Journal of Computer Science 6 (7): 728-734, 2010
ISSN 1549-3636
© 2010 Science Publications

A Backward Recovery Mechanism in Preemptive Utility Accrual Real Time Scheduling Algorithm

[1]Idawaty Ahmad and [2]Muhammad Fauzan Othman
[1]Department of Communication Technology and Network,
Faculty of Computer Science and Information Technology,
University Putra Malaysia, 43400 UPM, Serdang, Selangor DE, Malaysia
[2]Engineering Application and System Support Division,
Motorola Multimedia Sdn Bhd 3507 Prima Avenue, Jalan Teknokrat 5,
63000 Cyberjaya, Malaysia

Abstract: Problem statement: This study proposed a robust algorithm named as Backward Recovery Preemptive Utility Accrual Scheduling (BRPUAS) algorithm that implements the Backward Recovery (BR) mechanism as a fault recovery solution under the existing utility accrual scheduling environment. The problem identified in the TUF/UA scheduling domain is that the existing algorithms only considers the Abortion Recovery (AR) as their fault recovery solution in which all faulty tasks are simply aborted to nullify the erroneous effect. The decision to immediately abort the affected tasks is inefficient because aborted tasks produce zero utility causes the system to accrue lower utility. **Approach:** The proposed BRPUAS algorithm enabled the re-execution of the affected tasks rather than abortion to reduce the number of aborted task in the existing algorithm known as Abortion Recovery Preemptive Utility Accrual Scheduling (ARPUAS) algorithm that employed the AR mechanism. The BRPUAS ensure the correctness of the executed tasks in the best effort basis in such a way that the infeasible tasks are aborted and produced zero utility, while the feasible tasks are re-executed to produce positive utility and consequently maximized the total accrued utility to the system. The performances of these algorithms are measured by using discrete event simulation. **Results:** The proposed BRPUAS algorithm achieved higher accrued utility compared to ARPUAS for the entire load range. **Conclusion:** Simulation results revealed that the BR mechanism is more efficient than the existing AR mechanism, producing higher accrued utility ratio and less abortion ratio making it more reliable and efficient for adaptive real time application domain.

Key words: Adaptive real time, utility accrual scheduling, fault recovery, discrete event simulation

INTRODUCTION

A real time system is a system where the time at which event occurs is important. Real-time scheduling is fundamentally concerned with satisfying an application time constraints. In adaptive real time system an acceptable deadline misses and delays are tolerable and do not have great consequences to the system. For this type of system, a failure, though never desirable, degrades the reliability performance of the system. Thus, one has to build a system as resilient to fault as possible. Increasing the fault resilience in adaptive real-time systems is the focus of this study.

The latest scheduling paradigm in adaptive real time system environment is known as Time Utility Function/Utility Accrual (TUF/UA) scheduling (Wu *et al*., 2004). A TUF specifies the utility of completing a task as an application function of when the task completes as shown in Fig. 1. The urgency of a task is captured as a deadline on X-axis and the importance of a task is measured by utility in Y-axis. The completion of a task within the deadline (i.e., within the start time and terminate time) will accrue some positive utility (i.e., MaxAU) or zero utility otherwise.

The scheduling optimality criteria of TUF/UA are based on maximizing the total accrued utility from execution of all tasks in the system. These criteria are named as Utility Accrual (UA) criteria (Jensen *et al*., 1985). A TUF/UA scheduling algorithm that maximizes the sum of tasks' attained utilities will seek to meet all task deadlines and naturally tend to favor task that are more important from whom higher utility can be accrued.

Corresponding Author: Idawaty Ahmad, Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 UPM, Serdang, Selangor DE, Malaysia

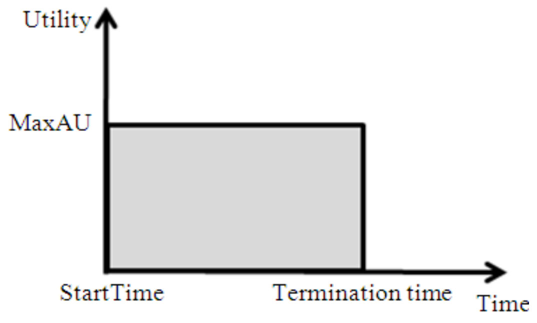

Fig. 1: The step TUF (Wu *et al*., 2004; Jensen *et al*., 1985)

