JAIST Repository
https://dspace.jaist.ac.jp/

Title	Hybrid Overloading and Stochastic Analysis for Redundant Real-time Multiprocessor Systems.
Author(s)	Sun, Wei; Zhang, Yuanyuan; Yu, Chen; Defago, Xavier; Inoguchi, Yasushi
Citation	26th IEEE International Symposium on Reliable Distributed Systems, 2007. SRDS 2007.: 265-274
Issue Date	2007-10
Type	Conference Paper
Text version	publisher
URL	http://hdl.handle.net/10119/7801
Rights	Copyright (C) 2007 IEEE. Reprinted from 26th IEEE International Symposium on Reliable Distributed Systems, 2007. SRDS 2007. This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of JAIST's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.
Description	

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Hybrid Overloading and Stochastic Analysis for Redundant Scheduling in Real-time Multiprocessor Systems*

Wei Sun[1], Yuanyuan Zhang[2], Chen Yu[1], Xavier Defago[1] and Yasushi Inoguchi[1, 3]

[1]Graduate School of Information Science
[3]Center for Information Science
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Nomi, Ishikawa, 923-1292 Japan
{sun-wei, yuchen, defago, inoguchi}@jaist.ac.jp

[2]Fujitsu Laboratories Ltd.
Kawasaki, Kanagawa, 211-8588, Japan
zhang.yuanyuan@jp.fujitsu.com

Abstract

In multiprocessor systems, redundant scheduling is a technique that trades processing power for increased reliability. One approach, called primary-backup task scheduling, is often used in real-time multiprocessor systems to ensure that deadlines are met in spite of faults. Briefly, it consists in scheduling a secondary task conditionally, in such a way that the secondary task actually gets executed only if the primary task (or the processor executing it) fails to terminate properly. Doing so avoids wasting CPU resources in the failure-free case, but primary and secondary tasks must then compete for resources in case of failure. To overcome this, overloading strategies, such as primary and backup overloading (PB) and backup-backup overloading (BB), aim at improving schedulability while retaining a certain level of reliability.

In this paper, we propose a hybrid overloading technique based on extended PB overloading, which combines advantages of both PB and BB overloading. The three overloading strategies are then compared through a stochastic analysis, and by simulating them under diverse system conditions. The analysis shows that hybrid overloading provides an excellent tradeoff between schedulability and reliability.

1. Introduction

Real-time multiprocessor systems are defined as systems in which the correctness depends on not only the logical results of computations, but also the time at which the results are produced [1]. Thus, it is essential that tasks are completed before their deadlines even in the presence of processor or task failures. Fault-tolerance is thus an inherent requirement of real-time systems.

In a multiprocessor system, fault-tolerance can be provided by redundantly scheduling copies of tasks on different processors [1-9]. Primary-backup scheduling is one instance of a fault-tolerant scheduling technique. In primary backup task scheduling, two instances of a task, namely, primary and backup tasks, are scheduled on two different processors. The backup task gets executed if and only if the primary task fails some acceptance test [4-11].

To improve the schedulability (i.e., the total number of tasks that can be scheduled), overloading is often used. Primary backup overloading (called PB overloading for short) allows a primary task to be scheduled onto a processor and time slot overlapping with that of the backup instance of another task [11](Fig.1a). Backup-backup overloading (BB overloading) allows only backup tasks to overlap with each other, but not with primary tasks [4,7,11](Fig.1b). Al-Omari *et al.* [11] find that PB overloading is able to achieve better performance than BB overloading, which is in turn better than any non-overloading strategy.

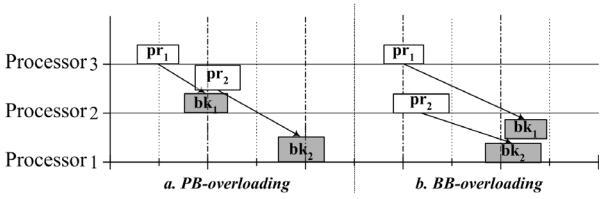

Figure 1. Examples of overloading.

In this paper, we improve existing PB overloading, and then propose a hybrid overloading strategy (not meant as a simple combination of PB and BB overloading). We evaluate the approach by comparing it with PB and BB overloading, using both a stochastic analysis and simulations.

*This research is conducted as a program for the "21st Century COE Program" by Ministry of Education, Culture, Sports, Science and Technology, Japan

1060-9857/07 $25.00 © 2007 IEEE
DOI 10.1109/SRDS.2007.11

IEEE computer society

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

Related work. In primary-backup task scheduling, the backup version is often referred to as a "ghost" task [16]; the backup is scheduled only conditionally and is removed upon the successful completion of the primary [4,6,11].

The PB overloading of Al-Omari *et al*. [11] limits the number of primaries to only two. With this limitation and on three processors, PB overloading offers better schedulability than BB overloading. However, on more than three processors, PB overloading with only two tasks can no longer offer better schedulability than BB overloading. Although we could not find any scheduling strategy combining PB and BB overloading, nothing prevents such a combination.

Schedulability and reliability are often used as the main metrics to evaluate fault-tolerant real-time task scheduling. PB overloading was shown to offer better schedulability but lower reliability than BB overloading [11]. Considering the tradeoff between schedulability and reliability in practical situations and the fact that failures are usually rare events, PB overloading is overall more effective than BB overloading.

With the approach of Ghosh *et al.* [4], backups are scheduled as late as possible, are overloaded on other backups whenever possible, and a function is used to control the overlapping depth between overloaded backups. To tolerate more faults, BB overloading can be made to take place only on a static subset of the processors [7] (static grouping). Al-Omari *et al.* [11] extended it to dynamic grouping and presented a PB overloading strategy with improved schedulability.

It is known that no optimal algorithm exists for dynamic scheduling on a multiprocessor system [15]. Aperiodic tasks (i.e., the arrivals and deadlines of which are not known in advance) require dynamic scheduling algorithm. When the scheduler of Ghosh *et al*. [4] cannot find a proper time slot for a new task, some primary is re-scheduled by moving it forward, while no backups can be re-scheduled. In [6,11], scheduling is based on the Spring approach [12], which is a heuristic that dynamically schedules tasks according to resource requirements.

