International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:9, 2016

Simulation of Utility Accrual Scheduling and Recovery Algorithm in Multiprocessor Environment

A. Idawaty, O. Mohamed, A. Z. Zuriati

***Abstract*—**This paper presents the development of an event based Discrete Event Simulation (DES) for a recovery algorithm known Backward Recovery Global Preemptive Utility Accrual Scheduling (BR_GPUAS). This algorithm implements the Backward Recovery (BR) mechanism as a fault recovery solution under the existing Time/Utility Function/ Utility Accrual (TUF/UA) scheduling domain for multiprocessor environment. The BR mechanism attempts to take the faulty tasks back to its initial safe state and then proceeds to re-execute the affected section of the faulty tasks to enable recovery. Considering that faults may occur in the components of any system; a fault tolerance system that can nullify the erroneous effect is necessary to be developed. Current TUF/UA scheduling algorithm uses the abortion recovery mechanism and it simply aborts the erroneous task as their fault recovery solution. None of the existing algorithm in TUF/UA scheduling domain in multiprocessor scheduling environment have considered the transient fault and implement the BR mechanism as a fault recovery mechanism to nullify the erroneous effect and solve the recovery problem in this domain. The developed BR_GPUAS simulator has derived the set of parameter, events and performance metrics according to a detailed analysis of the base model. Simulation results revealed that BR_GPUAS algorithm can saved almost 20-30% of the accumulated utilities making it reliable and efficient for the real-time application in the multiprocessor scheduling environment.

***Keywords*—**Time Utility Function/ Utility Accrual (TUF/UA) scheduling, Real-time system (RTS), Backward Recovery, Multiprocessor, Discrete Event Simulation (DES).

I. Introduction

ONE of the main approaches for fault recovery is the BR mechanism [1]-[5]. The BR mechanism attempts to take the system back to its initial safe state and then proceeds to re-execute the affected section of the faulty task. Fig. 1 elaborates the detail of the BR mechanism which is based on returning the execution of the task to a predetermined state. The task can return to the initial reset state or to one of a set of possible states after the error detection occurred. The BR mechanism is widely integrated in real time scheduling algorithms such as EDF and RM [4]-[8]. In order to execute the faulty section once again, the task restores its initial state and re-executes the partial affected section from its initial state before the error occurs to enable recovery.

A. Idawaty, O. Mohamed, and A. Z. Zuriati is with the Department of Computer Science and Information Technology, University Putra Malaysia, UPM, Serdang, 43400 Selangor, Malaysia (e-mail: idawaty@ upm.edu.my, mothman@ upm.edu.my, zuriati@ upm.edu.my).

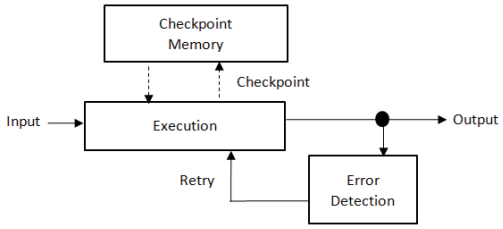

Fig. 1 Logical representation of BR mechanism [9]

This paper considers the BR mechanism to be integrated into the developed TUF/UA scheduling algorithms for multiprocessor environments. The scheduling objectives are to maximize the total accrued utility to the system in the stipulated erroneous environment.

II. Literature Review

Fault recovery phase is an attempt to substitute the erroneous system state with one which is error free. Two main approaches for fault recovery are the Backward Recovery (BR) and the Forward Recovery (FR) mechanism [1]-[15]. The BR mechanism attempts to take the system state to a safe previous state and then proceeds to re-execute the affected task. This paper considered the BR mechanism implemented in TUF/UA scheduling domain to enable recovery when transient fault occurs in the system.

A. Backward Recovery Mechanism

There are two types of the BR mechanism which are the static and dynamic [9]. The static BR mechanism has been used in real time scheduling domain in [1]-[3], [9]. Referring to Fig. 1, in the static approach of the BR mechanism, the task is roll back to its initial state before the re-execution procedure is executed.

A dynamic BR mechanism on the other hand, dynamically creates checkpoints that are the snapshots of the state at various points during the execution [1]-[3]. The dynamic approach of the BR mechanism is also known as checkpointing. It requires the task state to be monitored at regular checkpoint interval so that when the erroneous request is detected, the re-execution procedure is taken from the nearest checkpoint interval. Thus, a set of particular checkpoint interval must be identified in the task model. This research focuses on the static approach of the BR mechanism to adapt with the task and model being used in the TUF/UA scheduling domain that excludes the regular checkpoint interval in its task model.