Objective: The scheduling objectives of this research are to:

- Maximize the total accrued utility from all executed tasks in the system
- Ensure correctness of the executed tasks on best effort basis and achieve the fault free tasks as much as possible to increase the reliability of the system

Problem statement: Although the BR mechanism is widely integrated in real time scheduling algorithms such as EDF and RM (Brzezinski *et al*., 1995; Sahoo and Ekka, 2007), none of the existing TUF/UA scheduling algorithms consider the BR mechanism as their recovery solution. It is observed that the existing TUF/UA scheduling algorithms utilize the AR mechanism for their fault recovery mechanism (Edward, 2007; Fahmy *et al*., 2008) where the faulty task is aborted to perform recovery when a task encounters an error during its execution. The intuition to abort the faulty task (without repeating the same computation) was to accelerate the recovery time to release resources so that the resources can be used by another task as soon as possible.

However, it is observed that the decision to abort the faulty task in AR mechanism is inefficient because the aborted tasks produced zero utility thus resulting less total utility accrued to the system. Figure 2 illustrates the inefficiency scenario of the AR mechanism. The task characteristics of this scenario are depicted in Table 1. A task is generated at time 1.00 and its termination time (i.e., deadline) is at 1.50. A request for a resource occurs at time 1.10 and the duration to hold the resource is 0.15 sec indicated by the HoldTime parameter. After using the resource for 0.05 sec, an error occurs at time 1.15. The duration of the transient error denoted by TransientPeriod is 0.10 sec. In the AR mechanism, after the transient error period is over at time 1.25, the affected task is aborted.

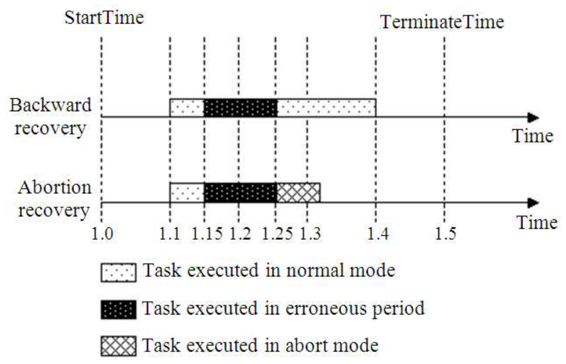

Fig. 2: Inefficiency scenario in the AR mechanism compared to the BR mechanism

Table 1: Task characteristics

Task parameters	Value
HoldTime	0.15 sec
AbortTime	0.08 sec
TransientPeriod	0.10 sec
Maximum utility (MaxAU)	9.00 sec

The aborted task executes the resource for 0.08 sec i.e., the time taken to release the resource indicated by the AbortTime parameter. Since the aborted task will not contribute any positive utility, the AR mechanism in this scenario accrued zero utility to the system.

Thus, it is important to observe that by reducing the number of aborted tasks, it is very likely that we would accrue higher utility to the system. Figure 2 illustrates the scenario of BR mechanism where the affected task is re-executed instead of being aborted after the transient error period is over at time 1.25. Since the time to hold the resource i.e., HoldTime is 0.15 sec, the re-execution of this task completed at time 1.40 before the deadline and produced 9 accrued utility to the system.

Approach: This research considers the BR mechanism to reduce the number of aborted tasks in the AR mechanism. The AR and BR recovery mechanisms are compared and executed under the existing TUF/UA scheduling environment as stated below.

Backward Recovery Preemptive Utility Accrual Scheduling (BRPUAS) algorithm: This algorithm implements the BR mechanism where the task is rolled back to its initial state and then proceeds to re-execute the affected request within the task. This algorithm ensure the correctness of the executed tasks in the best effort basis in such a way that the infeasible tasks are aborted and produced zero utility, while the feasible tasks are re-executed to produce positive utility and consequently maximized the total accrued utility to the system.

