candidate processor list. If the previous set is empty, i.e.,  $\mu_{P_j} > d_i, \forall j \in \{1, 2, \dots, m\}$ , the deadline of the candidate task cannot be met

using any degree of parallelization up to max-split. In this case, the time constraints of the candidate task are not guaranteed and it must be rejected. The previous process is repeated again for every remaining task until all tasks in the current set have been allocated and scheduled on the different available processors in the system. Finally, we evaluate the expected time cost and the variance for the finishing time distribution of each processor in the system. Then, choose the finishing time distribution of the processor with the maximum expected time cost to be the suitable distribution for the finishing time of the schedule, FTS. If more than one distribution has the same maximum, choose the one with maximum variance. The time complexity of the proposed algorithm for scheduling  $n$  tasks is  $O(mn)$ .

Scheduling using this strategy will increase the likelihood of mapping to the most reliable processor(s) in the system being able to execute the candidate task (or its split tasks) with the required probability to meet its deadline. Moreover, it is obvious to notice that the selected candidate processor(s), according to the previous approach, is the processor(s) with the minimum current load(s), which can help to yield a well-balanced load to each of the available processor in the system, relative to its associated processing power. This will maximize the processor utilization and hence minimizes the total expected time cost of the whole application. In addition, using the proposed reliabilities, or equivalently the proposed expected time costs, of different processors to guide the scheduling process can help to maximize the overall reliability of the application.

Given the max-split number and the required probability of each task to meet its deadline, is at least  $\alpha$ , as input parameters, the proposed algorithm can be written in steps as follows:

1. Evaluate the Z-value, say  $\beta$ , corresponding to the required probability from the standard normal tables.
2. Sort the current tasks in the task queue (Q), in ascending order of their deadlines. If more than one task has the same deadline, evaluate the laxity  $L_i, i \in \{1, 2, \dots, n\}$ , for these tasks and sort them in ascending order of their laxities. Sort the tasks with the same deadlines and laxities in a descending order of their variances.
3. Set three pointers  $i, j$  and  $k$ , to indicate the task index in the resulting task queue, the processor index, and the number of split tasks permitted for the

- candidate task, respectively. The pointer  $i$  changes from 1 to  $n$ ,  $j$  changes from 1 to  $m$ , where  $n > m$ , and  $k$  changes from 2 to max-split .
4. Evaluate the expected time cost and the variance of the proposed load for each processor in the system,  $\bar{\mu}_{P_j}$  and  $\bar{V}_{P_j}$ ,  $j = 1, 2, \dots, m$ .
  5. For  $i = 1$  to  $m$ , allocate a task  $T_i$  to a different processor  $P_j$ ,  $j \in \{1, 2, \dots, m\}$ .
  6. Initialize the pointer  $i$  to be  $m+1$ .
  7. Add the time cost distribution of the candidate task  $T_i$ , that has the mean,  $\mu_i$ , the variance,  $V_i$ , and the deadline,  $d_i$ , to the current load for each under loaded processor,  $P_j$ ,  $j \in \{1, 2, \dots, m\}$ .
  8. Evaluate the expected time cost,  $\mu_{P_j}$ , and the variance,  $V_{P_j}$ , for the resultant time cost distribution of each processor  $P_j$ ,  $j \in \{1, 2, \dots, m\}$ .
  9. Construct the reliable processor set which contains all processors that satisfy the condition  $\mu_{P_j} \leq \bar{\mu}_{P_j}$ . Processors in this set are sorted in ascending order of their current expected time costs. Order the processors of the same expected time costs in ascending order of their variances.
  10. Construct the candidate processor set which contains all processors, out of that selected in the reliable processor set, that satisfy the condition  $\mu_{P_j} \leq d_i$ .
    - 10.1 If the resultant set contains only one processor, select this processor to be the candidate processor.
    - 10.2 If the set contains more than one processor,  $q$  processors, and  $P_f$ ,  $f \in \{1, 2, \dots, q\}$ , is the first processor in the candidate processor list, evaluate the ratio  $\left[ \frac{d_i - \mu_{P_f}}{\beta \sigma_{P_f}} \right]$  for the current time cost distribution of the processor  $P_f$ .
    - 10.3 If the ratio  $\left[ \frac{d_i - \mu_{P_f}}{\beta \sigma_{P_f}} \right] \geq 1$ , parallelization for  $T_i$  is not necessary. Schedule the candidate task  $T_i$  on the processor  $P_f$ .
    - 10.4 If  $\left[ \frac{d_i - \mu_{P_f}}{\beta \sigma_{P_f}} \right] < 1$ , the candidate task  $T_i$  must be parallelized.