The second approach of fault recovery in RTS is the

Forward Recovery (FR) mechanism [10]-[15]. FR is based on the use of error correcting techniques and Abortion Recovery (AR) mechanism that aims to nullify the effects of the faulty execution and perform some re-initialization of the resources, rather than performing the whole computation once again. In the AR mechanism, after an error is detected in a task, the system continues from an erroneous task state by aborting the faulty task and operates in degraded mode.

It is observed that the existing TUF/UA scheduling algorithm uses the Abortion Recovery (AR) mechanism that simply abort the erroneous task as their fault recovery solution [10]-[15].The erroneous task is aborted due to the fail stop failures of the node and network failures to nullify the error when the task encounters an error during its execution. None of the existing algorithm in multiprocessor environment considered the transient fault and implements the BR mechanism as recovery solution. The next section elaborates the fault recovery mechanism used in the TUF/UA scheduling domain.

B. TUF/UA Scheduling Domain

A TUF of a task specifies the quantified value of utility gained by the system after the completion of a task [16], [17]. With reference to Fig. 2, in the event of the task being computed at time A, which denotes the range between the start of execution and the stipulated deadline, the system gains a positive utility. However, if the task is completed at time B, which causes failure of deadline compliance requirement, the system acquires zero utility [19]. A TUF integrates the importance and urgency of a task. Mapping this on Fig. 2, urgency is denoted as the completion time on X-axis whilst importance is measured by the utility on the Y-axis.

When the task characteristic is expressed using TUFs, the value of utility for each executed task is accumulated and the total attained utility are measured. The scheduling optimization goal is to maximize the sum of the tasks' accrued utilities which is known as Utility Accrual (UA) [16].

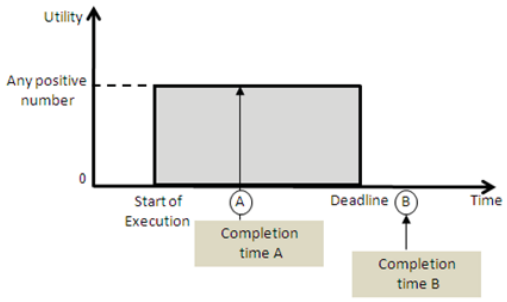

Fig. 2 Time/Utility Function

Scheduling algorithms that consider the UA as a criterion are known as TUF/UA scheduling algorithms [19]. The TUF/UA scheduling algorithms maximize the total accrued utilities when capacity surpluses load. However, in overloaded condition, the TUF/UA scheduling algorithms execute tasks that are more important from which greater utility can be accrued than those which are more urgent. Tasks that are less important with lower utility are aborted and cause some graceful performance degradation during overloaded conditions.

C. Fault Recovery Approach in TUF/UA Scheduling Domain

This paper focuses on the fault recovery mechanism in the TUF/UA scheduling domain. A detail review on the TUF/UA algorithms in [10]-[15] shows that the fault recovery mechanism is integrated into the multiprocessor scheduling environment. The first column of Table I depicts the existing algorithms that consider fault in its scheduling decision. These include the Aborted-assured Utility Accrual (AUA), Consensus Utility Accrual (CUA), and Real Time Gossip (RTG) algorithms. These algorithms consider the fault tolerance in a distributed wireless embedded system environment. The failure models considered in these algorithms are the fail-stop node failures and communication failures. Nodes are subject to crash failures that causes lose in its state memory and the process halts. A node will be considered to have failed when it is assumed to be permanently disconnected from the rest of the network. A communication failure occurs when any message does not reach its desired destination due to a physical break in the communication medium, communication buffer overflow, parity error, etc.

