Journal of Computer Science 7 (8): 1133-1140, 2011
ISSN 1549-3636

A Discrete Event Simulation Framework for Utility Accrual Scheduling Algorithm in Uniprocessor Environment

Idawaty Ahmad, Shamala Subramaniam, Mohamed Othman and Zuriati Zulkarnain
[1]Department of Communication Technology and Network,
Faculty of Computer Science and Information Technology,
University Putra Malaysia, 43400 UPM,
Serdang, Selangor DE, Malaysia

Abstract: Problem statement: The heterogeneity in the choice of simulation platforms for real time scheduling stands behind the difficulty of developing a common simulation environment. A Discrete Event Simulation (DES) for a real time scheduling domain encompassing event definition, time advancing mechanism and scheduler has yet to be developed. **Approach:** The study focused on the proposed and the development of an event based discrete event simulator for the existing General Utility Scheduling (GUS) to facilitate the reuse of the algorithm under a common simulation environment. GUS is one of the existing TUF/UA scheduling algorithms that consider the Time/Utility Function (TUF) of the executed tasks in its scheduling decision. The scheduling optimality criteria are based on maximizing accrued utility accumulated from execution of all tasks in the system. These criteria are named as Utility Accrual (UA). The TUF/ UA scheduling algorithms are design for adaptive real time system environment. The developed GUS simulator has derived the set of parameter, events, performance metrics and other unique TUF/UA scheduling element according to a detailed analysis of the base model. **Results:** The Accrued Utility Ratio (AUR) is investigated and compared to the benchmark of the modeled domain. Successful deployment of the GUS simulator was proven by the generated results. **Conclusion:** Extensive performance analysis of GUS simulator can be deployed using the developed simulator with low computational overhead. Further enhancements were to extend the developed GUS simulator with detail performance metrics together with a fault tolerance mechanism to support a reliable real time application domain.

Key words: Real Time Scheduling, Discrete Event Simulation (DES), Time/Utility Function (TUF), General Purpose Language (GPL)

INTRODUCTION

Real-time scheduling is fundamentally concerned with satisfying 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.

One of the scheduling paradigms in adaptive real time system environment is known as Time/Utility Function (TUF) (Idawaty *et al.*, 2009; Wu *et al.*, 2004; Jensen *et al.*, 1985). A TUF of a task specifies the quantified value of utility gained by the system after the completion of a task 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.

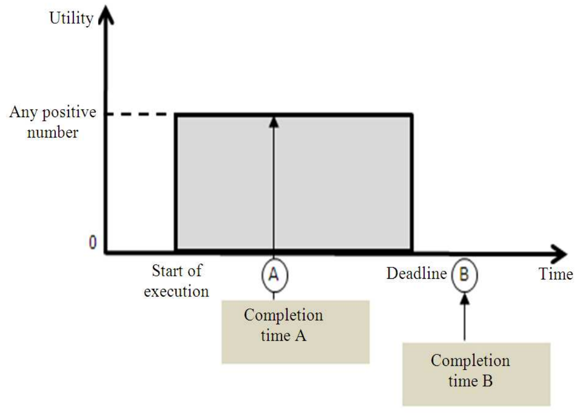


Fig. 1: The step TUF

Corresponding author: Idawaty Ahmad, Department of Communication Technology and Network, idawaty@fsktm.upm.edu.my

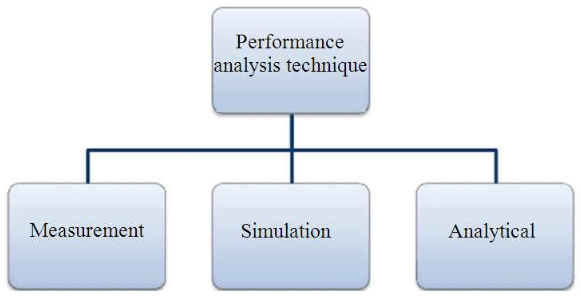


Fig. 2: Performance Analysis Techniques (Law, 2003)

With reference to Fig. 1, 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. When the tasks characteristics are 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) (Wu *et al*., 2004; Jensen *et al*., 1985). The scheduling algorithms that consider the UA as a criterion are known as TUF/UA scheduling algorithms.

GUS is a uniprocessor TUF/UA scheduling algorithm that manages the independence tasks and tasks that have dependencies with other tasks (Li *et al*., 2006). The dependencies are due to the sharing of resources via the single unit of resource request model. In enhancing and developing the GUS algorithm, performance analysis and its respective tools are evident. The performances are measured by using analytical, simulation and measurement methods as shown in Fig. 2 (Law, 2003). Analytical model uses mathematical notation and simulation model uses computer program to imitate the behavior of a system.

