Computer and Information Science
www.ccsenet.org/journal.html
Vol. 2, No. 1
February 2009

Fault Recovery Mechanisms in Utility Accrual Real Time Scheduling Algorithm

Idawaty Ahmad (Corresponding author)
Faculty of Computer Science and Information Technology
University Putra Malaysia
43400 UPM Serdang Selangor, Malaysia
Tel: 60-3-8946-6541 E-mail: idawaty@fsktm.upm.edu.my
Muhammad Fauzan Othman
Motorola Multimedia Sdn Bhd
3507 Prima Avenue, Jalan Teknokrat 5, Cyberjaya 63000 Malaysia
Tel: 60-3-8314-6777 E-mail: fauzan.othman@motorola.com

Abstract

In this paper, we proposed two recovery solutions over the existing error-free utility accrual scheduling algorithm known as General Utility Accrual Scheduling algorithm (or GUS) (Peng Li, 2004). A robust fault recovery algorithm called Backward Recovery GUS (or BRGUS) works by adapting the time redundancy model i.e., by re-executing the affected task after its transient error period is over. The BRGUS is compared with a less complicated recovery algorithm named as Abortion Recovery GUS (or ARGUS) that simply aborts all faulty tasks. Our main objectives are (1) to maximize the total accrued utility and (2) to ensure correctness of the executed tasks on best effort basis and achieve the fault free tasks as much as possible. Our simulation results reveal that BRGUS outperforms the ARGUS algorithm with higher accrued utility and less abortion ratio, making it more suitable and efficient in adaptive real time system.

Keywords: Adaptive Real-Time System, Utility Accrual Scheduling, Fault Recovery, Time Redundancy, Discrete Event Simulation

1. Introduction

A real time system is a system where the time at which events occur is important. In adaptive real time system, the deadline misses and delays during overloads are tolerable and do not have great consequences. The definition of deadline constraints in existing deadline based scheduling algorithms such as Earliest Deadline First (or EDF) is limited in expressiveness by its singular metrics. A clear distinction has been made between the urgency and the important of a task by Jensen and it is known as time/utility functions (or TUFs) (Jensen, 1985; Locke, 1986). As illustrated in Figure 1, the urgency is measured as a deadline on X-axis and importance is captured by utility in Y-axis. A task's time constraint expresses the utility for completing a task as a function of when the task is completed. As shown in Figure.1, completion of a task within its initial time and termination time will accrued some utility or zero utility otherwise. We specify the deadline constraint of a task a binary value, downward step shaped TUF as shown in Figure 1.

The scheduling optimality criteria are based on maximizing accrued utility from those tasks. These criteria are named as Utility Accrual (or UA) (Wu, 2004). A closer look at the UA algorithms in (Ravindran, 2005; Edward, 2007) indicates that only the Aborted-Assured Utility Accrual (or AUA) and Handler-Assured Utility Accrual (or HUA) algorithms consider fault in its scheduling decision. These algorithms consider the abortion and released handler for all task failures. To the best of our knowledge, none of the prior works in UA scheduling domains consider time redundancy in their fault recovery design model. In this paper, we apply the time redundancy model for fault recovery in UA scheduling domain.

2. Literature Review

2.1 Fault

The term error and fault are often used as synonym. However, the definition in (Sasikumar, 1997) is appropriate in the context of problems analyzed in this study. A fault is a defect in component or design of a system. Fault can be categorized as permanent and transient. Permanent faults include hardware breakdowns, connection disruption as well

as design errors. A transient fault category is when an error disappear shortly after it appearance. These types of faults are caused by software design error or environmental variations. We focused on transient faults since previous studies in (Sasikumar, 1997; George, 2003) have indicated that majority of faults observed during the lifetime of a system are caused by transient faults.

2.2 Fault Recovery

Almost every fault recovery mechanism relies on some form of redundancy. Space redundancy is employed by extra hardware and software component that is introduced in the system only for fault recovery purposes (Mejia, 1994). A less expensive approach is via the use of time redundancy, which is based on repeating the computation and typically does not require a large amount of extra resources (Sasikumar, 1997). This paper focused on time redundancy paradigm, since it is suitable for non-distributed and uniprocessor environment.