Roadmap. The rest of the paper is organized as follows. Section 2 introduces the system model and basic definitions, including the tasks and fault models used in our performance analysis. In Section 3, we present an improved PB overloading technique, as well as our hybrid overloading. Section 4 presents the stochastic analysis, and the simulation results are discussed in Section 5. Section 6 concludes the paper.

2. Models and Definitions

2.1. System model

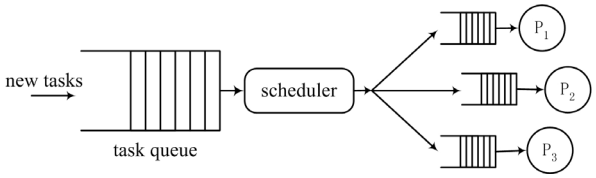

Figure 2. System structure.

The system model considered in this paper is similar to those found in the literature [2,6,7,11], and is illustrated on Fig.2. All tasks arrive at the scheduler, a centralized server, which process them on a first-come first-served basis. The tasks are then dispatched to other processors, and executed according to their schedule. All processors have identical computing capability and are connected through a network. The scheduler is running in parallel with the processors. Each processor has its own task queue. We assume that the scheduler never fails, either because it runs on a fault-proof processor, or because it has been replicated. The total number of processors is m.

2.2. Task model

Tasks have the following characteristics:

1. Tasks are asynchronous and aperiodic, i.e., task arrivals are not known in advance. Each task T_i has the numeric characteristics: *arrival time* (a_i), *worst-case computation time* (c_i), deadline (d_i) and task laxity l_i. The mean of all c_i is C.
2. Each task has two identical versions. The version to be scheduled earlier in a schedule is marked as primary (pr_i) and the other one is marked as backup (bk_i). After a primary has finished successfully, its backup is immediately cancelled. The outputs of the primary and its backup are always identical. In other words, the tasks are idempotent.
3. Tasks are not parallelizable. A task can be executed only on one single processor.
4. Tasks are independent. For tasks with precedence constraints, the deadlines of tasks can be modified to make tasks comply with their precedence constraints [7].
5. Tasks are non-preemptable.

2.3. Fault model

Each processor, except the scheduler, may fail due to hardware or software faults. The failure of a processor results in the failure of all tasks it executes. Faults can be either transient or permanent. Faults are assumed to be independent and only affect a single proc-

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

essor. For simplicity, we assume that, at any one time, at most one single processor can be crashed. In other words, we consider 1-timely-fault tolerant schedules, where a *k*-timely-fault-tolerant (*k-TFT*) schedule is defined as the schedule in which no task deadlines are missed, despite *k* arbitrary processor failures [13].

MTBF (mean time between failures) is the expected time between two consecutive failures. *TTSF* (time to second failure) is the time to the second *simultaneous* failure, i.e., the critical time between two consecutive failures where the second failure occurs during the vulnerability window opened by the first one. Hence, a longer *TTSF* means a lower reliability. In order to guarantee the successful execution of tasks, we assume that $\forall T_i(d_i - a_i)$ is much less than *MTBF*. If no overloading exists, in the worst case, *TTSF* will be equal to $max(d_i - a_i)$. Because of overloading, *TTSF* will be discussed again in the sequel.

A failure detection mechanism is assumed to exist and provide information on processor/task failures. It is assumed to be perfect (i.e., no false suspicions) and timely (i.e., failures announced before the starting time of backups). The scheduler will not schedule tasks to a known failed processor.

2.4. Definitions

An *overloading chain* refers to a sequence of tasks that are related due to overloading. In PB overloading, a chain is formed between two tasks T_1 and T_2, when the backup of T_1 overlaps with the primary of T_2. In BB overloading, a chain is formed between two tasks when their backups overlap each other.

The *time length* l_t of an overloading chain is the length of time interval between the start time of the first task and the end time of the last task in this chain.

The *overloading space s* of an overloading chain is the set of processors involved in this chain. The *space length* l_s of an overloading chain is the size of *s* minus 1, viz., $l_s = |s| - 1$. l_s is also equal to the number of primaries in the chain. An overloading chain is said to be *looped* when it involves the same processor repeatedly. For instance, the chain depicted in Fig.3 is a looped chain because processor 4 is involved both for pr_1 and later for bk_4. An overloading chain is *full* when either l_t or l_s reaches its maximum.

3. Overloading techniques

In this section, we first improve the PB overloading of Al-Omari *et al.* [11], and then introduce our hybrid overloading. In this paper, we only focus on the overloading techniques that are usually affected by practical scheduling algorithms.

3.1. Extended PB overloading

Although Al-Omari *et al.* [11] considered primary backup overloading that can involve only two primaries, it is easy for a scheduler to produce longer overloading chains. However, the problem is how to guarantee the validity and the reliability of the chain.

For PB overloading, looped chains are invalid. This is because a single processor failure may actually prevent some tasks from being executed at all. For instance, consider the looped chain of Fig.3. If processor 4 fails while executing pr_1, then bk_1 will have to be scheduled, triggering a domino effect by which each of the backup tasks of the chain must be executed instead of their primary. The problem is that bk_4 must be scheduled, but it cannot be executed since processor 4 has failed and may not have recovered. Note that, if $l_s > m$ for an extended PB overloading chain, then the chain must be looped.

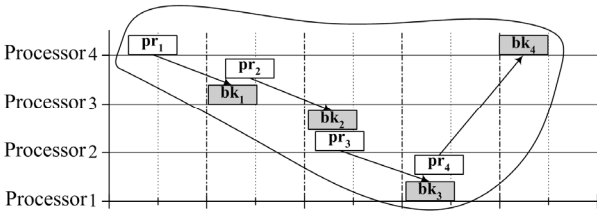

Figure 3. Looped chain. The chain across different processors joins itself on Processor 4.