Problem statement: The benchmark model of GUS was developed using OMNET++ that is one of the available discrete event simulation tools (Li *et al*., 2006). Table 1 depicts the existing TUF/UA scheduling algorithms and its simulation tools. It is observed that the SIMSCRIPT, OMNET++ and ns2 tools are used to investigate the performance of these algorithms. Though there exists the simulation tools, there does not exist a detailed description and a developed General Purpose Language (GPL) DES for the TUF/UA scheduling domain specifically the GUS algorithm. The lack of uniformity in the choice of simulation platforms is a clear limitation for investigating the performances of the TUF/UA scheduling algorithms.

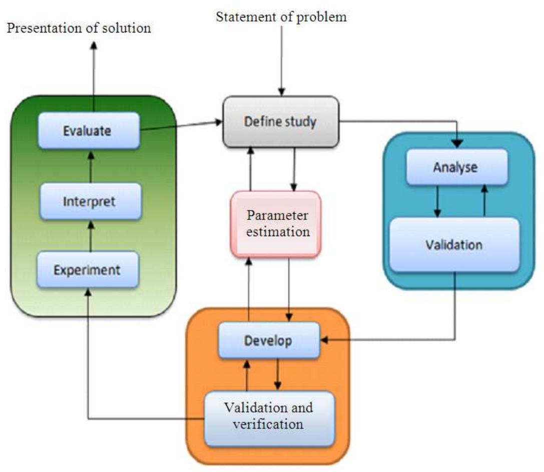


Fig. 3: The Simulation Study Life Cycle ((Law, 2003)

Table 1: The simulation tools used in the TUF/UA scheduling domain

Existing algorithms	Simulation tools	Year
LBESA	SIMSCRIPT	1985
DASA	SIMSCIRPT	1996
GUS	OMNET++	2004
MSA	OMNET++	2006
GCMUA	ns2	2009
Gamma	ns2	2010

This study presents the development of a DES for one of the TUF/UA scheduling algorithms i.e., GUS and a comprehensive model development. The model can be adopted and customized for further analysis with ease.

Objective: The GUS simulator is built from the scratch to enable customization requirements of any research and to provide the freedom to understand, configure TUF modules, draw desired scheduling environment and plot the necessary performance graphs. In order to evaluate and validate the performance of the designed simulator, a simulation model for the TUF/UA scheduling environment is deployed.

MATERIALS AND METHODS

Approach: The steps taken for developing the GUS simulator is shown in Fig. 3 specifying various phase to be followed (Karatza,2000; Law, 2003).

Study definition phase: The first stage in the simulation life cycle is the study definition phase. In this phase, the problem formulations and the objectives of the study are identified. Concurrently, the input and output requirements of the developed model are also identified.

Analysis phase: In the analysis phase, the main components such as entities, queues, events or resources are identified in the simulation model.

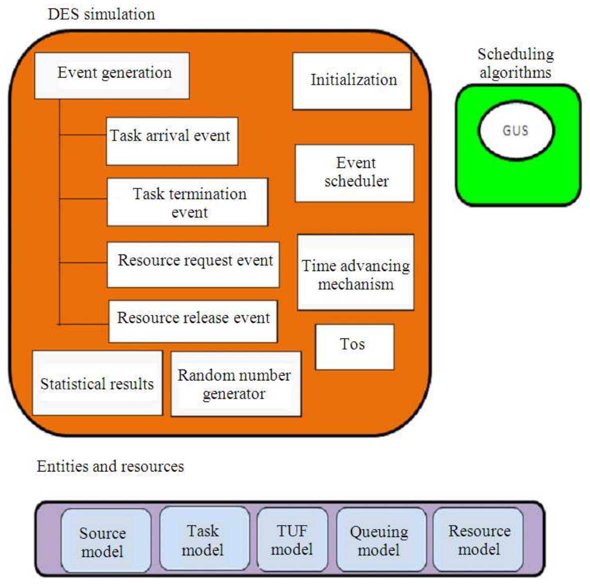


Fig. 4: Simulation framework

Parameter estimation phase: To obtain a convincing model, the values of parameters that quantify the effect in the model must indeed represent reality. Thus, parameter estimation must be set with precision and similar to the real system. One of the methods to realize this in this research is estimating parameters by absorbing the benchmark model which is the GUS algorithm (Li *et al*., 2006).

Model development phase: This phase consists of the development of the conceptual model as a computer program. This also constitutes the verification and validation steps as shown in Fig. 3. Verification is the process used to determine the model correctness. The validation phase is the process of determining if the simulation model is an accurate representation of the system and performs its stipulated intention. The most definitive test of a simulation model's validity is establishing that its output data closely resemble the output data that would be observed from the benchmark model i.e., GUS (Li *et al*., 2006). Validation of the developed simulator is given in the results section.

Experimentation and result analysis phase: The simulation model is executed in a series of parametric simulation runs which are performed to satisfy the aims of the simulation study. Based on the result analysis, Various conclusions are drawn. Conclusions of this study are given in the conclusion section.