Two main approaches for fault recovery are backward and forward recovery technique. In forward recovery technique, the system does not roll back to its previous safe state and it allows for another available resource to perform recovery. This technique works well in distributed system environment whereas extra resources are widely available. In contrast, the backward technique attempts to take the system back to its previous safe state and then proceeds to re-execute the affected task (Sasikumar, 1997).

3. The Proposed Algorithms

3.1 Task Model

During the lifetime of a task, it may request one or more resources. As shown in Figure 2, a task specifies duration to hold the requested resource in *holdtime*. We apply the Jensen's TUFs (Jensen, 1985; Locke, 1986) to define the time constraints of a task. Each task has an initial time and a termination time. If the termination time of a task is reached and the task has not completed its execution, it will then be aborted. Aborting a task will change the task state from Normal to Abort mode. Completion of task before the deadline in Normal mode accrues some uniform utility and accrues zero utility otherwise. Following (Peng Li, 2004), our proposed algorithm measures the metric called Potential Utility Density (or PUD) that was originally developed in (Jensen, 1985). The PUD of a task measures the amount of utility that can be gained per unit time by executing the task. Thus, executing task in Abort mode will accrues zero PUD and accrued zero utility to the system.

3.2 Fault Definition

In our fault model we assumed that the system have error containment capabilities to prevent the propagation of a specific error to other tasks. The transient faults in a request can be effectively overcome by re-execution of the request in the affected task. A request is suspended temporarily to model the transient error defects. Figure 3 shows the procedures to detect and simulate the transient error in the system. The fault occurrences and its duration follow the exponential distribution as detailed in Table 1. We further assumed that no error occurred during the fault recovery process.

3.3 The Fault Recovery Algorithms

Figure 4 shows a description of Backward Recovery GUS (or BRGUS) and Abortion Recovery GUS (or ARGUS) for error recovery. The recovery process follows three stages and is described as follows:

1) The transient erroneous period of the erroneous task, *Trec* is over.

After the transient error is over, the task needs to release the resource before executing the recovery algorithm in stage 2.

2) Executes the fault recovery algorithm, either ARGUS or BRGUS

In BRGUS, after the erroneous period of a request is over, the time taken to re-execute the request (i.e., *holdtime)* is measured. The system is rolled back to its previous safe state (i.e., before error occurred) and then proceeds to re-execute the affected task (Sasikumar, 1997). A request is eligible for re-execution if the *holdtime* of the request does not exceed the remaining execution time of a task. Otherwise, the task will be aborted since re-executing the task will finally result in abortion later during termination time. The *AbortTime* is the time taken for the task to abort the resource. Our proposed recovery algorithm works in a best-effort basis, in the sense that a task is simply aborted if it does not have enough time to be re-executed or continue re-execution otherwise. In ARGUS, all faulty tasks are simply aborted and no error recovery performed (Edward, 2007).

3) Check the availability of the requested resource.

If the related resource is available, the requesting task continues its re-execution or abortion procedure directly. On the other hand, if the resource is busy and currently being used by other task, the requesting task has to wait for the resource in the unordered task list (or *utlist*).

4. Experimental Model

4.1 Simulation Model

We developed a discrete event simulator to verify the performance of our proposed algorithm. We used experiment settings that are similar to those in (Peng Li, 2004) for comparison purpose. Figure 5 shows the entities involved in the simulation model. It consists of a stream of 1000 tasks that are exponentially generated, an unordered task list (or *utlist)* and a set of active resources. The requests from tasks are queued in the unordered task list before it can use the resources. The scheduling algorithm decides which request to be executed by calculating the PUD of the request. The tasks are assumed to be independent of each other. The events defined in the simulation model are the arrival of a task, the completion of a task, a resource request, a resource release, the arrival of termination time of a task, the arrival of fault in a request and the arrival of a fault recovery in a request. Table 1 summarized the details task settings used in our model.

4.2 Performance Metric

The Accrued Utility Ratio (or AUR) metric defined in (Jensen, 1985) has been used in many algorithms stated in (Locke, 1986; Wu, 2004; Peng Li, 2004; Ravindran, 2005; Edward, 2007) and can be considered as standard metric in UA scheduling domain. AUR is defined as the ratio of accrued aggregate utility to the maximum possibly attained utility. The Abortion Ratio (or AR) is the ratio of aborted task to the total of task in the system. The Average Response Time (or ART) measures the time taken for a task to complete its execution.