- 10.4.1 Add the  $k^{\text{th}}$  split task of the candidate task  $T_i$ , with mean  $\mu_i^k$  and variance  $V_i^k$ ,  $k \in \{2,3,\dots,q(\text{or max-split})\}$  to the  $k^{\text{th}}$  processor in the current candidate processor list, say  $P_r$ ,  $r \in \{1,2,\dots,q\}$ .
- 10.4.2 Evaluate the expected time cost,  $\mu_{P_r}$ , and the variance,  $V_{P_r}$ , for the resultant time cost distribution of the processor  $P_r$ .
- 10.4.3 Find the smallest value of  $k$ ,  $k \in \{2,3,\dots,q(\text{or max-split})\}$ , such that the ratio  $\left[ \frac{d_i - \mu_{P_r}}{\beta \sigma_{P_r}} \right] \geq 1$
- 10.4.4 If such  $k$  exists, schedule the  $k$  split tasks of  $T_i$  on the first  $k$  processors in the current candidate processor list.
- 10.4.5 If  $\left[ \frac{d_i - \mu_{P_r}}{\beta \sigma_{P_r}} \right] < 1 \forall k \in \{2,3,\dots,q(\text{or max-split})\}$ , drop the task  $T_i$  from the current task set and reject its execution.
- 10.5 In the case of empty candidate processor set, i.e., all processors in the candidate processor set have  $\mu_{P_j} > d_i$ , drop the task  $T_i$  from the current task set and reject its execution.
- 11 After allocating and scheduling the candidate task (or its split tasks), update the expected time cost(s) and the variance(s) for the current load(s) of the selected processor(s).
- 12 Check for the selected processor(s) load(s). If  $\mu_{P_r} + \sigma_{P_r} < \bar{\mu}_{P_r} + \bar{\sigma}_{P_r}$ ,  $r \in \{1,2,\dots,q\}$ , the processor  $P_r$  is under loaded and can cooperate in the next steps. Otherwise, discard the processor  $P_r$  from the scheduling process in the next steps, since it takes its proposed average load.
- 13 Increment  $i$  and repeat steps 6 to 12 until  $i = n$ , i.e., until all tasks in the current task queue are scheduled on different processors in the system.
14. Evaluate the finishing time distribution of the processor with the maximum mean to be the distribution for the finishing time cost of the schedule, FTS. In the case of several possibilities, choose the one with maximum variance.

## 4 Simulation Study

To evaluate the effectiveness of task parallelization in yielding better schedulability and reliability, we conducted extensive simulation study on the



parallel real-time system model, which was illustrated in section 2. In this study, we compare the performance of the proposed approach, the Performance and Reliable-driven scheduling for Parallelizable Stochastic tasks (PRPS) algorithm, with the Reliability driven Real Time Periodic scheduling (RRTP) algorithm [28] and the (Repars) algorithm [31]. Both of the compared algorithms schedule a set of periodic real-time tasks on heterogeneous multiprocessor systems. As mentioned earlier, we are concerned in whether or not the proposed algorithm produces an appropriate reliability of a task set (an application) and all the tasks in this set (application) can finish before their deadlines with an acceptable probability, taking into account minimizing the scheduling overheads. Therefore, three major metrics were used to evaluate the performance of this algorithm. The first metric is the reliability ratio (RR) which is defined as the ratio of the number of task sets found schedulable by the proposed algorithm, with an acceptable overall reliability, to the number of task sets considered for scheduling. In this study, we consider that the reliability of a task set is accepted, if it is equal to  $(\bar{R}_A - \epsilon)$ , where  $\bar{R}_A$  is the proposed reliability of the application and  $\epsilon$  is a pre specified acceptable deviation factor, which was assumed to be 0.25. The second metric is the success ratio (SR) which is defined as the ratio of the number of task sets found schedulable by the proposed algorithm, with an acceptable probability of each task to meet its deadline, compared to the number of task sets considered for scheduling. For this study, we considered that the probability of a task to meet its deadline is accepted if it is at least equal to 0.75, i.e.  $\alpha \geq 0.75$ . Finally, the third metric is the scheduling overhead ratio (OHR) which is defined as the number of scheduling steps performed by proposed (PRPS) algorithm relative to that performed by the (RRTP) or the (Repars) algorithm to schedule the same task set (application).

We ran simulations for different random task test sets that model real-time applications with different means and variances, to represent the time cost distributions of different stochastic tasks in these sets. Schedulable task sets are generated for simulation using the following approach:

1. Tasks of the task test set (or the application) are generated till schedule length, which is an input parameter, with no idle time in the processors, as described in [2, 4, 7, 14].
2. Number of tasks,  $n$ , in each task set is uniformly distributed varies between 50 to 700 tasks.
3. Each of the mean and variance,  $\mu_i$  and  $V_i$  of the time cost distribution for the task  $T_i$ ,  $i \in \{1, 2, \dots, n\}$ , in the normal-case, were chosen uniformly distributed between two input parameters,  $(Min\_M, Max\_M)$  and  $(Min\_V, Max\_V)$ , respectively.
4. In the worst-case, the allowable maximum degree of parallelization of a task, max-split number, varies from 2 to 16.

5. The worst-case means and variances,  $\mu_i^k$  and  $V_i^k$ , when a task  $T_i$  is executed on  $k$  processors in parallel,  $2 \leq k \leq \text{max-split number}$ , were evaluated by using the sub-linear relations:

$$\mu_i^k = \left\lfloor \mu_i^{k-1} * (k-1)/k \right\rfloor + 1 \quad \text{and} \quad V_i^k = \left\lfloor V_i^{k-1} * (k-1)/k \right\rfloor + 1.$$

6. The deadline,  $d_i$ , for each task,  $T_i, i \in \{1, 2, \dots, n\}$ , is chosen such that  $d_i = (1 + R) * \mu_i$ , where  $R$  is the laxity parameter, which varies from 1.5 to 3.5, that denotes the tightness of the deadline [2, 4-7, 13, 31], and  $\mu_i$  is the proposed mean of the task  $T_i$ . In other words, we considered the laxity,  $L_i$ , of each task,  $T_i$ , equal to  $R * \mu_i$ .

On the other hand, we considered a target computing heterogeneous system with  $m$  processors. The number of processors varies from 4 to 40. The failure rate of each processor,  $\lambda_{p_j}, j = 1, 2, \dots, m$ , was assumed to be uniformly distributed between 0.001 and 0.0001 failures/hr [25-31]. In each test case, we assumed that the total number of tasks is greater than the number of the available processors in the system ( $n > m$ ). In addition, we considered that the processing power of any processor in the system is greater than that required by any task in the test set.