Figure 3 elaborates the BR mechanism in BRPUAS algorithm. After an erroneous request is detected in a task i.e., Trec, the time taken to re-execute the request known also as HoldTime is placed into a feasibility test to check whether the request is eligible for re-execution. A task is feasible if the re-execution of the affected request does not exceed the termination time of the task. The remaining execution time of task Trec before its termination time denoted as ExecTime is measured. The calculation of the ExecTime can be done by capturing the termination time of task Trec (i.e., TerminateTime) and subtracts it with the current simulation clock time denoted as sclock. If the value of HoldTime is more than the ExecTime, this indicates that the task is infeasible and the re-execution of the request will exceed termination time which ending up the task to be eventually abort later on. In this case, the re-execution is omitted and task Tec is aborted. The status of resources that are currently held by task Trec is changed to ABORT mode and the resources are executed according to their AbortTime.

If task Trec is feasible indicated by the value of the HoldTime that is less than the ExecTime, then the affected request is re-executed in NORMAL mode to perform the computation once again. The status of the resource is changed to BUSY state and task Trec is set as the owner of the resource.

Abortion Recovery Preemptive Utility Accrual Scheduling (ARPUAS) algorithm: This algorithm implements the AR mechanism where all faulty task is aborted after the transient error period is over. Figure 3 gives the details the scheduling decision made by the AR mechanism in ARPUAS after the erroneous period for a request in a task, Trec is over. After an erroneous request is detected in a task i.e., Trec, the resources that are currently held by task Trec are aborted and the resources are executed according to their AbortTime. This mechanism is simple because all the faulty tasks are simply aborted to enable recovery.

MATERIALS AND METHODS

We developed a Discrete Event Simulator (DES) to verify the performance of our proposed algorithms. The rationale of using DES lies in the fact that most of the research in TUF/UA scheduling paradigm are based on the discrete event simulation tools (Jensen *et al*., 1985; Ravindran *et al*., 2005; Li *et al*., 2006). Therefore, in order to precisely model the fault recovery algorithms, DES written in C language is the best method to achieve this objective. Figure 4 shows the entities involve in our simulation study. It consists of a stream of 1000 tasks, a queue of an unordered task list, the scheduler and a set of resources.

Task model: The task model is shown in Fig. 5. The average execution time i.e., ExecTime for a task is 0.50 sec. Each task has an initial time and a termination time. Initial time is the earliest time for which the utility of a task is defined and termination time is the latest time for which the utility is defined. That is, utility is defined in the time interval of (StartTime, TerminateTime) for each task. Beyond that, the utility is undefined.

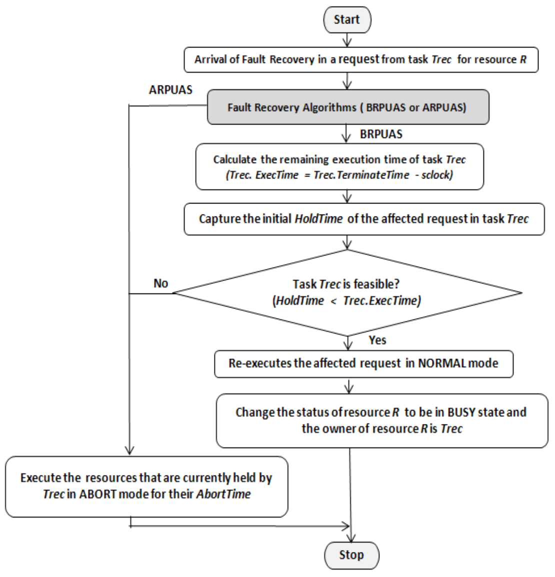
Start
Arrival of Fault Recovery in a request from task Trec for resource R
ARPUAS
Fault Recovery Algorithms (BRPUAS or ARPUAS)
BRPUAS
Calculate the remaining execution time of task Trec (Trec. ExecTime = Trec.TerminateTime - sclock)
Capture the initial HoldTime of the affected request in task Trec
No
Task Trec is feasible? (HoldTime < Trec.ExecTime)
Yes
Re-executes the affected request in NORMAL mode
Change the status of resource R to be in BUSY state and the owner of resource R is Trec
Execute the resources that are currently held by Trec in ABORT mode for their AbortTime
Stop