TABLE I
FAULT TOLERANCE IN TUF/UA SCHEDULING DOMAIN

algorithm	task model (distributable)		fault model		recovery mechanism	
	collaborative	independent	node	network	AR	BR
AUA	-	✓	✓	-	✓	-
RTG	-	✓	✓	✓	✓	-
CUA	✓	-	✓	-	✓	-

It is observed that the task model utilizes by these algorithms is distributable type of tasks. A distributable task has a globally unique number identity that requests for local resources (within its node) and remote resources at other nodes in a distributed environment. The distributable tasks can be scheduled in two paradigms that are the node independence scheduling and collaborative scheduling [12]. In the independent scheduling, each node in a distributed system schedules the tasks that it hosts without any communication with other nodes [14]. A task makes a request to a remote resource and propagates its scheduling parameters to the destination node. The destination node then uses these propagated parameters to perform its own local scheduling [12]. The AUA algorithm considers tasks that are scheduled at nodes using the propagated task scheduling parameters without any interaction with other nodes [14]. The RTG solves the discovery of failure nodes by using a gossip protocol for propagation of task scheduling parameters to the neighbor nodes via multicasting in a wireless mobile ad-hoc (MANET) environment [10].

In collaborative scheduling, distributed nodes collaborate to construct a system-wide task schedules. The CUA algorithm

considers collaborative scheduling where distributed nodes explicitly cooperate to construct a schedule that anticipates node crash failure [13]. The similarities of these algorithms are elaborated in Table I. The fail-stop failures models are used which include the arbitrary task failures, node failures, and message loss (due to network failure). These TUF/UA scheduling algorithms have developed with the AR mechanism as their fault recovery mechanism to accommodate the fail-stop failures; none have directly addressed the BR mechanism as their recovery solution for transient fault.

III. Developed Recovery Algorithm

For the purpose of implementing the BR mechanism with the existing multiprocessor framework, two events are designed as follows:

- Fault Model
- Fault Recovery Model

A. Fault Model

The fault model of a system is a set of assumptions on the type of faults that are possible to occur in the system. Once the system fault model is defined, the approach for implementing the fault recovery mechanisms in the system is addressed. In this paper, the fault model and the BR recovery mechanism developed in the Responsive Algorithm (RA) algorithm [3] is used to be implemented in the TUF/UA scheduling environment. The fault model considers the following assumptions:

1. A fault detection mechanism exists to detect transient faults.
2. A task may encounters an error while holding a resource such as request execution failure or any external triggers that occurs to the respective resource.
3. The transient faults that occur in a request can be effectively recovered by re-execution of the affected request.
4. The exponential distribution is used to characterize the arrival of aperiodic faults. The commonly used distributions in RTS are the exponential distribution, Weibull distribution and uniform distribution [2], [9].
5. After a fault is detected, a roll back time i.e., *RollBackTime* is taken by the faulty task to its initial state before a recovery procedure is initiated as shown in Fig. 3. The computation requirement i.e., the *RollBackTime* of recovery request is known at the instant of its arrival.
6. The re-execution procedure can be preempted by a larger PUD task.

B. Fault Recovery Model

The BR mechanism is implemented to provide solution for fault recovery in the TUF/UA scheduling domain. The BR mechanism is adopted from the existing RA algorithm [3] for uniprocessor environment. The RA algorithm provides a responsiveness level for handling a recovery request and provides a set of priorities at which the recovery request should be re-executed. The scheduling objective is to maximize the total accrued utility to the system in transient fault environment.

As any system has finite resources, the ability to schedule recovery requests while meeting TUF requirements is limited. An overloaded condition may arise if the recovery requests have to be re-executed together with the error free requests. Under such conditions, the decision of whether or not a recovery request can be scheduled will depend on five consecutive tests as stated below:

1.Feasibility Test

Feasibility test is run to ensure that only the feasible recovery request is re-executed in the system. This is done by calculating the slack time of the recovery request's task. The slack time is defined as the remaining execution time before the task deadline if the recovery request is to be scheduled in the system. Referring to Fig. 4, the slack time is denoted in the *SlackTime* parameter. The time taken to re-execute the recovery request is equal to its *HoldTime*. If the recovery request is to be scheduled instantly at current time (i.e., sclock), the *SlackTime* is calculated as *TerminateTime* - (*sclock*+*HoldTime*). The current time is represented as *sclock*.