Our experimental work showed that the laxity parameter,  $R$ , the number of the available processors in the system,  $m$ , and the maximum allowable degree of parallelization of a task, the max-split, have the greatest effect on the performance of the proposed approach. Results of the simulation study showed that the introduction of the parallelization action in the scheduled tasks, if it is necessary, leads to a significant improvement in the performance measures of the system such as the schedulability and the processor utilization, for all cases in this study over our previous approach, Performance Effective Stochastic Scheduling (PESS) algorithm [5-7]. In addition, these results show that the average performance provided by the proposed algorithm, in terms of the success ratio SR and the reliability ratio RR, provided by the proposed approach is always greater than that of the other compared algorithms in all cases of this study. Figures 3,4,5 and 6,7,8 show results of simulations that represent, respectively, the reliability ratio RR and the success ratio SR offered by the proposed algorithm (PRPS) over the (RRTP) algorithm and the (Repar) algorithm, for different values of the laxity parameter,  $R$ , different numbers of processors,  $m$ , and different values of max-split number. Each point in the performance curves is the average of 5 simulation runs, each with 300 task sets. Each task set contains approximately 50-700 tasks by fixing the schedule length to 800 during the task set generation.

The effect of the laxity parameter ( $R$ ) is studied in Figures 3 and 6. As the laxity parameter increases, the deadline of the candidate tasks increases, the success ratio for all algorithms increases in all cases. From Figures 3 and 6, the difference in success ratio (SR) and the reliability ratio (RR) between the proposed algorithm (PRPS) and the other algorithms for lower laxities is high and

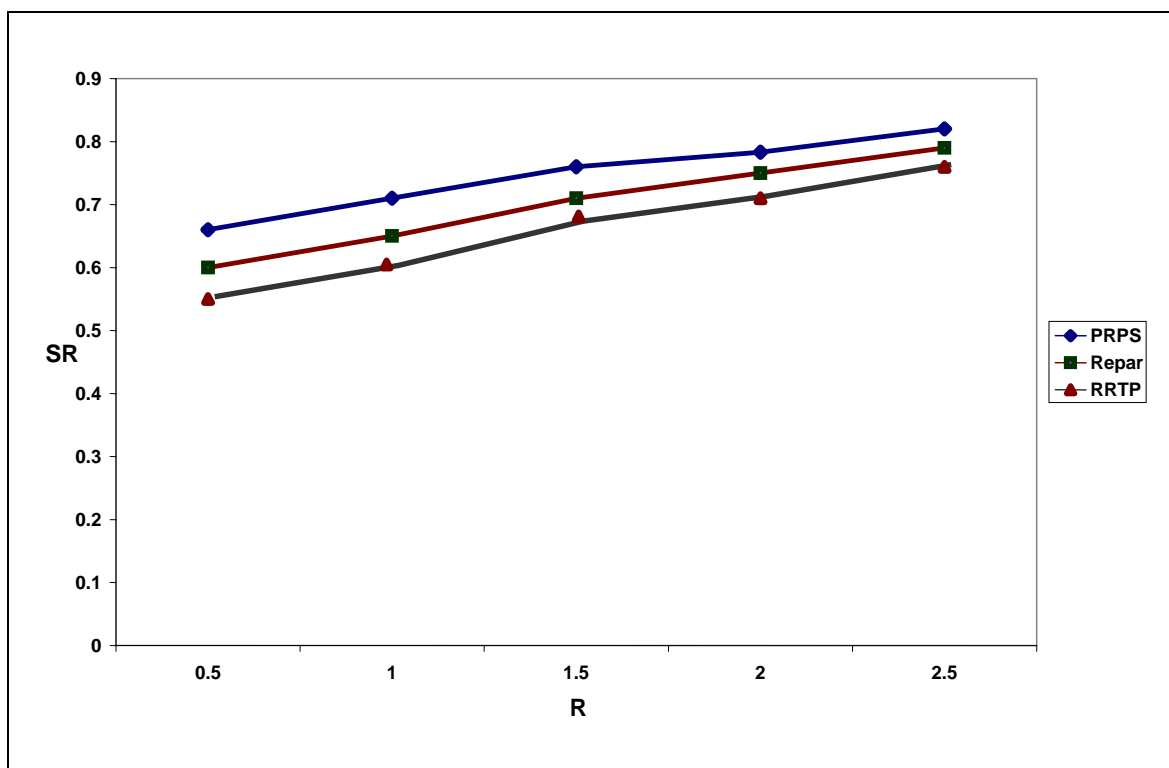
decrease with increasing laxity until no improvement, approximately, can be obtained when  $R > 2.5$ . When  $R < 1.5$ , the value of (RR) can be significantly increased by more than 15%. This is due to the fact that tasks experience more degree of parallelization (in order to meet their deadlines) when their laxities (deadlines) are tight, but the same tasks with higher laxities rarely need parallelization since their deadlines can be met without parallelizing them. This shows that task parallelization is more effective for tasks having tighter laxities.

The effect of varying the number of processors ( $m$ ) is studied in Figures 4 and 7. The increase in the number of processors increases the success ratio and the reliability ratio for all algorithms, and vice versa. The difference in success ratio of all algorithms for two successive values of  $m$ , i.e.  $m$  and  $m+1$ , is very high when  $m$  is small, and decreases as  $m$  increases. This is because the limited availability of resources, i.e., the bottleneck is the resources and not the processors. This means that if ( $m$ ) is increased beyond 24, there cannot be much improvement in the success ratio. On the other hand, while the number of processors increases, the proposed load for each processor, based on its processing power, decreases as expected. In parallel to this decrease, the failure probability of applications also decreases, i.e., the reliability of applications in terms of the resultant success ratio increases.