Fig. 3: Flow charts of the fault recovery mechanisms

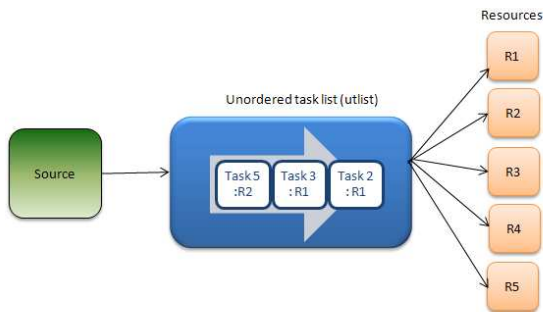

Fig. 4: Simulation model

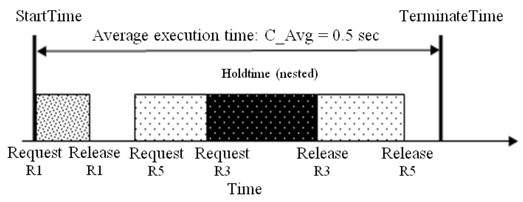

Fig. 5: Task model

During the lifetime of a task, it may request one or more resources. In general, the requested time intervals of holding resource maybe overlapped. A task specifies the duration to hold the requested resource in HoldTime parameter. The duration to hold a resource is randomly generated following the normal distribution as depicted in Table 2. The scheduler uses the HoldTime information at run time to make scheduling decisions.

Table 2 summarized the details task settings configured for the simulation model. The arrival times of tasks into the system (i.e., iat) are random which follows exponential distribution. Each task has its maximum utility that could possibly accrued by the system from the task if it is completed within its deadline. We refer this value as MaxAU.

If task has not completed its execution, it will then be aborted. Abortion of a task usually involves necessary cleanup operating by both the system software and the exception handlers in the application. We refer to the time consumed by this cleanup as AbortTime.

Fault model: The fault model is a set of assumptions on the kind of faults that are possible to occur in the system. During the execution of a task it may request one or more resources. These requests may encounter error such as request execution failure, request queuing failure, resources error and external triggers that occurs during the runtime of a task. It is assumed that these transient software faults that occur in a request can be effectively recovered by re-execution or abortion of the affected request. The fault model defined in this research is shown in Fig. 6. Three steps are taken in order to induce an error into a request of a task as stated below.

Step 1: For the erroneous task i.e., Terror, the erroneous request is randomly chosen among the entire possibly available request within that task

Step 2: For this case, the request and the resource that is currently being used by the task denoted as rid is freezed for TransienPeriod to model the transient default

Step 3: The next task to be in error state i.e., Terror is determined randomly following the exponential distribution with mean error rate denoted by mean_tasks

Resource model: The resource model represents the physical and logical resources. Examples of physical resource include disks or network interfaces for performing disk I/O or network I/O, respectively. An example of logical resource is critical sections of source codes in real time applications. To model the resources the following assumptions are made:

- Resources are reusable and can be shared but have mutual exclusion constraints. Thus, only one task can be using a resource at any given time
- Only single instance of a resource is presented in the system
- A resource request from a task can only request a single instant of the resource. If multiple resources are needed for a task to make progress, the task must acquire all resources through a set of consecutive resource requests

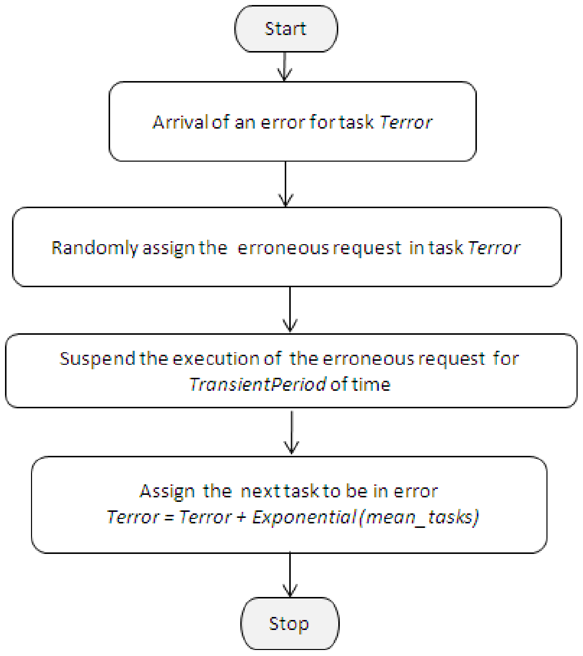
Start
Arrival of an error for task *Terror*
Randomly assign the erroneous request in task *Terror*
Suspend the execution of the erroneous request for *TransientPeriod* of time
Assign the next task to be in error
Terror = Terror + Exponential (mean_tasks)
Stop