TTSF is equal to l_t for an overloading chain. For BB overloading, in the worst case we have that

$$TTSF = max(l_t) = max(d_i - a_i), \forall T_i , \quad (1)$$

For PB overloading, in the worst case we have that

$$TTSF = max(l_t) = l_s \cdot max(d_i - a_i), \forall T_i . \quad (2)$$

If we do not properly control the length of PB overloading chains, then extended PB overloading may possibly be very unreliable.

Hence, we have the following rules for extended PB overloading:

1. The primary and backup of a task cannot coexist on the same processor.
2. If the number of processors is *m*, the maximum space length l_s of any chain is *m*-1.
3. The time length l_t of a chain must be less than the minimum time interval between failures.
4. A full overloading chain (i.e., when either space or time length is maximal) cannot be extended.
5. If a full overloading chain is shortened after the successful completion of a task, this chain is no longer full and can be extended again.

For a real-time system, it is usually difficult to predict the minimum time interval between failures. Thus, l_t can be set as an empirical parameter. However, for a

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

multiprocessor system with only a few processors, the limiting factor for the length of overloading chains will likely be the number of processors rather than the interval between failures. Similarly, for a system with many processors, overloading will usually take place within subgroups of processors [7,11]. As a result, overloading chains, limited by the number of processors in each group, will remain relatively short, in particular, shorter than the *MTBF*. In fact, even in a system with a large number of processors, the probability of forming the maximum l_t remains small. A stochastic analysis about the reliability is presented in Section 4.

Extended PB overloading is a natural extension of PB overloading. Hence, in the sequel, PB overloading will refer to its extended version.

3.2. Hybrid overloading

All tasks in previous schedules have four forms: single primary (*pr*), single backup (*bk*), overloaded primary and backup (*pr*, *bk*) and overloaded backups (*bk*, *bk*). A BB overloading chain can contain (*pr*) and (*bk*, *bk*). A PB overloading chain can contain (*pr*), (*pr*, *bk*) and (*bk*). With a simple combination of PB and BB overloading, an overloading chain cannot contain all four forms. But, in our hybrid overloading, a chain can contain all four forms, plus a new form, overloaded primary and backups (*pr*, *bk*, *bk*). Two possible hybrid overloading chains are shown in Fig.4.

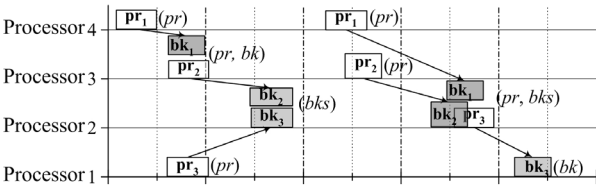

Figure 4. Examples of hybrid overloading.

Hybrid overloading inherits all rules and advantages from BB and PB overloading. It has better schedulability than BB overloading and yet better reliability than PB overloading.

Under some conditions, a hybrid overloading chain can be transformed into different shapes, including into a purely BB or PB overloading chain. This means that hybrid overloading can be made to satisfy different scheduling requirements, such as, high schedulability first, resource utilization first, or reliability first. Due to the flexibility of hybrid overloading, many different algorithms can be designed to schedule tasks into hybrid overloading chains. In this paper, we however focus on the introduction and analysis of hybrid overloading itself, rather than scheduling algorithms.

4. Stochastic analysis

In this section, we compare overloading techniques according to their respective schedulability and reliability. We use stochastic analysis considering task arrival, task laxity, and different worst case execution time. Tasks are scheduled on a first-come first-served basis.

The number of primaries in an overloading chain is denoted as n, and the number of primaries in a schedule is denoted as N. The time interval of task arrivals follows an exponential distribution with mean $1/\lambda$, where λ is the arrival rate of tasks.

The worst-case execution time is distributed uniformly over the interval $[C_{min}, C_{max}]$, with $C = \frac{C_{max} + C_{min}}{2}$.

The task laxity l is distributed uniformly over the interval $[3, L]$.

The time length and space length of overloading chains follow uniform distribution within their maximum values and minimum values respectively.

4.1. Resource utilization analysis

Since primary-backup fault-tolerant scheduling relies on redundancy, the improved reliability of the system comes at the expense of resource utilization, in particular, schedulability. For a full overloading chain, PB overloading, BB overloading, and hybrid overloading have the same maximum resource utilization, as long as tasks can be tightly overloaded. This resource utilization is $(m-1)/m$, which means that only one time slot is used for backups. However, maximal resource utilization cannot always be reached, depending on task arrival, task computation time, and task laxity.

A full overloading chain cannot always be scheduled successfully. For the model presented in Section 2, we have the following lemmas.

Lemma 1. The probability of emerging a BB overloading chain with n tasks, $Pr_{BB}(n)$, is

$$Pr_{BB}(n) = \int_{l=3}^{L} \frac{1}{L-3} \int_{c=C_{min}}^{C_{max}} \frac{1}{C_{max} - C_{min}} \times \left\{ 1 - \sum_{z=0}^{n-2} \left[\frac{(l \cdot c - 2c)^z}{z!} \cdot \lambda^z \cdot e^{-\lambda \cdot (l \cdot c - 2c)} \right] \right\} \cdot dc\,dl, \quad 2 \le n \le m-1. \quad (3)$$

Proof Sketch. An example of BB overloading is shown in Fig.5, where x_i denotes the time intervals between task arrivals. All tasks must be allocated within $l \cdot c$. In this paper, a primary and its backup cannot be executed in parallel, that is, the execution of a primary cannot be overlapped with the execution of its

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

backup. Thus, if bk_2 is scheduled to overload on bk_1, the second task T_2, must satisfy $l \cdot c - x_1 \geq 2c$ .Similarly, task T_3 must satisfy $l \cdot c - x_1 - x_2 \geq 2c$. If n tasks are overloaded together, the probability of forming a BB overloading chain is

$$Pr_{BB}(n) = Pr(x_1 + x_2 + \cdots + x_{n-1} \leq l \cdot c - 2c) .$$

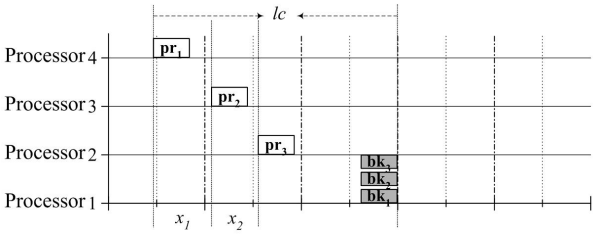

Figure 5. Variables in BB overloading.