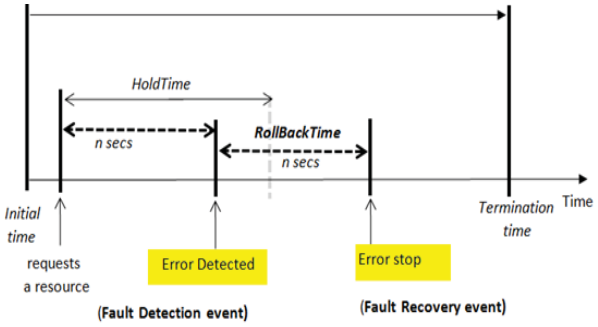

Fig. 3 Fault Model

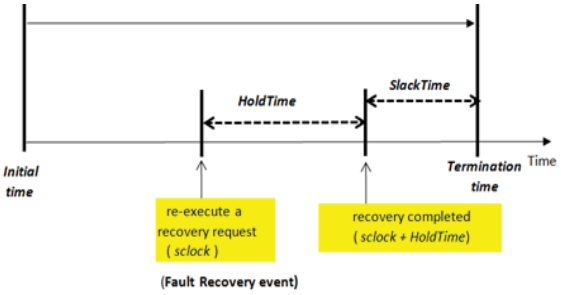

Fig. 4 Responsiveness Test

In the example shown in Fig. 4, the *SlackTime* is a positive value that indicates the recovery request is feasible to be scheduled.

2.Criticality Test

After the recovery request has successfully tested through the feasibility test, the criticality test is initiated. This test checks whether the re-execution of the recovery request possess the highest Potential Utility Density (PUD) among the other competitor tasks. PUD forms a crucial standard metric

for operating advancement proposed by [16]. PUD of a task measures the amount of utility that can be accrued per unit time by execution of the respective task. It essentially measures the Return on Investment (RoI) for executing the task at current simulation time. There are two possibilities for the recovery request as follows:

i. The recovery request possesses the highest PUD among the other competitor tasks. This indicates that the recovery request is critical and need to be scheduled for re-execution.
ii. There exists the other request from the other task that possess the highest PUD. This shows that the recovery request is less critical and eligible to be scheduled later and being put in a queue or to be aborted. The decision of whether to abort or to insert in a queue is depending on the subsequent responsiveness test. The taxonomy of the procedures taken for a recovery request in a task is depicted in Fig. 5.

3.Responsiveness Test

After a recovery request is identified to be less critical, the responsiveness test is initiated. This test is executed only when recovery request is less critical. This is shown in Fig. 5. The purpose of executing this phase is to discover the impact to the system towards the existence of a recovery request. There are two levels of responsiveness test results as follows:

i. Non-Intrusive Recovery Level

This level indicates that the system is possible to schedule the recovery request without violation any other competitor tasks deadlines. Since the recovery request is less critical and non-intrusive to the processor, it will be inserted into a queue and not necessarily scheduled instantly.

ii. Intrusive Recovery Level

This level indicates that the system becoming highly loaded to schedule the recovery request thus may violate the competitor tasks deadlines. Referring to Fig. 5, for this level, the migration characteristics of the recovery request is checked. In multiprocessor environment, if the recovery request can migrate, the responsiveness test is run in other processor in the system. For the uniprocessor scheduling environment, the recovery request will be aborted due to the intrusiveness.

The global scheduling allows tasks to migrate from one processor to other processor in a scheduling event. Taking advantages on this characteristic, the BR mechanism is further explored to the other processor if the local processor is highly overloaded to accommodate the recovery request. Only after the local processor is discovered as intrusive, the scheduler in global scheduling subsequently will search for an idle and identical resource at the other processors. Under such conditions, the decision to schedule a recovery request at the other processor will depend on two consecutive tests as stated below:

4.Availability Test

The availability test is executed to search for the idle and identical resources at the other processors. If all resources are busy, the recovery request is put into the sufficiency test to decide whether the recovery request is necessary to be migrated to the other processor.

5.Sufficiency Test

This test is executed to ensure that if the recovery request is to be migrated to another processor, the load in the respective processor is sufficient to re-execute the recovery request without violating any competitor tasks deadline. If all the resources are insufficient, then the recovery request is not migrated and will be locally queued. If a recovery request went through these tests, it is confirmed that the recovery request is not suitable to be scheduled immediately and must be inserted into the queue to be scheduled later. If the recovery request is to be migrated to the queue located at another processor, the workload of the queue in the respective processor is sufficient to re-execute the recovery request without violating the task deadline. If all resources are insufficient, then the recovery request is not migrated and is inserted into the queue located at the local processor. The abovementioned procedures taken in this phase are illustrated in Fig. 5.