Fig. 6: Fault model

Table 2: Simulation parameters

Parameter	Range	Description
iat	Exponential (C_AVG/load)	Task inter-arrival time
HoldTime	Normal (0.25, 0.25)	Duration for holding a resource.
MaxAU	Normal (10, 10)	Task maximum utility
AbortTime	Any random number that is less than HoldTime	Duration for cleanup time of a task before it can releases the resources.
Transient period	0.10 sec	Duration of transient error of a request
ExecTime	0.50 sec	Average task execution time
Mean_task	0.1, 0.5 or1.0	Task error rate

RESULTS

The performances of TUF/UA scheduling algorithms are measured by the metrics that relies on the application specifications. For TUF/UA scheduling domain, the Accrued Utility Ratio (AUR) metric defined in (Ravindran *et al*., 2005) has been used in many algorithms (Wu *et al*., 2004; Ravindran *et al*., 2005; Li *et al*., 2006) and considered as a standard metric in this domain. AUR is defined as the ratio of accrued aggregate utility to the maximum possibly attained utility. Figure 7 shows the comparison of the fault recovery algorithms i.e., BRPUAS and ARPUAS in three different error rates i.e., 0.1 (lowest), 0.5 (medium) and 1.0 (highest). The value of 0.1, 0.5 and 1.0 indicates that almost 10, 50 and 100% of the total executed tasks, respectively experienced transient fault during their execution. The nature of the curves in Fig. 6 clearly indicates that the proposed BRPUAS algorithm achieved better performance by producing higher accrued utility compared to ARPUAS for every error rates.

In the highest error rate, BRPUAS accrued almost 24.6% of the utility compared to ARPUAS that accrued almost 1% utility to the system. As the error rate decreases i.e., in the medium error rate, BRPUAS accrued higher utility i.e., 50.6% and ARPUAS accrued 38.5% to the system. In the lowest error rate, BRPUAS accrued almost 71.6% utility compared to 66.5% accrued by ARPUAS. The higher utility accrued in BRPUAS is because the BR mechanism eliminates the abortion that occurs to erroneous tasks and it re-executes the affected tasks in best effort manner that possibly produces positive utility to the system. Thus, lead to greater accrued utility rather than AR mechanism in ARPUAS that abort erroneous tasks which definitely produces zero utility to the system. Since aborted tasks produce zero utility, consequently ARPUAS produces more zero utility tasks that ultimately contributed to lower accrued utility compared to BRPUAS.

In addition, we consider two other metrics to precisely examine the effectiveness of our proposed algorithms. The Success Ratio (SR) is the ratio of task successfully attained positive utility to the total task executed in the system. The SR supports the result of AUR because it measures the exact number of tasks that contributed to AUR.

Figure 8 plots the success ratio performances of the BRPUAS and ARPUAS algorithms. It highlights the improvement of BRPUAS compared to ARPUAS in the highest load. In the highest error rate, the BRPUAS achieved almost 32% compared to ARPUAS that acquired only 7% of the successful tasks.