5. Results and Analysis

Figure 6 shows the AUR results under an increasing load. The error-free in GUS is the upper limit of our proposed recovery algorithms. We observed that BRGUS algorithm accrued higher utility compared to ARGUS for the entire load range. Clearly, the attempt to re-execute the faulty tasks significantly improved the accrued utility in BRGUS. Since aborted tasks produced zero utility, this caused ARGUS to accrue lower utility. Figure 7 plots the success ratio of BRGUS and ARGUS under an increasing error rate. We observed that the number of successfully executed tasks decreases as the error rate and load is higher. Figure 8 verifies our speculation, proving that the abortion ratio in BRGUS is lower than in ARGUS, which ultimately leads to higher utility accrued.

Figure 9 shows the ART effects of the proposed algorithms. Although BRGUS accumulates higher ART compared to ARGUS, it is always lower than its execution time (i.e., 0.5seconds). This is identified as BRGUS tradeoff, whereby the extra time taken is the time overhead incurred for re-execution of the faulty tasks during recovery process. The more tasks get re-executed, the higher utility that BRGUS is able to accrue back to the system

6. Conclusion

This paper presents quantitative results of a best effort UA real time scheduling algorithm called BRGUS that applied time redundancy paradigm for fault recovery. Simulation studies shows that the BRGUS achieved higher accrued utility with smaller abortion ratio, conforming to our design objectives. Future work includes implementing in real time operating system (RTOS) to observe the actual behavior of BRGUS algorithm.

References

Edward A.Curley (2007). Recovering From Distributable Thread Failures with Assured Timeliness in Real Time Distributed System. *Master Thesis*, Virginia Polytechnic Institute and State University

George Marconi de Araujo Lima (2003). Fault Tolerance in Fixed Priority Hard Real Time Distributed Systems. *PhD dissertation*, University of York

Jensen,E.D., Locke,C.D. and H.Tokuda (1985). A time driven scheduling model for real time system. *Proceeding of Real Time System Symposium*, 112-212

Locke,C.D. (1986). Best-effort decision making for real time scheduling. *PhD dissertation*, Carnegie Mellon University

Mejia Alvarez.P and D.Mosse (1994). A Responsiveness Approach for Scheduling Fault Recovery in Real Time Systems. *Proceeding of Fifth IEEE Real-Time Technology and Applications Symposium*,4-13

Peng Li (2004). Utility Accrual Real Time Scheduling" Model and Algorithm. *PhD dissertation*, Virginia Polytechnic Institute and State University

Ravindran,B., Jensen,E.D., Peng Li (2005). On Recent Advanced in Time/Utility Function Real-Time Scheduling and Resource Management. *Proceeding of Eight IEEE International Symposiums on Object-Oriented Real-Time Distributed Computing (ISORC),* 55-60.

Sasikumar Punnekat (1997). Schedulability Analysis for Fault Tolerant Real Time Systems. *PhD dissertation*, University of York

Wu,H., Ravindran,B. and Peng Li (2004). CPU Scheduling for Statistically Assured Real-Time Performance and

Improved Energy Efficiency. *Proceeding* of the 2nd *IEEE/ACM/IFIP International Conference of Hardware/Software Codesign and System Synthesis*, 55-60

Table 1. Simulation Parameters

Parameters	Value	Descriptions
load	0.2 to 1.5	Range of load
C_AVG	0.5 seconds	Task Average Execution Time
iat	Exponential(C_AVG/load)	Task Inter-Arrival Time
holdtime	Normal (0.25,0.25)	Time duration for holding a resource
max_au	Normal (10,10)	Task's Maximum Utility
mean_tasks	Exponential (0.1)	Inter- arrival of error in the system
MEAN_DURATION	Exponential (0.1)	Duration of error (transient)
MAX_RESOURCES	5	Number of available resources

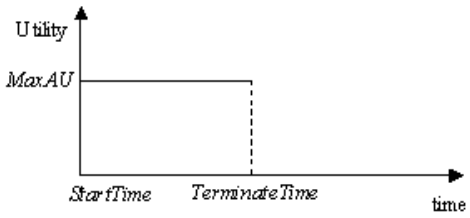

Figure 1. The Step TUF [1], [4]