IV. Simulation Model

We developed a Discrete Event Simulator (DES) to verify the performance of BR_GPUAS algorithm. 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 [15]-[17]. Therefore, in order to precisely model the fault recovery algorithms, DES written in C language is the best method to achieve this objective. Fig. 6 shows the entities involve in the developed simulation framework. It consists of a stream of 1000 tasks, a scheduler and multiple processors. The simulation parameters are given in Table II.

The *M/M/C* queuing model is used to indicate a multiprocessor system with *C* processor that have unlimited queue capacity and an infinite population of potential task arrivals. Generally, the inter arrival times denoted by λ and the service times per server denoted by μ are exponentially distributed [18]. To reflect the *M/M/C* with the multiprocessor scheduling model, *C* is the number of processors in the system which is also known from the *MAX_CPU* parameter as shown in Table III. The inter arrival time, denoted as λ is defined in the unit of *tasks/secs* measures the number of tasks that arrived into the system in one second. The service rate per processor denoted as μ measures the number of tasks that is being processed by each processor within one second.

For multiprocessor, the maximum service rate for all processors is equal to $C\mu$. Note that inter service time for all processors $C\mu$ is calculated according to the number of processors *C*. From the general estimation of the system behavior for *M/M/C* queuing model, the system is considered to be stable when the arrival rate λ is less than the maximum service rate $C\mu$ i.e., $\lambda < C\mu$. Since the same value of *C_AVG* is used in the multiprocessor environment, each service rate μ

is calculated 1/ *C_AVG* that is equal to 2 *tasks/secs*. Hence, the maximum service rate for two, four, and eight processors is 4, 8, and 16 *tasks/secs* respectively. This is shown in the second column of Table III.