From Figures 5 and 8, it is an interesting to note that, in general, an increase in degree of parallelization, max-split, increases the success ratio and the reliability ratio offered by the proposed algorithm (PRPS) over the other compared algorithms. It is clear that lower values of max-split are more sensitive to change in laxity parameter  $R$  (deadline) than higher values of max-split. For tasks with tight laxities, the growth in the value of max-split has great influence to help them to meet their deadline with the required probability. Executing tasks, especially those of higher expected time cost, as one unit on a single processor means that the processor will run this task for longer time which has great influence to increase the probability of having a failure. i.e., reduces the application reliability. Parallelizing this task by executing its split tasks in more than one processor, each of them needs a shorter time and that will decrease its probability of failure. This gives a chance for the processors with lower processing power, which are capable to execute these split tasks, to enter the candidate processor set. Selecting the candidate processor(s) from a set of large number of processors can help to maximize the reliability of the application and to avoid (reduce) processing power fragmentation.

For the same previous experiments, simulation results showed that the (OHR) offered by the proposed algorithm (PRPS) relative to (Repar) or (RRTP) algorithm decreases as the performance parameters  $R$ ,  $m$ , max-split decreases. The proposed approach doesn't backtracks, like the other compared algorithms, when the deadline of the candidate task cannot be met. Instead, it uses parallelization as an effective tool to distribute its split tasks directly among different processor to help it in yielding the required probability to meet its deadline. The number of split tasks is chosen as small as possible, to avoid the idle times of the candidate

processors and decrease scheduling overheads. Evaluation of the mean and variance, when a task is parallelized, uses the sub-linear speedup assumptions [2, 7, 22], that take into account the overheads associated with communication and synchronization among the split tasks of a task. This emphasizes that the proposed algorithm has lower scheduling overhead than (Repar) and (RRTP), in this case. On the other hand, when one of the performance parameters increases, (PRPS) and (RRTP) incur almost the same amount of scheduling overhead, i.e. the (OHR) of (PRPS) relative to (Repar) tends to 1, while the (OHR) offered by each of these algorithms relative to (RRTP) algorithm continue to decrease substantially.



**Fig. 3: The effect of the laxity parameter (R) on the success ratio (SR).**

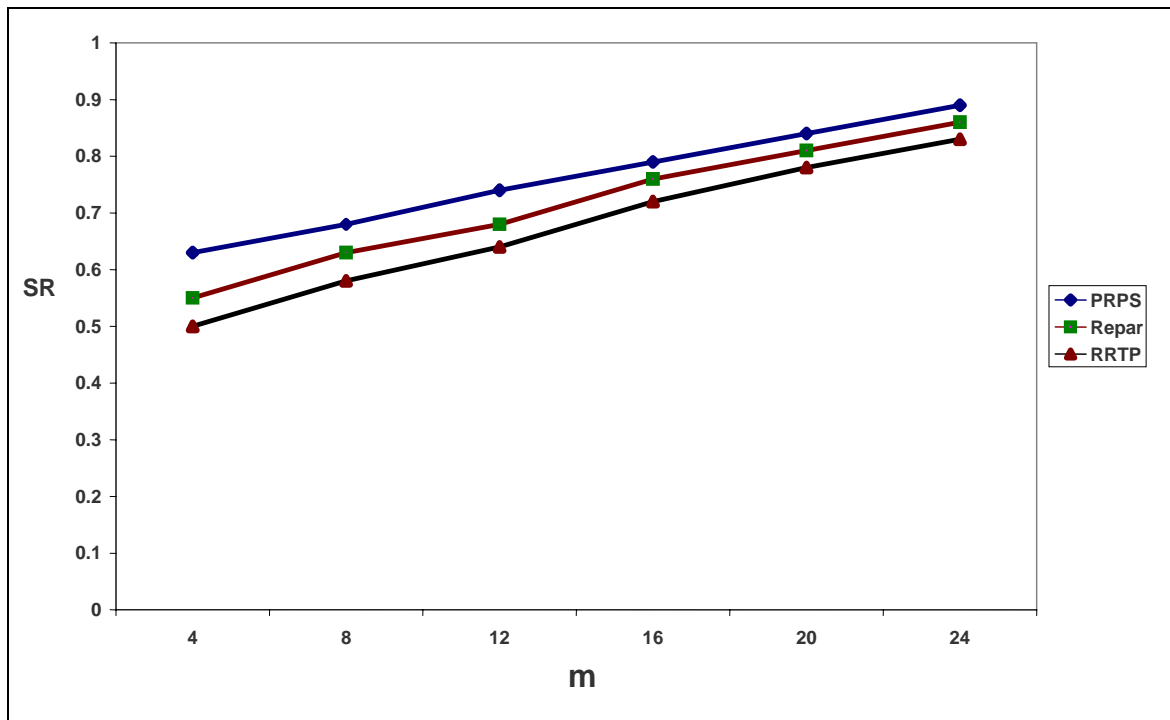


Fig. 4: The effect of the number of processors (m) on the success ratio (SR).