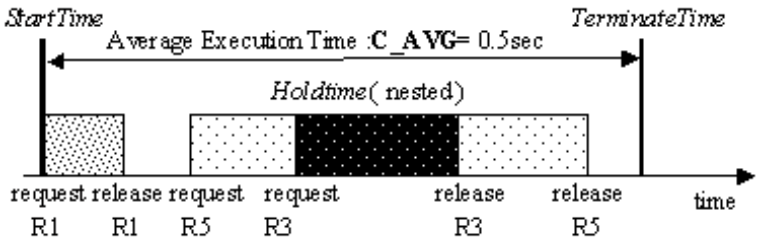

Figure 2. Task Model

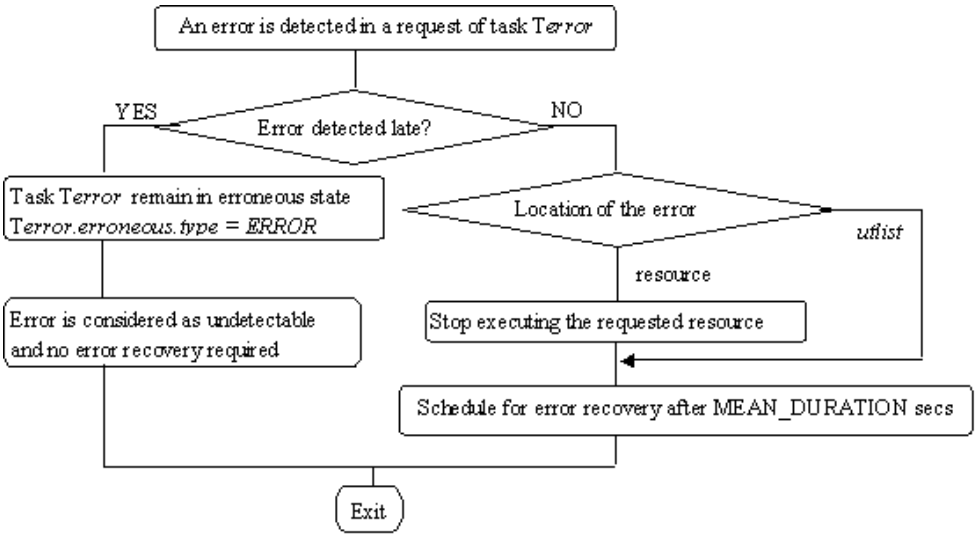

Figure 3. Fault Assignment

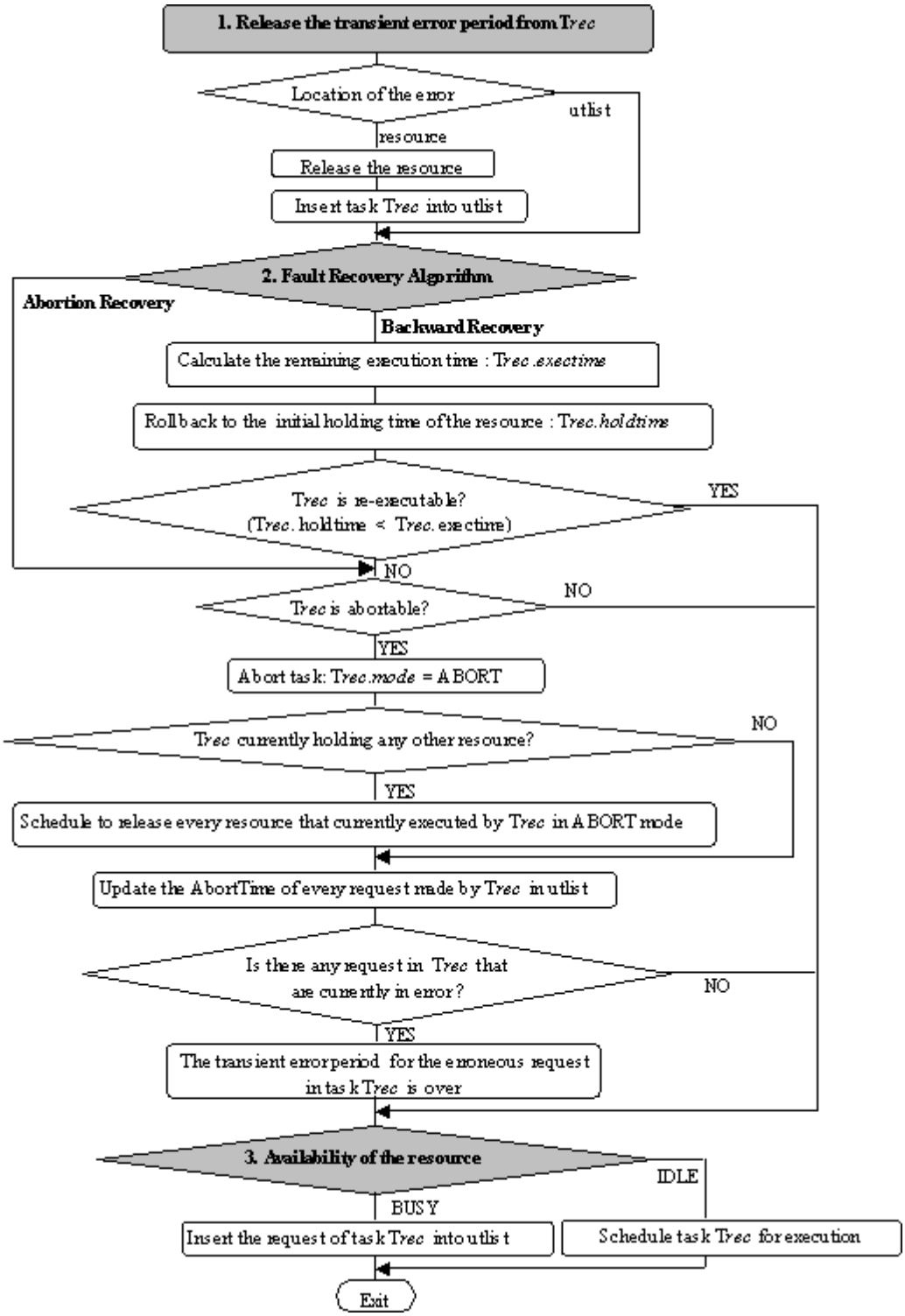