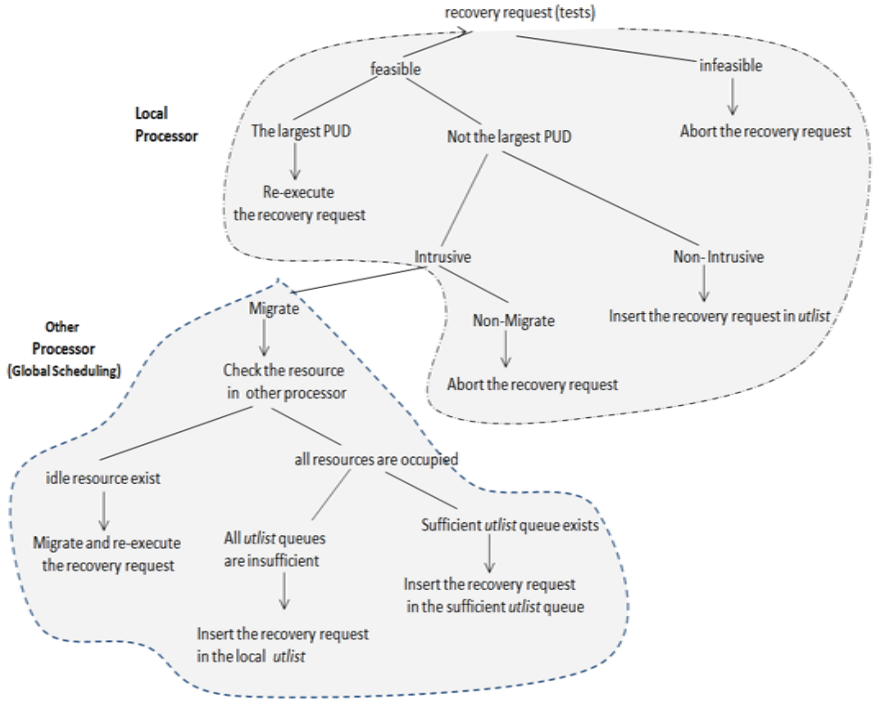

Fig. 5 Taxonomy of the developed BR mechanism

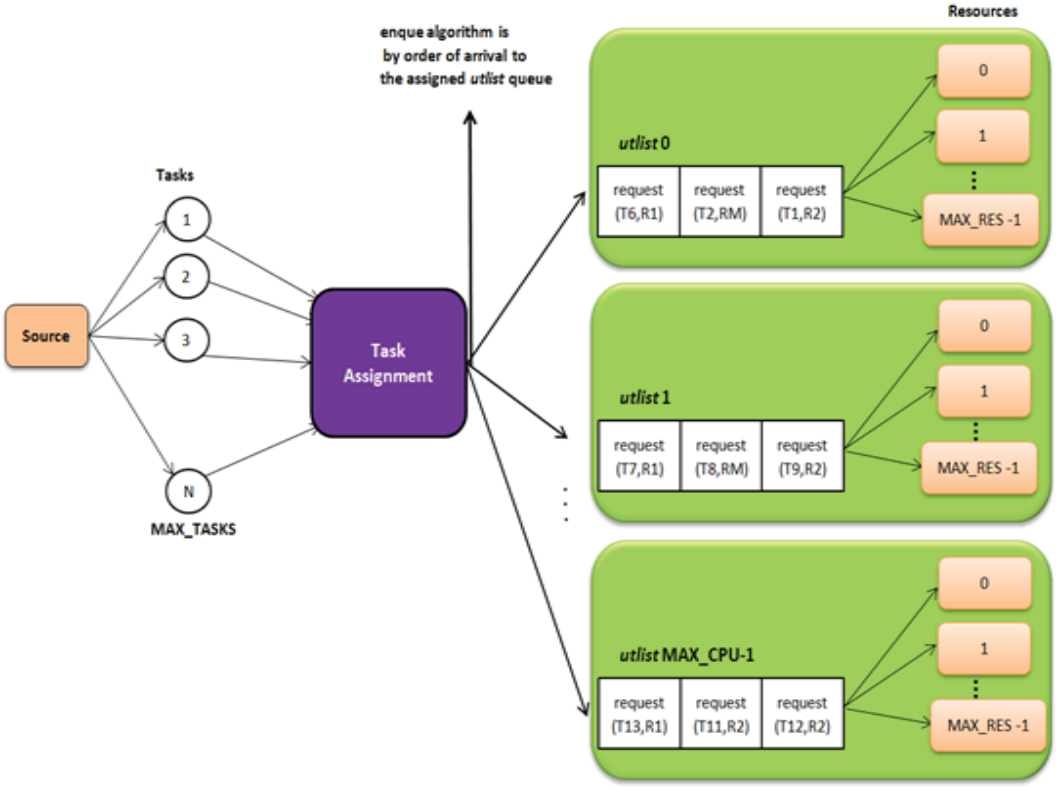

Fig. 6 Simulation Framework [17]

TABLE II
SIMULATION PARAMETERS

Parameter	Value	Description
MAX_TASK	1000	Maximum number of tasks in the system
MAX_CPU	1,2, 4 and 8	Maximum number of CPU
load	1-10	Range of average load in the system
Max_AU	Normal(10,10)	The maximum utility of a task. It follows normal distribution with mean value of each task is 10 and the variance is 10.
C_AVG	Exponential (0.50)	Tasks inter arrival time follows exponential distribution with mean value of 0.50 secs.

TABLE III
PARAMETER ESTIMATION IN M/M/C QUEUING MODEL

Number of CPU	Parameters	
	$C\mu$ (*tasks/secs*)	$\rho = \lambda/\mu < C$
1	2	$\rho < 1$
2	4	$\rho < 2$
4	8	$\rho < 4$
8	16	$\rho < 8$

From the literature, the system is considered to be stable when the arrival rate λ is less than the maximum service rate $C\mu$ i.e., $\lambda < C\mu$ [18]. Equivalently, the offered load $\rho = \lambda/\mu < C$. Thus, the offered load ρ must be less than the number of processors C. Hence, the general estimation of the simulation model, the system is considered to be under load of when the offered load $\rho < C$. In the simulation model, the value of ρ is stored in the *load* parameter and the value of C is depicted in the *MAX_CPU*. Therefore, the rough estimation for the stable behavior of the multiprocessor system is for the value of *load* < *MAX_CPU*. Hence, in all the experiment the range value of *load* is selected as $1 \leq load \leq 10$. For every number of *MAX_CPU*, the system is started to be overloaded starting when *load* = MAX_*CPU*. Referring to Table III, for dual core processors, the system is estimated to be overloaded when *load* = 2. In the quad core processor environment, the system is expected to be overloaded when *load* = 4. For eight core platform, the system is considered as overloaded when *load* = 8. Note that these loads are the approximation value that may be considered as a rough guide to the behavior of the system. Practically, the results observed from the simulation are used to measure the performances of the system.

The Accrued Utility Ratio (AUR) is defined as the ratio of accrued aggregate utility to the maximum possibly attained utility. Each task *i* has its maximum value of utility which is denoted as *MaxAU(i)*. After a task *i* have completed its execution, it will yield a value denoted as *Utility (i)*. These values are then accumulated for all tasks i.e., *MAX_TASKS*.