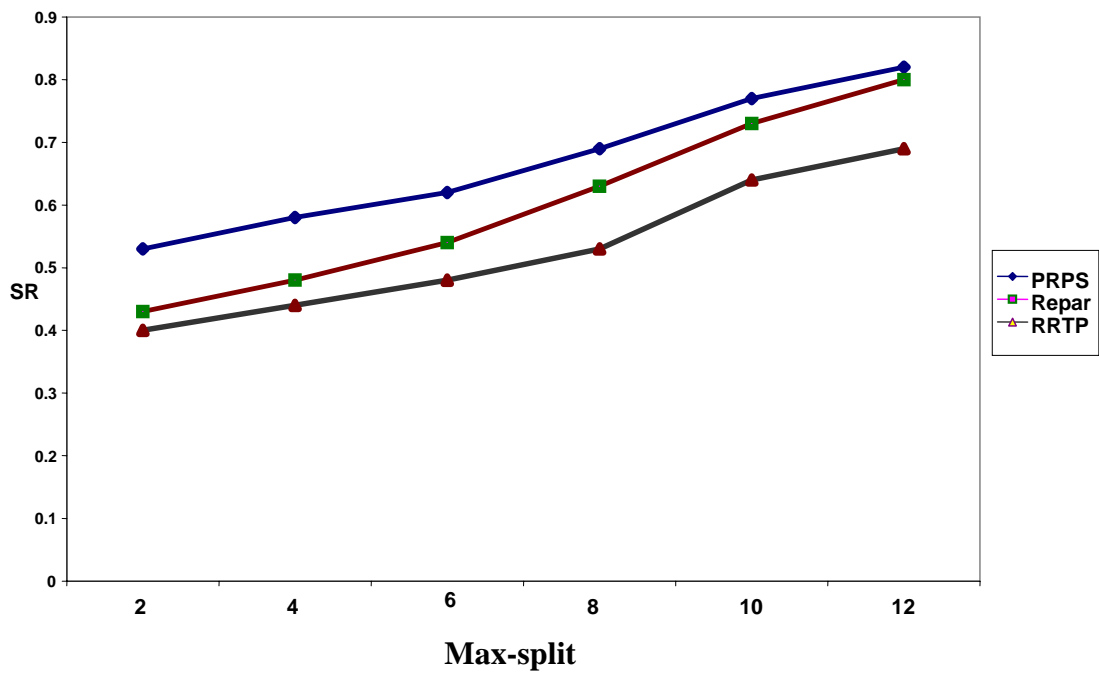


Fig. 5: The effect of the max-split number on the success ratio (SR).

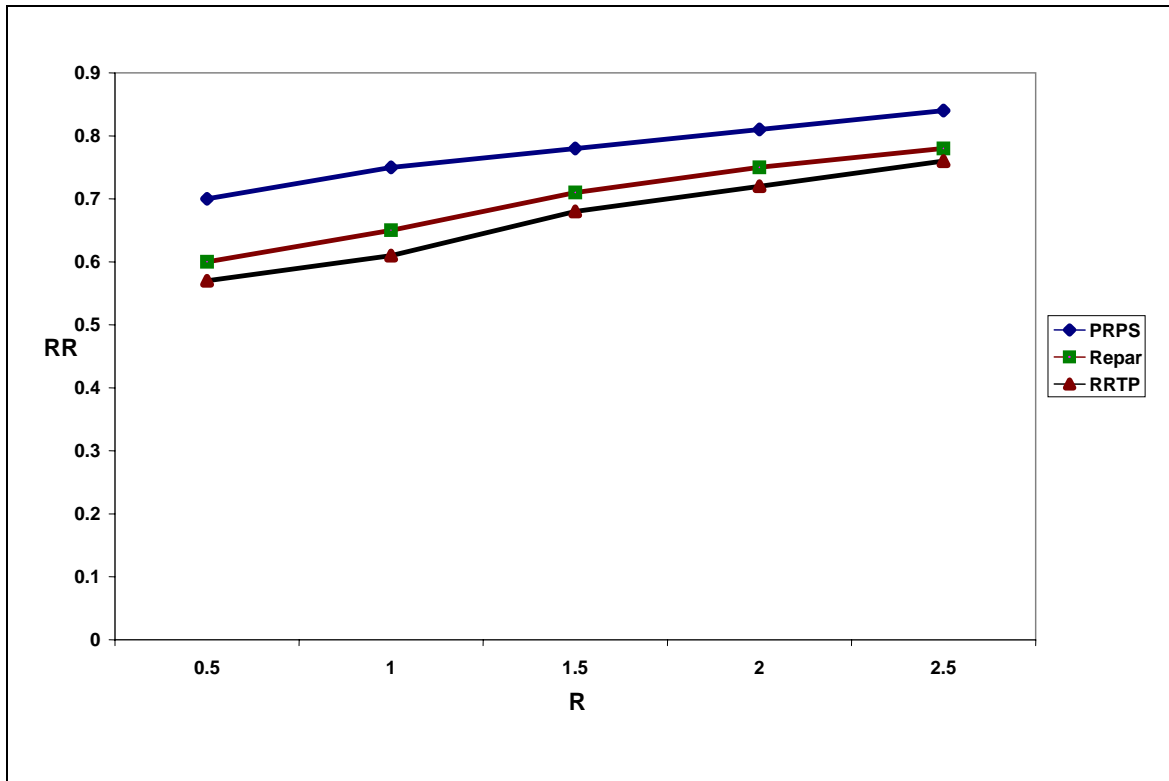


Fig. 6: The effect of the laxity parameter (R) on the reliability ratio (RR).