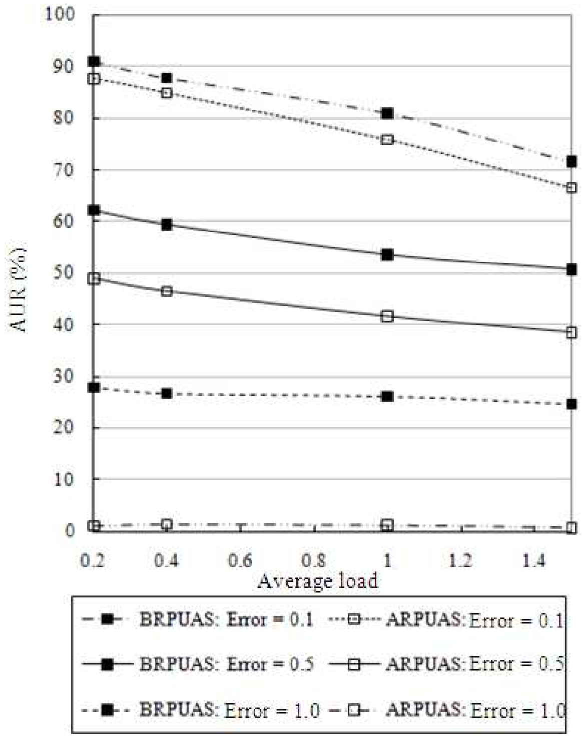

Fig. 7: AUR results

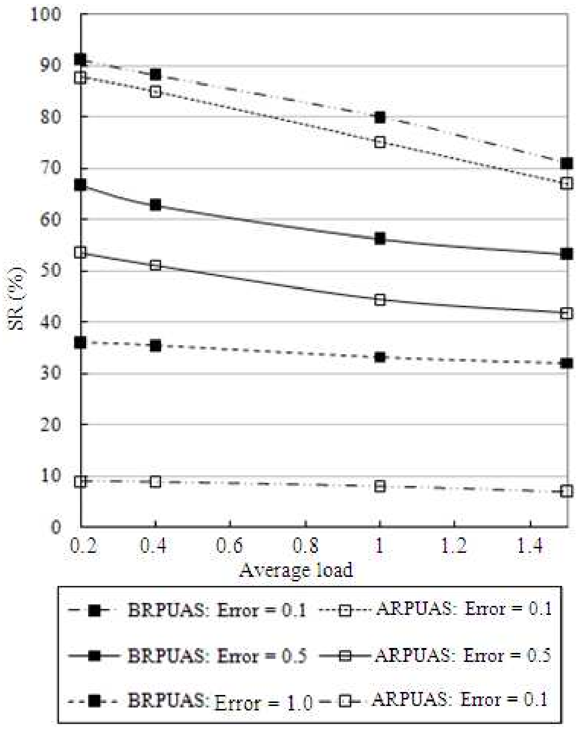

Fig. 8: SR results

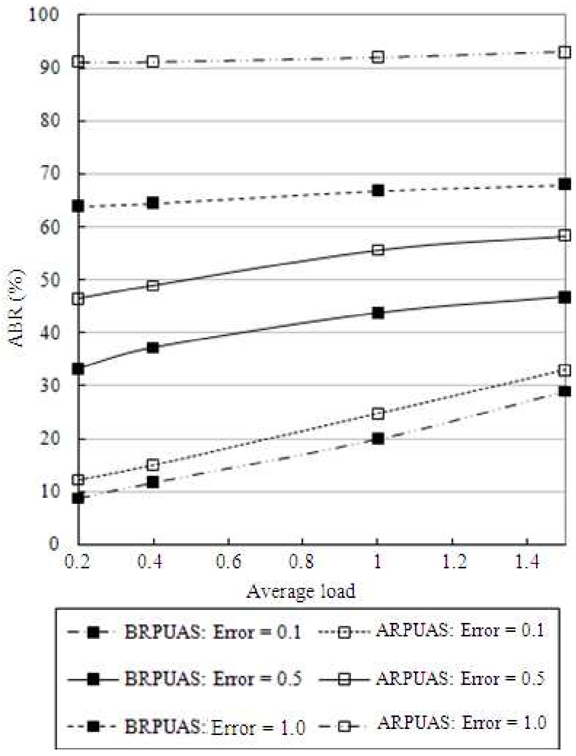

Fig. 9: ABR results

As the error rate decreases, higher successful tasks are recorded in which BRPUAS achieved almost 53% and ARPUAS gained 41% of successful tasks ratio. In the lowest error rate, the BRPUAS achieved almost 71% and ARPUAS acquired 67% of the successful tasks.

Figure 8 supports the AUR results shown in Fig. 7 because it measures the exact number of tasks that has successfully contributed to AUR. The curves clearly indicates that the proposed BR mechanism in BRPUAS achieved better performance by producing higher number of tasks that has accrued positive utility to the system compared to AR mechanism in ARPUAS. Thus, it proves that the reason of BR mechanism acquired higher utility compared to the AR mechanism is specifically because of the increases on the number of tasks that has successfully contributes utility to the system that lead to greater accrued utility.