x_i follows an exponential distribution. We give the following formula for the sum of n exponentially distributed random variables, of which we omit the proof.

$$Pr(x_1 + x_2 + \cdots + x_n \leq k) = 1 - \sum_{z=0}^{n-1} \left(\frac{k^z}{z!} \cdot \lambda^z \cdot e^{-\lambda \cdot k} \right), k \geq 0 .$$

With this formula, the probability is

$$Pr_{BB}(n) = Pr(x_1 + x_2 + \cdots + x_{n-1} \leq l \cdot c - 2c) = \int_{l=3}^{L} \frac{1}{L-3} \int_{c=C_{min}}^{C_{max}} \frac{1}{C_{max} - C_{min}} \times Pr(x_1 + x_2 + \cdots + x_{n-1} \leq l \cdot c - 2c) dcdl .$$

Hence, we have Eq.3. □

Lemma 2. The probability of emerging a PB overloading chain with n tasks, $Pr_{PB}(n)$, is

$$Pr_{PB}(n) = \left(\int_{l=3}^{L} \frac{1}{L-3} \int_{c=C_{min}}^{C_{max}} \frac{1}{C_{max} - C_{min}} \times \int_{0}^{l \cdot c - c} \lambda \cdot e^{-\lambda \cdot x} dxdcdl \right)^{n-1}, 2 \leq n \leq m-1 . \quad (4)$$

Proof Sketch. An example of PB overloading is shown in Fig.6. To schedule a PB overloading chain with n tasks, the chain must satisfy

$$l_2 \cdot c_2 - x_2 \geq c_1, l_3 \cdot c_3 - x_3 \geq c_2, \cdots, l_n \cdot c_n - x_n \geq c_{n-1} .$$

Since l_i, c_i, and x_i are independent. The probability of emerging a PB overloading chain with n tasks is

$$Pr_{PB}(n) = [Pr(x \leq l \cdot c - c)]^{n-1} = \left(\int_{l=3}^{L} \frac{1}{L-3} \int_{c=C_{min}}^{C_{max}} \frac{1}{C_{max} - C_{min}} \int_{0}^{l \cdot c - c} \lambda \cdot e^{-\lambda \cdot x} dxdcdl \right)^{n-1} .□$$

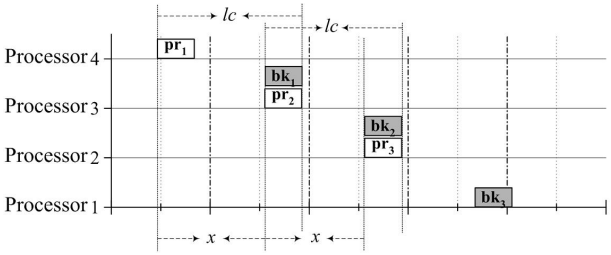

Figure 6. Variables in PB overloading.

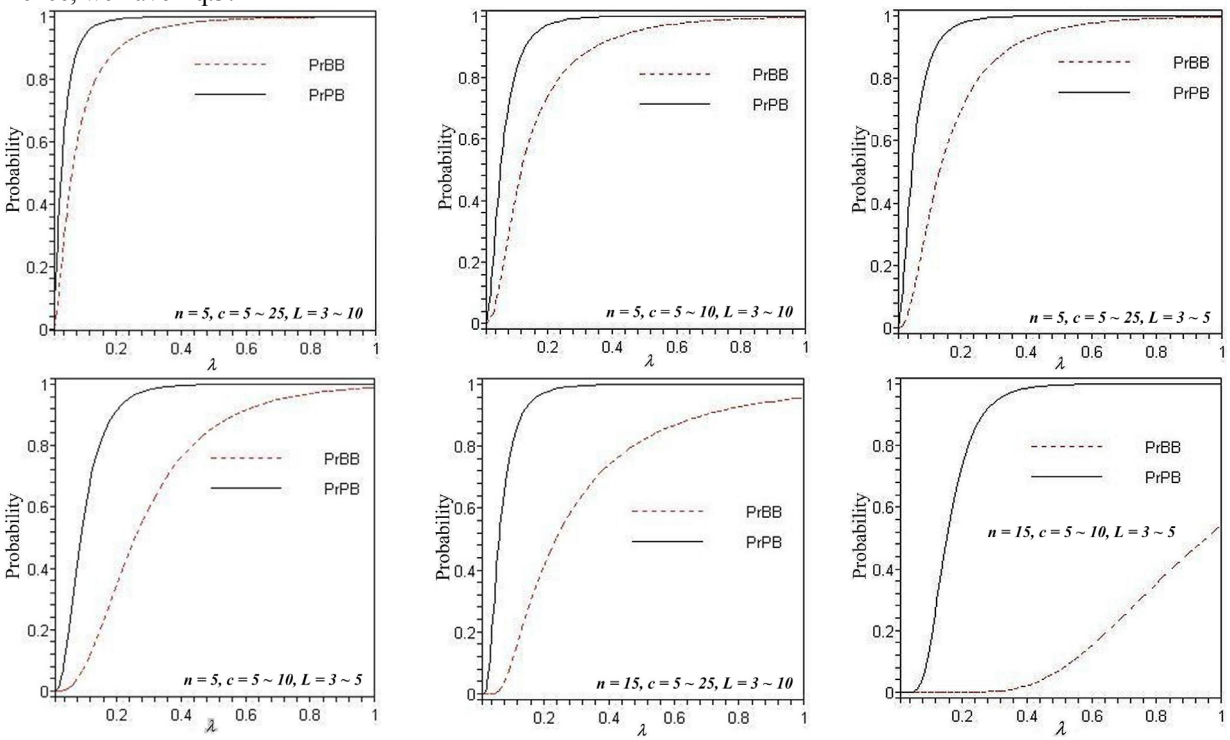

Figure 7. The probabilities of emerging an overloading chain in different scenarios.