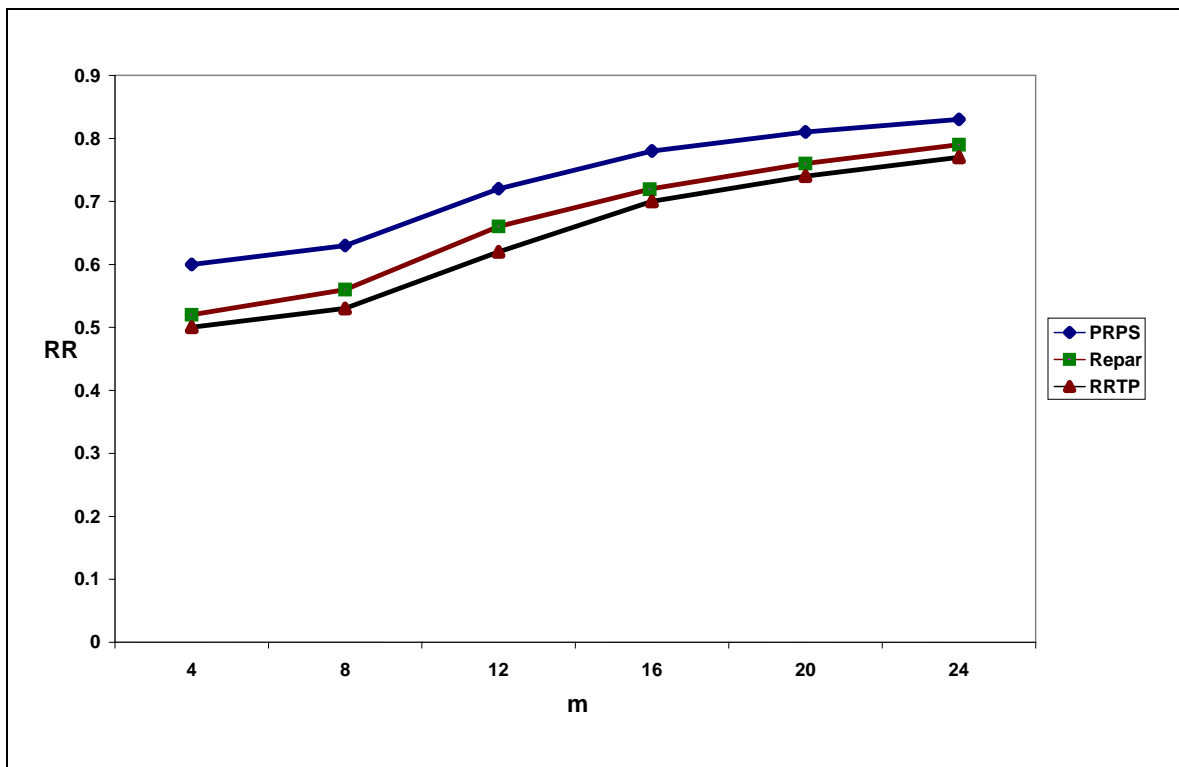


Fig.7: The effect of the number of processors (m) on the reliability ratio (RR).

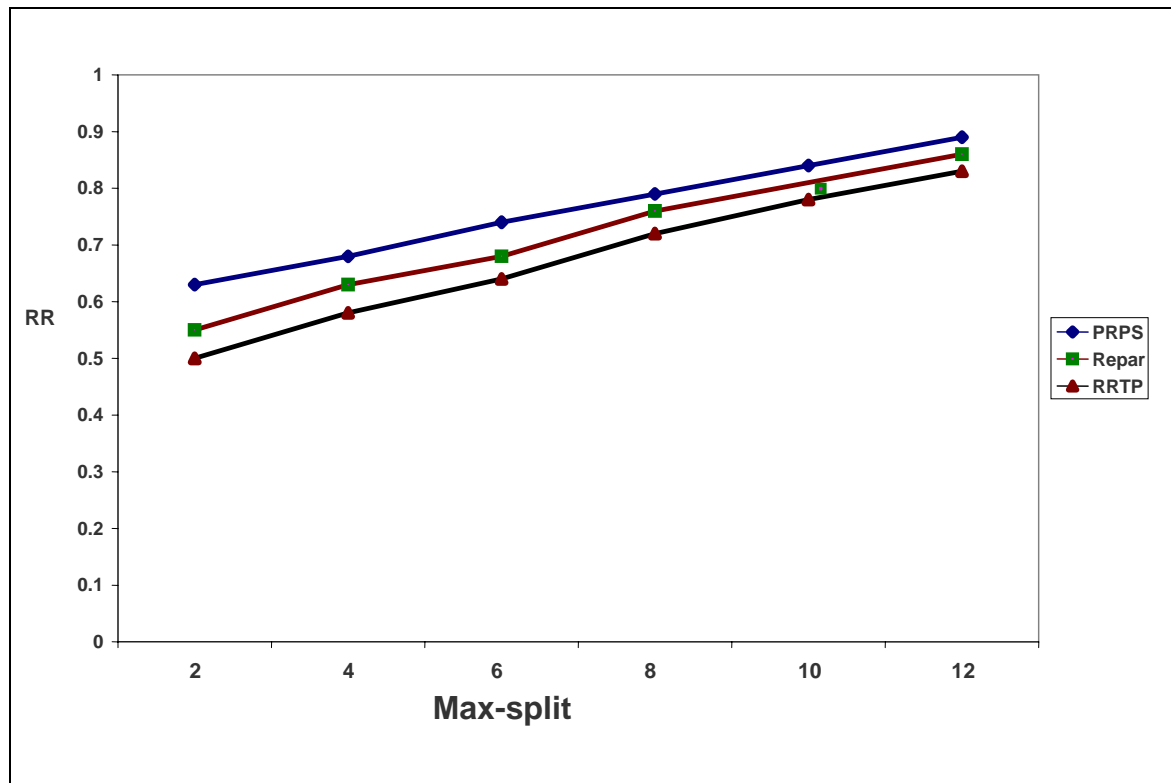


Fig. 8: The effect of the max-split number on the reliability ratio (RR).