Figure 9 plots the Abortion Ratio (ABR) performances of the BRPUAS and ARPUAS algorithms. The Abortion Ratio (ABR) is defined as the ratio of aborted tasks to the total of executed tasks. It is observed that the BR mechanism in BRPUAS is able to reduce the number of abortion as compared to the AR mechanism in ARPUAS for every error rates. The attempt to re-execute the affected tasks in BRPUAS has prevented the tasks from being aborted compared to the ARPUAS that simply abort erroneous tasks and thus failed to contribute any utility to the system. Therefore, lower abortion ratio is recorded for the BR mechanism in BRPUAS.

DISCUSSION

The proposed BRPUAS algorithm achieved the best performances with higher accrued utility, success ratio and lowest abortion ratio compared to the ARPUAS algorithm. In general, our proposed algorithms BRPUAS have successfully reduced the number of aborted tasks in ARPUAS that ultimately contributed to reliable and higher accrued utility to the system.

CONCLUSION

In this study we proposed an efficient TUF/UA fault recovery scheduling algorithm called BRPUAS that considers task subjected to deadline expressed using step TUFs. The proposed BRPUAS algorithm implemented the BR recovery mechanism to overcome the abortion problem that occurs in the ARPUAS algorithm that implemented the AR mechanism. Simulation results reveal that BRPUAS outperform the ARPUAS with highest accrued utility and lowest abortion ratio making it more reliable and efficient in real time application domain.

A number of extensions to this research can be carried out and are given as follows:

- The algorithms can be deployed in network and distributed environment. Flow control and routing algorithms should be integrated into the research. Thus, increasing the feasibility in actual implementation of the algorithms
- The real implementation of BRPUAS on real-time POSIX-compliant operating system using the meta-scheduling framework can also demonstrates the effectiveness of this algorithm

REFERENCES

Brzezinski, J., J.M. Helary and M. Raynal, 1995. Semantics of recovery lines for backward recovery in distributed system. J. Annall. Telecommun., 50: 874-887. DOI: 10.1007/BF03005244

Edward, A.C., 2007. Recovering from distributable thread failures with assured timeliness in real time distributed system. Proceeding of the 25th IEEE Symposium on Reliable Distributed System, Oct. 2-4, IEEE Xplore Press, Leeds, United Kingdom, USA., pp: 267-276. DOI: 10.1109/SRDS.2006.38

Fahmy, S., B. Ravindran and E.D. Jensen, 2008. On Collaborative scheduling of distributable real time threads in dynamic, networked embedded systems. Proceeding of 11th IEEE Symposium on Object Oriented Real Time Distributed Computing, May 5-7, IEEE Computer Society, Washington DC., USA., pp: 485-491. http://www.computer.org/portal/web/csdl/doi/10.1109/ISORC.2008.11

Jensen, E.D., C.D. Locke and H. Tokuda, 1985. A time driven scheduling model for real time operating systems. Proceeding of the IEEE Symposium on Real-Time System, Dec. 1985, IEEE Xplore Press, USA., pp: 112-122. http://www.real-time.org/docs/rtss85.pdf

Li, P., H. Wu, B. Ravindran and E.D. Jensen, 2006. A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints. IEEE Trans. Comput., 55: 454-469. DOI: 10.1109/TC.2006.47

Ravindran, B., E.D. Jensen and P. Li, 2005. On recent advances in time/utility function real-time scheduling and resource management. Proceeding of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, May 18-20, IEEE Xplore Press, USA., pp: 55-60. DOI: 10.1109/ISORC.2005.39

Sahoo, B. and A.A. Ekka, 2007. Backward fault recovery in real time distributed systems of periodic tasks with timing and precedence constraint. Proceedings of the International Conference on Emerging Trends in High Performance Architecture, Algorithms and Computing, July 11-13, Chennai India, pp: 124-130. http://dspace.nitrkl.ac.in/dspace/bitstream/2080/447/1/hipaac.pdf

Wu, H., B. Ravindran, E.D. Jensen and P. Li, 2004. CPU scheduling for statistically-assured real-time performance and improved energy efficiency. Proceeding of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Sept. 8-10, IEEE Xplore Press, USA., pp: 110-115. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1360490