V. RESULTS

This section presents the results for step TUF task set. Fig. 7 depicts the AUR results under an increasing load for 10% error rate imposed in the system. A lower utility accrued by all the scheduling algorithms as the load increases. As the load increases, the number of executed tasks plus the recovery tasks to be scheduled becomes highly overloaded in the system causing the lower PUD tasks to be aborted due to the limited resources consumption. A larger amount of aborted tasks produces more zero utility tasks and ultimately a low utility accrued to the system as the load increases.

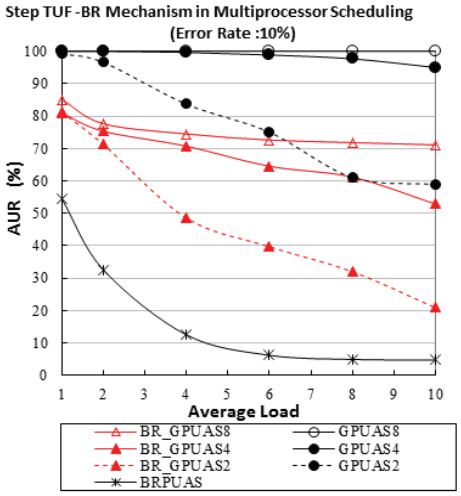

Fig. 7 Results for BR_GPUAS

The overall performances of the BR mechanism in the multiprocessor scheduling environment have achieved better performances by producing a higher utility accumulated to the system as compared to the uniprocessor scheduling (i.e., BRPUAS) for the entire load range. This is because a higher number of available resources are equipped in the dual, quad and eight core processors as compared to BRPUAS that runs in a single processor. A larger number of processors provide a larger number of resources to schedule the intact tasks together with the recovery tasks in the system. Moreover, the BR_GPUAS algorithm can reduce the number of abortion that possibly occurs to the recovery task by migrating it to the other available processors.

In dual core platform, the system is estimated to be overloaded at load equals to 2. Referring to Fig. 7, at this load, in error free environment GPUAS can maximally achieved 96.58% of the accumulated utilities. In erroneous environment with 10% error rate imposed in the system, BR_GPUAS2 significantly achieved 71.25% that preserved 25.33% of the accumulated utilities. At the highest load, BR_GPUAS2 can save 20.86% of the accumulated utilities.

In quad core platform, the system is estimated to be overloaded when load is equal to 4. From Fig. 7, when the load is equals to 4, the BR_GPUAS4 acquired 70.71% and saved 28.92% of the accumulated utilities as GPUAS4 can maximally achieved 99.63% of utilities in the error free environment. The BR_GPUAS2 moderately achieved 48.52% of utilities. The BR_GPUAS4 can save 22.19% as compared to the BR_GPUAS2 algorithm. This is because BR_GPUAS4 can reduce the number of abortion that possibly occurs to the recovery task in a processor by migrating it to the other available processors. As the number of processor increases,

more resources are available to accommodate the migrated recovery tasks. At the highest load i.e., *load = 10*, BR_GPUAS4 significantly achieved 52.84% of the accumulated utilities while the BR_GPUAS2 achieved 20.86% of the accumulated utilities. Therefore, BR_GPUAS4 can save 31.98% of the accumulated utilities as the load increased to the highest load in the system.

In eight core platforms, the highest utility accumulated by the system can be achieved by the BR_GPUAS8 for the entire load range. The BR_GPUAS8 algorithm has accumulated 85.09% of AUR when load is equal to 1. At the highest load at 10, the BR_GPUAS8 algorithm acquired 71.12% of the accumulated utilities. In the BR_GPUAS8 algorithm, the BR mechanism is executed in the global scheduling which executed the migrate tasks with highly available resources in eight core processors environment. The recovery tasks are capable for migration due to the insufficient load to process the recovery tasks in the local processor. As the number of processor increases, more resources are available to accommodate the migrated recovery tasks. This is why the highest utility accumulated by B_GPUAS8 for the entire load range.

VI. Conclusion

The performances of the BR mechanism implemented in multiprocessor environment known as BR_GPUAS are measured using DES. Simulation results revealed that BR_GPUAS algorithm is proven to save almost 20-30% of the executed tasks when maximum error rate 10% is imposed in the system thus accrued a higher utility making it reliable and efficient for the real-time application.