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

The probabilities of emerging an overloading chain are affected by the task arrival rate, the number of tasks in the chain, and the task laxity. Given specific parameters, the probability is represented in Fig.7. $l \cdot c$ is defined to be the task window [4]. As the task window decreases, the probabilities decrease. As the number of tasks (in a chain) increases, the probabilities decrease. Regardless of the parameters, Pr_{PB} is always larger than Pr_{BB}. This is summarized in the following proposition.

Proposition 1. The probability of emerging a PB overloading chain is larger than that of emerging a BB overloading chain, if the number of tasks in the PB overloading chain is equal to that in the BB overloading chain.

The probability of emerging an overloading chain with n tasks, $Pr_*(n)$, can be used to calculate the number of time slots occupied by all primaries and backups in a previous schedule with N tasks. ($*$=*PB* or *BB*).

The scheduling algorithms with overloading techniques always try to improve resource utilization and increase the number of tasks overloaded together. If tasks cannot form a long overloading chain, a relatively short chain will be formed instead. For a prior schedule with N primaries, the time slots occupied by all primaries and backups, N_a, can be calculated from the following theorem.

Theorem 1. Given N previously scheduled tasks, the number of time slots occupied by all primaries and backups, N_a, is

$$N_a = \sum_{i=m-1}^{b}\left[N \cdot R(i) \cdot \frac{i}{i+1}\right] + 2N \cdot \left[1 - \sum_{i=m-1}^{b} R(i)\right], b \geq 2, \quad (5)$$

$$R(y) = \begin{cases} \left[1 - \sum_{j=m-1}^{y+1} R(j)\right] \cdot Pr_*(y), & 2 \leq y < m-1 \\ Pr_*(m-1), & y = m \end{cases}. \quad (6)$$

where b is the minimum bound for which $\sum_{x=m-1}^{b} R(x) \leq 1$. and $R(i)$ is the ratio of the number of tasks in the chains (with $l_s = i$) to N tasks.

Proof Sketch. Since a short overloading chain will be formed only if a long one cannot be made, a longer chain emerges always prior to a shorter one. The following equations can represent this process.

$$R(m-1) = Pr_*(m-1),$$
$$R(m-2) = [1 - Pr_*(m-1)] \cdot Pr_*(m-2),$$
$$R(m-3) = \{1 - Pr_*(m-1) - [1 - Pr_*(m-1)] \cdot Pr_*(m-2)\} \times Pr_*(m-3),$$
$$R(m-4) = \{1 - R(m-1) - R(m-2) - R(m-3)\} \cdot Pr_*(m-4),$$
$$\vdots$$

Here, $R(i)$ is defined to be the ratio of the number of tasks in chains with $l_s = i$ to all previously scheduled tasks. This process will go on until all tasks are scheduled. Thus, we can get Eq.6. When this process ends, the shortest space length of an overloading chain can be any value between 2 and m-1, and there might be some tasks not being overloaded on any previous task. For a system with m processors, the maximum number of tasks in a full overloading chain is m-1, and these tasks occupy m time slots. Generally an overloading chain with i tasks will occupy i+1 time slots. A task that is not overloaded will occupy two time slots. The iterative calculation is given in Eq.5. □

Finally the resource utilization is given as

$$Utilization = \frac{N}{N_a}. \quad (7)$$

According to Proposition 1, the resource utilization of PB overloading is always larger than BB overloading.

4.2. Schedulability and Reliability Analysis

For a system with finite resources, the tradeoff between schedulability and reliability always comes as a result of resource competition between primaries and backups. In this section, we first consider a failure-free system, and then discuss fault-tolerant systems. For a failure-free real-time system, when a new task arrives and finds N previous tasks in the current schedule, this task can be accepted only if its deadline can be guaranteed. Thus, the probability of accepting $(N+1)_{th}$ task is

$Pr_a(N+1) = Pr\left(l \cdot c - \frac{N \cdot C}{m} \geq c\right)$. $N \cdot C/m$ is the average finish time of the last task on each processor.

$$Pr\left(l \cdot c - \frac{N \cdot C}{m} \geq c\right) = 1 - Pr\left(l \cdot c - \frac{N \cdot C}{m} < c\right)$$
$$= 1 - Pr\left[(l-1) \cdot c < \frac{N \cdot C}{m}\right]$$
$$= 1 - \int_3^L \frac{1}{L-3} \int_{C_{min}}^{\frac{N \cdot C}{m \cdot (l-1)}} \frac{1}{C_{max} - C_{min}} dc dl$$
$$= 1 - \frac{N \cdot C \cdot ln\left(\frac{L-1}{2}\right)}{(L-3) \cdot (C_{max} - C_{min}) \cdot m} + \frac{C_{min}}{C_{max} - C_{min}}. \quad (8)$$

In Eq.8, all parameters are constant for a specific system, except for N. Hence the probability of accepting a task is a function of the number of tasks in a previous schedule.

$$Pr_a(N+1) = f(N) = \begin{cases} 1, & N \leq m \\ 1 - \frac{N \cdot C \cdot ln\left(\frac{L-1}{2}\right)}{(L-3) \cdot (C_{max} - C_{min}) \cdot m} + \frac{C_{min}}{C_{max} - C_{min}}, & m < N \end{cases} \quad (9)$$

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

For PB overloading, a new task can be overloaded on previous backups that are not in a full overloading chain. The function should be

$$Pr_a^{PB}(N+1)= f\left\{N\cdot[1-Pr_{PB}(m-1)]+N\cdot Pr_{PB}(m-1)\cdot\frac{m}{m-1}\right\} \quad (10)$$

For BB overloading, all time slots occupied by all primaries and backups cannot be used for the new task. Therefore the function is

$$Pr_a^{BB}(N+1)= f(N_a). \quad (11)$$

The acceptance probabilities are shown in Fig.8. The acceptance probability of PB overloading is always larger than that of BB overloading. As the number of previous tasks increases, the acceptance probabilities decrease. The number of previous tasks will reach a maximum value, which is the equilibrium status of arrival, departure, and rejection of tasks. This maximum number for PB overloading is larger than BB overloading, that is, the system with PB overloading can accept more tasks than the system with BB overloading. When the task arrival rate is much larger for a system, the probability for PB overloading is close to that of BB overloading because almost all chains will reach the maximum space length, which is the same for both PB and BB overloading. Note that we do not consider the effect of scheduling algorithm in this analysis.