Figure 4. Fault Recovery Algorithms

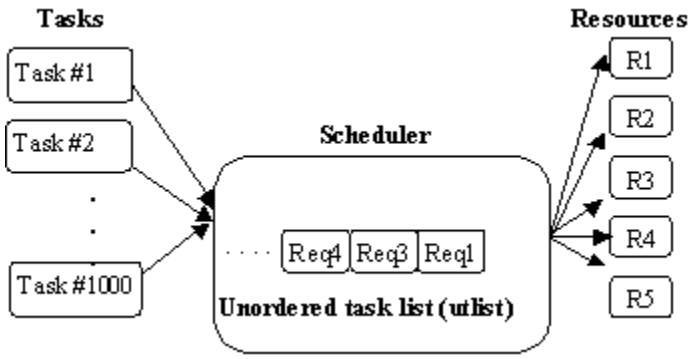

Figure 5. Simulation Model

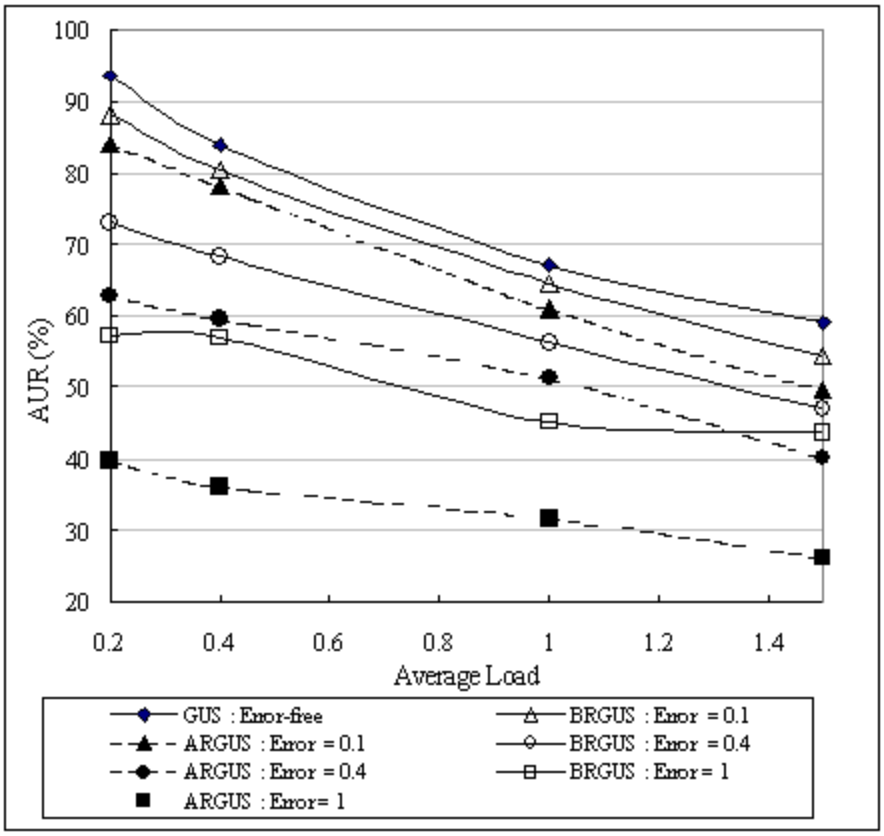