## 5 Conclusion

High schedulability and reliability are the main keys requirements in the design and development of heterogeneous computing environments. In this paper, we address the stochastic scheduling problem for heterogeneous systems with reliability constraints. This paper introduces a simple compromise heuristic algorithm for scheduling a real-time application represented by a set of independent stochastic tasks with parallelizable contents in terms of maximizing the reliability of the application and the performance measures of the system, such as the success ratio and the processor utilization, simultaneously. The algorithm exploits parallelism in the scheduled tasks, if it is necessary, to improve the capabilities of the scheduling process. In this algorithm, all the scheduling decisions are based on the reliability measurements of the available processors in the system, in addition to the time cost distributions of different tasks and their deadlines. This can be used, with statistical properties and probability rules, as a good tool to guide the scheduling process efficiently to maintain the required objectives. Our algorithm, the Performance and Reliable-driven scheduling for Parallelizable Stochastic tasks (PRPS) algorithm was evaluated with the Reliability driven Real Time Periodic scheduling (RRTP) algorithm [28] and the (Repar) algorithm [31]. Results of simulations showed that the proposed

scheduling approach is beneficial in that it provides a better scheme with higher schedulability, in terms of the success ratio, and reliability while maintaining lower scheduling overhead compared to (RRTP) and (Repar) algorithms for a wide variety of task and system parameters.

## 6 Open Problem

Over the last decade, heterogeneous systems have been widely used for scientific and commercial applications. To improve the performance of applications running in heterogeneous systems, past research has developed a wide variety of scheduling algorithms for heterogeneous computing systems. However, there are many open related problems that remain to be solved before to maximize the performance and efficiency of these systems. Two of these problems that we plane to study in future are outlined below.

Although numerous approaches have been developed for real-time scheduling in heterogeneous systems, less attention has been devoted to reliability-driven scheduling for fault-tolerant systems. To bridge this gap in real-time scheduling technology, we need to develop a fault-tolerant scheduling algorithm consists of two consecutive phases. The first phase manages to assign tasks in a way to improve the reliability of the system while meeting real-time constraints of tasks. The second phase aims to make the system fault-tolerant by incorporating the primary-backup scheme in the process of scheduling.

High availability is a key requirement in the design and development of heterogeneous computing systems where processors operate at different speeds and are not continuously available for computation. Most existing scheduling algorithms designed for heterogeneous systems do not factor in availability requirements imposed by multi-class applications. To remedy this shortcoming, we need to investigate the stochastic scheduling problem for multi-class applications running in heterogeneous systems with availability constraints. Each node in a heterogeneous system is modeled using its speed and availability. Multiple classes of tasks submitted to the system are characterized by their time cost distributions and availability requirements. To incorporate availability and heterogeneity into scheduling, we must define new metrics to quantify system availability and heterogeneity for multi-class tasks. We then propose a scheduling algorithm to improve the availability of heterogeneous systems while maintaining good performance of the system in terms of its schedulability and the processor utilization.

### Acknowledgements

The author would like to thank the referees for their highly valuable comments and suggestions that greatly helped to improve the earlier version of this work. All of their suggestions were incorporated directly in the text.



## References

- [1] W. A. Halang, R. Gumzej, M. Colnarić, and M. Druzovec, "Measuring the performance of real-time systems", *Real-Time Systems*, Vol. 18, No.1, (2000), pp. 59-68.
- [2] G. Manimaran and C.R. Murthy, "An efficient dynamic scheduling algorithm for multiprocessor real-time systems", *IEEE Trans. Parallel and Distributed Systems*, Vol. 9, No. 3, (1998), pp. 312-319.
- [3] W. Li, K. Kavi and R. Akl, "A non-preemptive scheduling algorithm for soft real-time systems", *Computers and Electrical Engineering*, Vol. 33, No. 1, (2007), pp. 12-29.
- [4] E. Y. Abdel Maksoud, "Efficient combined scheduling of hard and soft real-time tasks in multiprocessor systems under a processing power-share strategy", *Parallel Processing Letters*, Vol. 19, No. 1, (2009), pp. 23-38.
- [5] E. Y. Abdel Maksoud and R. A. Ammar, "Effective Scheduling Algorithm for Stochastic Tasks in Distributed Multiprocessor Systems", *Proc. the 15<sup>th</sup> Int'l Conf. on Parallel and Distributed Computing Systems*, Louisville, Kentucky, (2002), pp. 277-282.
- [6] E. Y. Abdel Maksoud, R.A. Ammar and N. Debnath "An efficient scheduling algorithm for stochastic tasks with deadlines", *Proc. the 2<sup>nd</sup> IEEE Int'l Symp. on Signal Processing and Information Technology*, Marrakesh, Morocco, (2002), pp. 62-67.
- [7] E. Y. Abdel Maksoud and R. A. Ammar, "Heuristic scheduling algorithms for stochastic tasks in a distributed multiprocessor environment", *Lecture Series on Computer and Computational Science*, Vol. 2, (2005), pp. 1-5.
- [8] M.A. Iverson, F. Ozguner, and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment", *IEEE Trans. Computers*, Vol. 48, No. 12, (1999), pp. 1374-1379.
- [9] C. Rathbone, "Processing time analysis in a distributed/parallel environment with interrupts," MS thesis, University of Connecticut, 1989.
- [10] D. Liang and S.K. Tripathi, "On performance prediction of parallel computations with precedent constraints", *IEEE Trans. Parallel and Distributed Systems*, Vol. 11, No. 5, (2000), pp. 491-508.
- [11] J. M. Schopf and F. Berman, "Stochastic scheduling", *Proc. ACM/IEEE Conf. Supercomputing*, (1999), pp. 13-18.
- [12] X. Cai, X. Wu, and X. Zhou, "Dynamically optimal policies for stochastic scheduling subject to preemptive-repeat machine breakdowns", *IEEE Trans. on Automation Science and Engineering*, Vol. 2, No. 2, (2005), pp. 158-172.
- [13] T. Xie and X. Qin, "Stochastic scheduling for multiclass applications with availability requirements in heterogeneous clusters", *Cluster Computing*, Vol. 11, No. 1, (2008), pp.33-43.