Acknowledgment

This research was funded by the Ministry of Higher Education Malaysia and Universiti Putra Malaysia under Fundamental Research Grant FRGS 08-01-15-1722FR.

References

[1] S. Punnekat, A. Burns, and R. Davis, "Analysis of checkpointing for real time systems," *Real Time System*, vol. 20, no. 1, pp. 83–102, 2001.
[2] P. Alvarez, "Scheduling fault recovery operations in real time systems," *Journal of Computer System*, vol. 6, no. 1, pp. 51–61, 2002.
[3] P. Alvarez and D. Mosse, "A responsiveness for scheduling fault recovery in real time systems," in *Proc. 5th IEEE Real Time Technology and Application Symposium*, Vancouver, 1999, pp. 4–13.
[4] R. Pathan, *Scheduling Algorithms for Fault Tolerant Real Time Systems.* PhD Thesis, Chalmers University of Technology, 2010.
[5] G.A. Lima, *Fault Tolerance in Fixed Priority Hard Real Time Systems.* PhD Thesis, University of York, 2003.
[6] B. Sahoo and A. Ekka, "Backward fault recovery in real time distributed systems of periodic tasks with timing and precedence constraint," in *Proc. of International Conference on Emerging Trends in High Performance Architecture, Algorithms and Computing (HiPAAC'07)*, Chennai, 2007, pp. 124–130.
[7] F. Zhang, and A. Burns, "Schedulability analysis for real time systems with EDF scheduling," *IEEE Trans on Computers*, vol. 58, no. 9, pp. 1250–1258, 2009.
[8] H. Aydin, R. Mosse, and P. Alvarez, "Optimal reward-based scheduling for periodic real time tasks," *IEEE Journal of Computer*, vol. 50, no. 2, pp. 111–130, 2001.
[9] S. Punnekat, *Schedulability Analysis for Fault Tolerant Real Time Systems.* PhD Thesis, University of York, 1997.
[10] K. Han, B. Ravindran, and E. Jensen, "RTG-L: Dependably scheduling real time distributed threads in large scale, unreliable networks," in *Proc. 13th IEEE Pacific Rim International Symposium on Dependable Computing (PDRC'07)*, Melbourne, 2007, pp. 314–321.
[11] H. Cho, B. Ravindran, and E. Jensen, "Garbage collector scheduling in dynamic, multiprocessor real time system," *IEEE Transactions on Parallel and Distributed System*, vol. 20, no. 6, pp. 845–856, 2009.
[12] S. Fahmy, B. Ravindran, and E. Jensen, "On collaborative scheduling of distributable real time threads in dynamic, networked embedded systems," in *Proc. 11th IEEE Symposium on Object Oriented Real Time Distributed Computing(ISORC'08)*, Orlando, 2008, pp. 485–491.
[13] B. Ravindran, B. Anderson and E. Jensen, "On distributed real-time scheduling in networked embedded systems in the presence of crash failures," in *Lecture Notes in Computer Science*, 4761/2007, Springer Berlin/Heidelberg, 2007, pp. 67–81.
[14] E. Curley, *Recovering From Distributable Thread Failures with Assured Timeliness in Real Time Distributed System.* Master Thesis, Virginia Polytechnic Institute and State University, 2007.
[15] E. Jensen, E. Locke, and H. Tokuda, "A time driven scheduling model for real time system," in *Proc. 6th IEEE Real Time Symposium*, San Diego, California, 1985, pp. 112–212.
[16] P. Li, Utility Accrual Real Time Scheduling: Models and Algorithms. PhD Thesis, Virginia Polytechnic Institute and State University, 2004.
[17] A. Idawaty, O. Mohamed, and A.Z. Zuriati, "Enhanced preemptive global utility accrual real time scheduling algorithms in multicore environment," *Journal of Computer Sciences*, vol. 11, no. 12, pp. 1009–1107, 2015.
[18] J. Banks, J. Carson, B. Nelson, and D. Nicol, *Discrete Event System Simulation.* 3rd edition, Prentice Hall, 2000.
[19] A. Idawaty, S. Shamala, O. Mohamed, and A.Z. Zuriati, "A Discrete Event Simulation Framework for Utility Accrual Scheduling Algorithm in Uniprocessor Environment," *Journal of Computer Sciences*, vol. 7, no. 8, pp. 1133–1140, 2011.