Figure 6. Accrued Utility Ratio (AUR) vs. Average Load

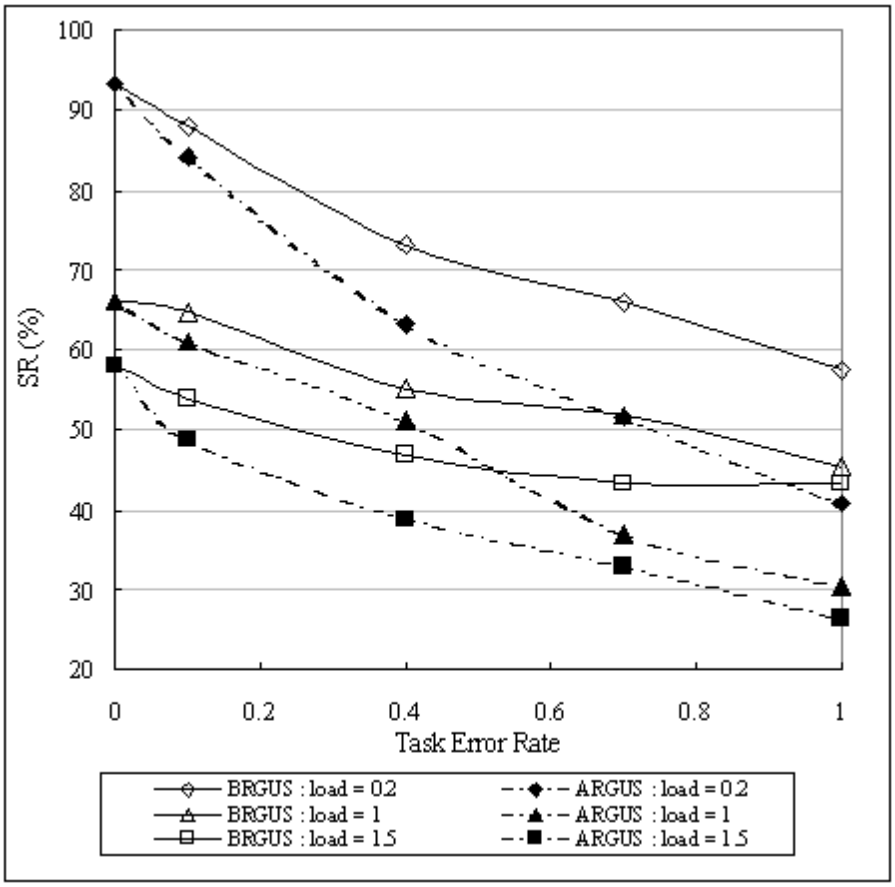

Figure 7. Success Ratio (or SR) vs. Task Error Rate

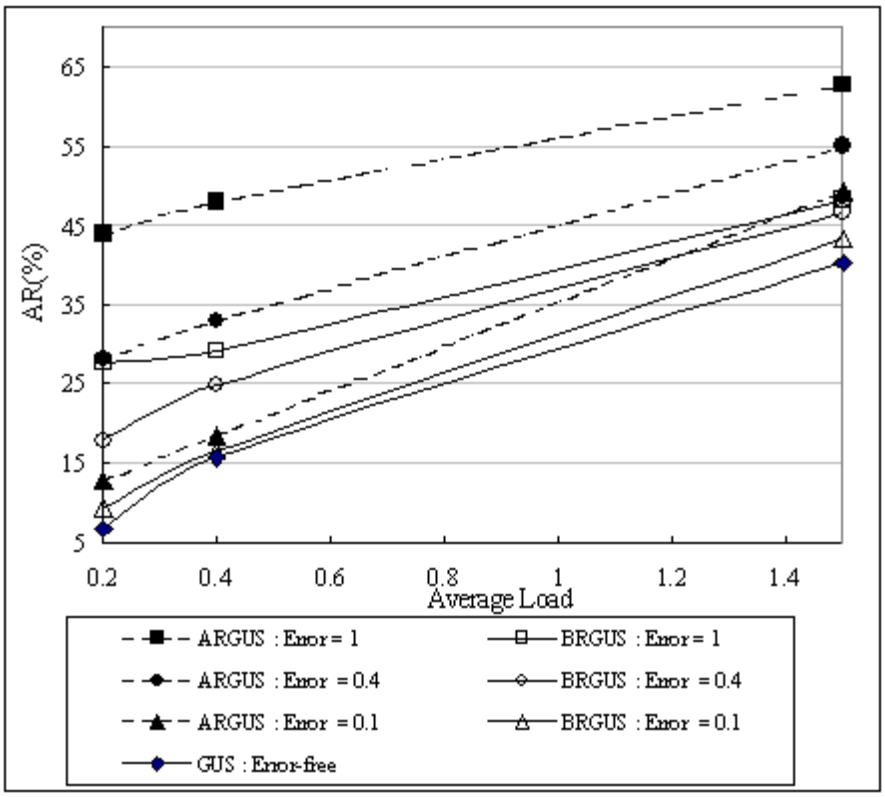