- [14] K. Ramamritham and J.A. Stankovic "Scheduling algorithms and operating systems support for real – time systems," *IEEE*, Vol. 82, No. 1, (1994), pp. 55-67.
- [15] L. Tao, B. Narahari, and Y.C. Zhao, "Heuristics for mapping parallel computations to heterogeneous parallel architectures", *Proc. Heterogeneous Computing Workshop*, (1993), pp. 36-41.
- [16] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C.L. Wang, "Heterogeneous computing: challenges and opportunities", *Computer*, Vol. 26, No. 6, (1993), pp. 18-27.
- [17] M. Maheswaran and H.J. Siegel, "A Dynamic matching and scheduling algorithm for heterogeneous computing systems", *Proc. the 7<sup>th</sup> Heterogeneous Processing Workshop*, Orlando, Florida, USA, (1998), pp. 57-69.
- [18] A. Radulescu and A.J.C Van Gemund, "Fast and effective task scheduling in heterogeneous systems", *Proc. the 9<sup>th</sup> Heterogeneous Computing Workshop (HCW 2000)*, Cancun, Mexico, (2000), pp. 229-234.
- [19] H. Topcuoglu, S. Hariri, and M. Wu, "Performance – effective and low – complexity task scheduling for heterogeneous computing", *IEEE Trans. Parallel and Distributed Systems*, Vol. 13, No. 3, (2002), pp. 260-274.
- [20] R. Bajaj and D. P. Agrawal, "Improving of tasks in a heterogeneous environment", *IEEE Trans. Parallel and Distributed Systems*, Vol. 15, No. 2, (2004), pp.107-118.
- [21] B. Ucar, C. Aykanat, K. Kaya and M. Ikinici, "Task assignment in heterogeneous computing systems", *Parallel and Distributed Computing*, Vol. 66, No. 1, (2006), pp. 32-46.
- [22] G. Manimaran, C. S. Murthy, and K. Ramamritham, "A new approach for scheduling parallelizable tasks in real-time multiprocessor systems", *Real-Time Systems*, Vol. 15, No. 1, (1998), pp. 39-60.
- [23] E.E. Lewis, *Introduction to reliability engineering*, John Wiley & Sons, (1987).
- [24] S. Srinivasan and N. Jha, "Safety and reliability driven task allocation in distributed systems", *IEEE Trans. Parallel and Distributed Systems*, Vol. 10, No. 3, (1999), pp. 238-251.
- [25] A. Dogan and F. Ozguner "Reliable Matching and Scheduling of Precedence-Constrained Tasks in Heterogeneous Distributed Computing", *Proc. the 29<sup>th</sup> Int'l Conf. Parallel Processing (ICPP 2000)*, Toronto, Canda, (2000), pp. 307-314.
- [26] X. Qin, H. Jiang, C. Xie, and Z. Han, "Reliability – driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems," *Proc. the 12<sup>th</sup> Int'l Conf. Parallel and Distributed Computing and Systems (PDCS2000)*, Las Vegas, USA, (2000), pp. 617-623.
- [27] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing", *IEEE Trans. Parallel and Distributed Systems*, Vol. 13, No. 3, (2002), pp. 308-223.

- [28] N. Auluck and D. P. Agrawal, "Reliability driven, non-preemptive real-time scheduling of periodic tasks on heterogeneous systems", *Proc. the Int'l Conf. on Parallel and Distributed Computing and Systems (IASTED)*, Cambridge, USA, (2002), pp. 803-809.
- [29] A. Amin, R. Ammar and A. Eldouski, "Scheduling real-time parallel structure on cluster computing with possible processor failures", *Proc. the 9<sup>th</sup> IEEE Symposium on Computers and Communication*, Alexandria, Egypt, Vol. 1, (2004), pp. 62-67.
- [30] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters", *Parallel and Distributed Computing*, Vol. 65, No.8, (2005), pp.885-900.
- [31] W. Luo, X. Qin, and K. Bellam, "Reliability-Driven Scheduling of Periodic Tasks in Heterogeneous Real-Time Systems", *Proc. the 21<sup>st</sup> Int'l Conf. on Advanced Information Networking and Applications Workshops*, AINAW , Vol. 1, (2007), pp. 778-783.
- [32] M. Hamdaoui and P. Ramanathan, "Dynamic priority assignment technique for streams with (m,k) firm deadlines", *IEEE Trans. on Computer*, Vol. 44, (1995), pp. 1443-1451.
- [33] R. I. Davis, K.W. Tindell, and A. Burns, "Scheduling slack time in fixed priority pre-emptive systems", *Proc. the 14<sup>th</sup> IEEE Real-Time System Symposium*, Raleigh-Durham, North Carolina, USA, (1993), pp. 222-231.
- [34] K.S. Trivedi, *Probability & Statistics with Reliability, Queuing and Computer Science Applications*, John Wiley and Sons, New York, (2001).