In the literature, real-time systems are usually characterized by Markov model [4, 7, 11] and queuing model [18, 19, 20]. In particular, a real-time system can be characterized by a queuing model such as *M/M/m*. In [19, 20], it was shown that *M/M/1* is very accurate to model a real-time system with one single processor. The fault-tolerant real-time systems with primary-backup techniques can be characterized by *M/M/m* with customer leaving. The rejection of tasks can be considered as customers leaving the system. The acceptance probability can be used to build an *M/M/m* model with customer leaving. The average acceptance ratio of the system derives from the probability of *N* tasks waiting in the system and the acceptance probability of the $(N+1)_{th}$ tasks. In this paper, we will not give further analytical results of queuing model due to space constraints.

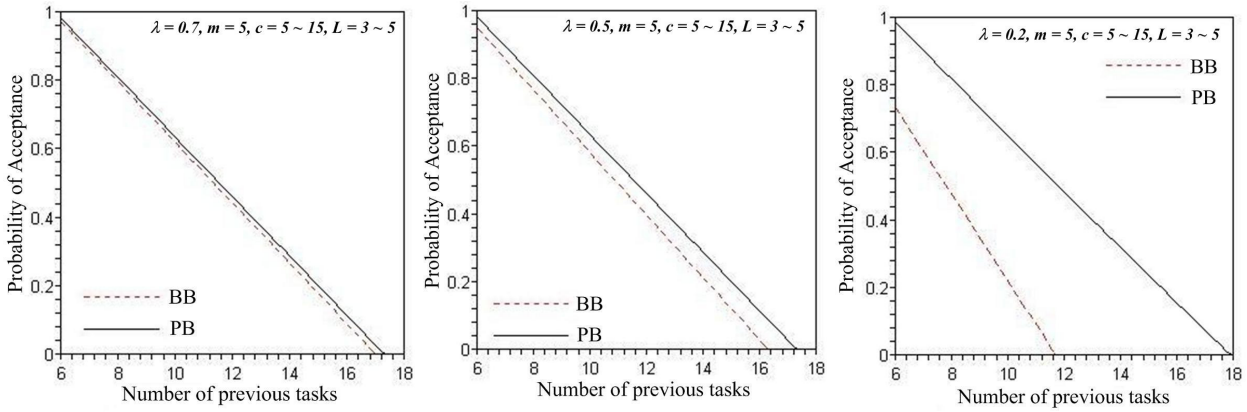

Figure 8. The Probabilities of acceptance in different scenarios.

The reliability is usually measured by the time to second failure (*TTSF*) [4,11]. As shown in Eq.1 and Eq.2, *TTSF* is affected by the time length of an overloading chain. It may seem that the time length of a PB overloading chain is much longer than that of a BB overloading chain, and that the former might be so long as to be even longer than that *MTBF*. In fact, it turns out that the time length of a PB overloading chain is limited by the maximum number of tasks that can be accepted by the system.

For BB overloading chains the time length l_t^{BB} is between 2*c* and *l*·*c*. Hence the average time length is

$$l_{t\_a}^{BB}=\frac{L\cdot C+2C}{2}=\frac{(L+2)\cdot(C_{max}+C_{min})}{4}.$$

For PB overloading, the maximum number of accepted tasks in a schedule can be worked out from the calculation of Eq.8, where the upper bound of integration should be less than C_{max}. Thus we have

$$\frac{N\cdot C}{m\cdot(l-1)}\le C_{max}\Rightarrow\frac{N\cdot C}{m}\le(l-1)\cdot C_{max}$$

$$\Rightarrow\frac{N\cdot C}{m}\le(L-1)\cdot C_{max}.$$

$(L-1)\cdot C_{max}$ is the real maximum time length of PB overloading chains. The minimum time length of a full PB overloading chain is given by $m\cdot C_{min}$. Thus we have the average time length for PB overloading chains

$$l_{t\_a}^{PB}=\frac{(L-1)\cdot C_{max}+m\cdot C_{min}}{2}.$$

The average time length of full overloading chains is listed in Table 1.

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

Table 1. The Average time length of overloading chains in different scenarios.

C_{min} (s)	C_{max} (s)	L	m	BB (s)	PB (s)	PB/BB
5	10	5	4	26.25	30	1.143
5	15			35	40	1.143
5	25			52.5	60	1.143
5	10	10		45	55	1.222
5	15			60	77.5	1.291
5	25			90	122.5	1.361
5	10	10	8	45	65	1.444
5	15			60	87.5	1.458
5	25			90	132.5	1.472
5	10	15		63.75	90	1.411
5	15			85	125	1.470
5	25			127.5	195	1.529

As shown in Table 1, the time length of PB overloading chains is longer than that of BB overloading chains, but the ratio of PB to BB is not very large. Note that the calculation of time length assumes that the time length follows a uniform distribution between the maximum and the minimum. In a practical scheduling process, the time length of PB overloading chains is usually close to its minimum due to the heavy traffic of tasks.

Finally, as shown in Section 3.2, in hybrid overloading a new task can be scheduled as BB overloading when there are lots of free time slots, and scheduled as PB overloading to use the time slots occupied by the previous backups when few free time slots are available. The behaviour of hybrid overloading is totally decided by a practical scheduling algorithm. Regardless of the scheduling algorithm, because of the flexibility of hybrid overloading, hybrid overloading can achieve the same schedulability as PB overloading and a short *TTSF* close to BB overloading. When the task arrival rate is very large, the acceptance probability of BB overloading is close to PB overloading. Scheduling tasks as BB overloading will not hurt the schedulability. Even if the new task cannot find an available time slot, it can use the previous time slots of backups, just like PB overloading. Let R_{BB} denote the ratio of tasks in BB mode to all tasks in hybrid overloading chain, and R_{PB} for the tasks in PB mode. We have

$R_{BB} + R_{PB} = 1$.