Figure 8. Abortion Ratio (or AR) vs. Average Load

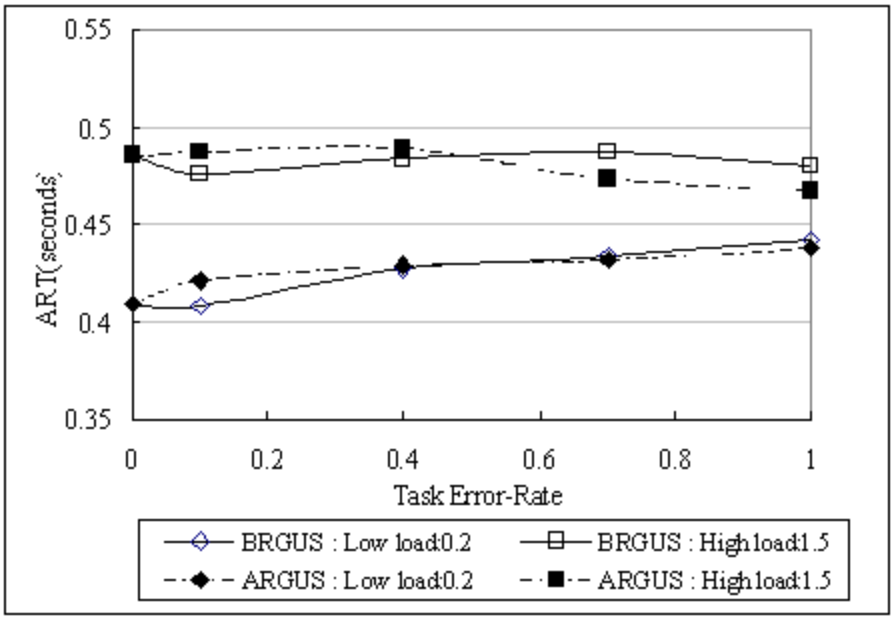

Figure 9. Average Response Time (or ART) vs. Task Error Rate