The average time length of hybrid overloading chain is

$l_{t\_a}^{H} = R_{BB} \cdot l_{t\_a}^{BB} + R_{PB} \cdot l_{t\_a}^{PB}$,

and

$l_{t\_a}^{BB} \le l_{t\_a}^{H} \le l_{t\_a}^{PB}$.

R_{BB} and R_{PB} are given in a practical algorithm. An adaptive scheduling algorithm will certainly be better than one with fixed R_{BB} and R_{PB}.

5. Simulation Results

The parameters used in our simulations are summarized in Table 2.

Table 2: Parameters of simulations

Parameter	Description	Value
m	number of processors	3,…,10
l	task laxity	3,4,…,10
C_{min}	maximum execution time of tasks	25s
C_{max}	minimum execution time of tasks	5s
λ	task arrival rate	0.1,…,1

The distributions of task laxity, worst-case execution time, and time interval of task arrivals are totally the same as those of Section 4.

This paper focuses on overloading techniques and stochastic analysis for primary-backup based fault tolerant task scheduling. We do not present a detailed scheduling algorithm. There exist many optimal scheduling algorithms [1-4,7,11,12]. In order to be fair for each overloading technique, a first-come first-served scheduling policy is used in this simulation, and a task is rejected once no time slot is available. The primary of a new task is scheduled as early as possible. If the backup of a new task cannot be overloaded on any previous backup, it is scheduled closest to its primary in BB overloading. The backup of a new task is also scheduled closest to its primary in PB overloading. For hybrid overloading, we try to ensure that at least one task in a full overloading chain is scheduled as primary-backup mode.

In this simulation, we also consider the scheduling of real-time tasks with no fault-tolerance requirement. This scheduling always allocates a new task to the processor where this new task can be finished earlier.

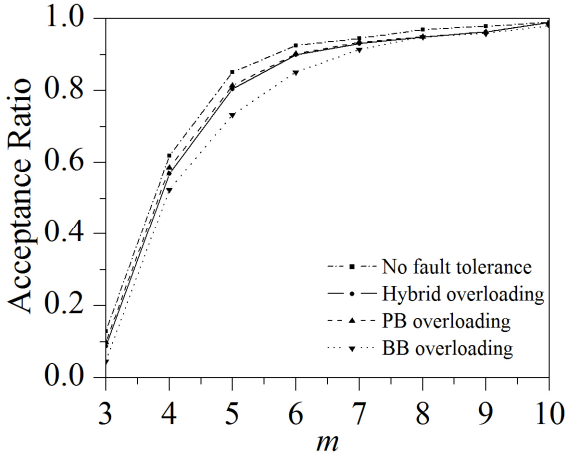

Figure 9. Acceptance Ratio (λ = 0.2, l=3~5).

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

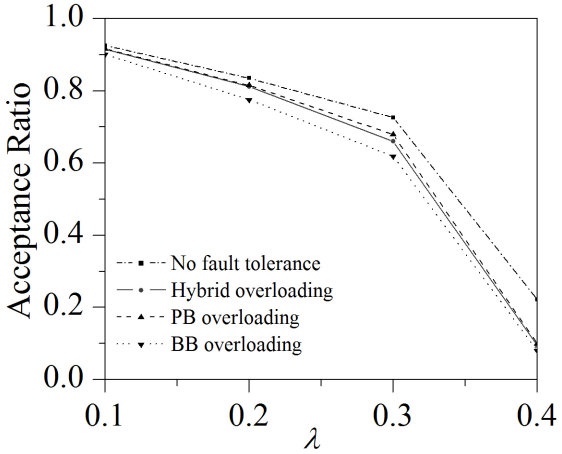

Figure 10. Acceptance Ratio (m=6, l=3~5).

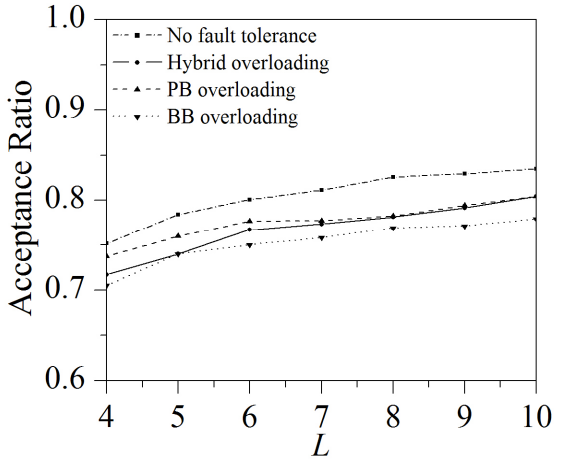

Figure 11. Acceptance Ratio (m=4, λ = 0.1).

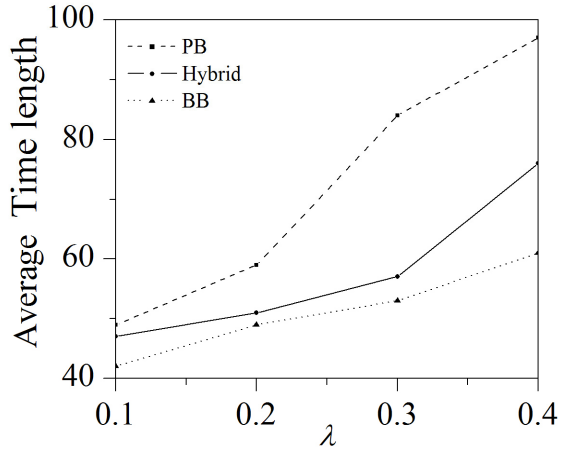

Figure 12. Average Time Length.

The simulation results are shown in Fig.9-12. The solid lines represent hybrid overloading.

The acceptance ration is the number of admitted tasks to all tasks. *TTSF* is decided by the time length of overloading chains, therefore we show the average time length in Fig.12.

We can see in Fig.9-11 that hybrid overloading can achieve an acceptance ratio similar to that of PB overloading, and in Fig.12 an average time length similar to that of BB overloading. The simulation shows that hybrid overloading provides a good tradeoff between schedulability and reliability.

Note that, with an optimized scheduling algorithm, the difference between the overloading techniques will be much larger and the advantage of hybrid overloading will then be even more obvious.

6. Conclusions

In this paper, we have extended the existing PB overloading, and proposed a hybrid overloading strategy. The hybrid overloading is a new technique which combines the advantages of both PB and BB overloading. All overloading strategies are analyzed theoretically, and we compared them through simulation. Simulation results have shown the different performance of each overloading strategy in various conditions. Through the stochastic analysis and the simulation studies, the hybrid overloading is shown to provide a good tradeoff between schedulability and reliability, i.e., better schedulability than BB overloading and better reliability than PB overloading.

In primary-backup based task scheduling, the existence of backups will necessarily reduce the schedulability. This is the cost for fault-tolerance. Overloading techniques are the methods to mitigate this cost by improving schedulability. The improvement of the schedulability always affects the reliability of a system. A good scheduler must trade well between schedulability and reliability. According to our analysis, the conflict between schedulability and reliability is caused by resource competition. A scheduling algorithm with overloading techniques should be able to improve resource utilization while preserving good reliability at the same time. Obviously, hybrid overloading is a good solution. Therefore, a practical scheduling algorithm should 1) choose proper time slots for new tasks to increase the number of overloaded tasks, 2) avoid to form looped chains, 3) exploit the flexibility of hybrid overloading to adapt the parameters R_{BB} and R_{PB}, and 4) control the time length of overloading chains within a reasonable scope when the number of processors, the task window, and the task arrival rate are very large. An adaptive scheduling algorithm with hybrid overloading is shown in [21].

References

[1] K. Ramamritham, J.A. Stankovic, "Scheduling algorithms and operating system support for real-time

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.

systems," *Proc. IEEE,* 82(1):55–67, 1994.

[2] G. Manimaran, C. Siva Ram Murthy, "An efficient dynamic scheduling algorithm for multiprocessor real-time systems," *IEEE Trans. Parallel Distributed Systems,* 9(3):312–319, 1998.

[3] K. Hashimoto, T. Tsuchiya, and T. Kikuno, "A New Fault-Tolerant Scheduling Technique for Real-Time Multiprocessor Systems," *J. Systems and Software*，53(2), pp. 159–171, 2000.

[4] S. Ghosh, R. Melhem, D. Mosse, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distributed Systems,* 8(3):272–284, 1997.

[5] H. Zou, F. Jahanian, "Real-time primary-backup (RTPB) replication with temporal consistency guarantees," *IEEE Intl. Conf. Distributed Computing Systems (ICDCS)*, pp. 48–56, May 1998.

[6] R. Al-Omari, A.K. Somani, G. Manimaran, "An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems," *J. Parallel and Distributed Computing*, 65(5):595–608, 2005.

[7] G. Manimaran, C. Siva Ram Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Trans. Parallel Distributed Systems,* 9(11):1137–1152, Nov. 1998.

[8] H. Kopetz, A. Damm, C. Koza, et al., Distributed fault tolerant real-time systems: The MARS approach, *IEEE Micro*, 9(1):25–40, 1989.

[9] K. Hashimoto, T. Tsuchiya, and T. Kikuno, "Fault-Secure Scheduling of Arbitrary Task Graphs to Multiprocessor Systems," *IEEE/IFIP Int'l Conf. Dependable Systems and Networks (DSN)*, pp. 203–212, 2000.

[10] K. Kim, J. Yoon, Approaches to implementation of reparable distributed recovery block scheme, *Proc. IEEE Fault-tolerant Computing Symposium (FTCS),* pp. 50–55, 1988.

[11] R. Al-Omari, A.K. Somani, G. Manimaran, "Efficient overloading techniques for primary-backup scheduling in real-time system," *J. Parallel and Distributed Computing*, 64(5):629–648, 2004.

[12] J.A. Stankovic, K. Ramamritham, "The spring kernel: a new paradigm for real-time operating systems," *ACM SIGOPS Oper. Systems Rev.,* 23(2):77–83, 1995.

[13] Y. Oh and S. H. Son, "Scheduling real-time tasks for dependability," *J. Operation Reserch Society*, 48(6):629–639, 1997.

[14] C. Shen, K. Ramamritham, and J.A. Stankovic, "Resource Reclaiming in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems*, 4(4):382–397, 1993.

[15] M.L. Dertouzos and A.K. Mok, "Multiprocessor On-Line Scheduling of Hard Real-Time Tasks," *IEEE Trans. Soft. Eng.*, 15(12):1497–1506, 1989.

[16] C.M. Krishna and K.G. Shin, "On Scheduling Tasks With Quick Recovery From Failure," *IEEE Trans. Computers*, 35(5):448–455, 1986.

[17] B. Kalyanasundaram and K.R. Pruhs, "Fault-Tolerant Scheduling", *ACM Symp. Theory of Computing (STOC)*, pp. 115-124, May 1994.

[18] L. Kleinrock, Queuing System. John Wiley & Sons, 1975.

[19] J.P. Lehoczky, "Real-Time Queueing Theory", *17th IEEE Real_Time Systems Symp. (RTSS)*, pp.186-195, 1996.

[20] J.P. Lehoczky, "Using Real-Time Queueing Theory to Control Lateness in Real-Time Systems", *ACM Sigmetrics*, pp.158-168, 1997.

[21] W. Sun, Y. Zhang, Y. Chen, X. Défago and Y. Inoguchi, "Real-time Task Scheduling Using Extended Overloading Technique for Multiprocessor Systems", *IEEE Int'l Symp. Distributed Simulation and Real Time Applications*, to appear.

Authorized licensed use limited to: Japan Advanced Institute of Science and Tech. Downloaded on November 10, 2008 at 20:19 from IEEE Xplore. Restrictions apply.