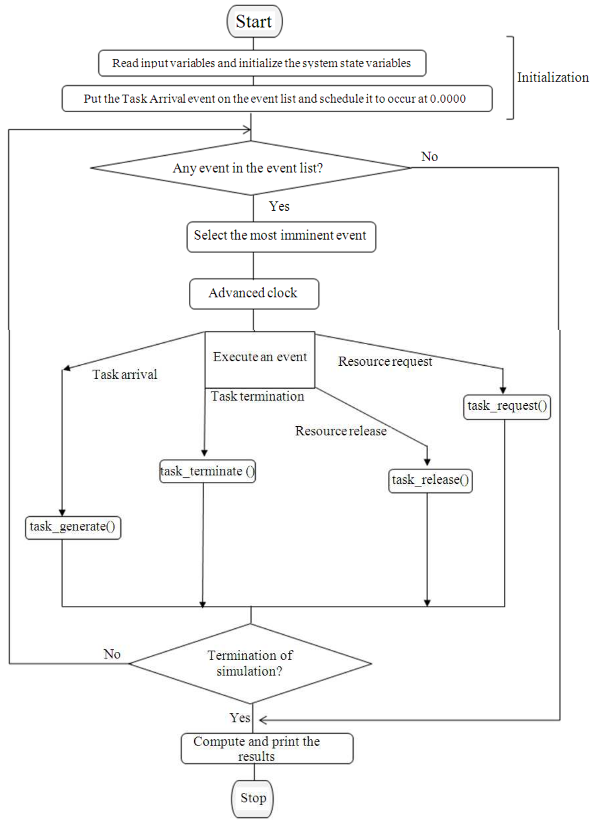

Start
Read input variables and initialize the system state variables
Put the Task Arrival event on the event list and schedule it to occur at 0.0000
Initialization
Any event in the event list?
No
Yes
Select the most imminent event
Advanced clock
Execute an event
Task arrival
Resource request
Task termination
Resource release
task_request()
task_terminate ()
task_release()
task_generate()
No
Termination of simulation?
Yes
Compute and print the results
Stop

Fig. 5: Flowchart of the simulation program

Discrete simulation framework: A discrete event simulation framework is developed to verify the performance of the GUS scheduling algorithm. In order to precisely remodel and further enhance the GUS algorithm, DES written in C language in Visual C++ environment is the best method to achieve this objective.

Figure 4 shows the developed GUS simulator framework. It consists of the four major components i.e., the DES simulation, scheduling algorithm, entities and resources components.

DES simulation component: The core component to execute the developed simulator consists of the events, events scheduler, time advancing mechanism, random number generator, Termination Of Simulation (TOS) and statistical results.

A flow chart of the execution of the simulator is depicted in Fig. 5. It illustrates the structure of the simulation program and the events involved. The initialization triggers the deployment of the entire simulation. Relating the norm of an idle system, no task can depart without invoking its creation (i.e., Task Arrival Event). Thus, the assumption of the event arrival schedule is set to 0.0000.

Referring to Fig. 5, after initialization the next pre-requisite mandatory step is to scan the event list and select the event with the earliest time of occurrence. Mapping the selection to DES is embedded in the time advancing mechanism (i.e., simulation clock). The simulation clock is then advanced to the time of occurrence of the selected event. The simulator then executes the selected event and updates the system state variables affected by the event. Each of the identified events is auctioned by calling an associated event routine which results in the addition of future events to the event list. The execution of event routines is done to achieve the stipulated two purposes to:

- Model the deployment of an event and
- Track the resource consumption status of the event

Referring to Fig. 5, the defined events and their respective routine descriptions in this research are as follows:

- Task Arrival event
- Resource Request event
- Resource Release event
- Task Termination event

The completion of the simulation will be done upon the convergence of the repetitive structure to a predefined value which also known as TOS. TOS is critical in determining the validity of the acquired results. It must represent the system in entirety. In this research, the simulator is terminated if one of these two conditions is fulfilled:

- The event list is empty
- The arrival of task termination event for the final task (i.e., the Nth task) is executed

Entities component: Entities are the representation of objects in the system (Karatza, 2000; Law, 2003). Fig. 6 shows the interaction between the entities and resource models that are designed throughout the simulator. i.e., the source and tasks entities, the resources and a queue of an unordered task list named as utlist.

Source model: Simulating the source model involves the representation of the load generation of the system under study. It is vital to accurately represent the load to ensure the algorithms deployed are tested on the actual scenario.

A source injects a stream of tasks into the system. The maximum numbers of tasks are 1000 and denoted as MAX_TASKS. Upon generation, a task is executed for 0.50 seconds (i.e., the average execution time denoted as C_AVG). Given the task average execution time C_AVG and a load factor load, the tasks inter arrival time follows exponential distribution with mean value of C_AVG/load.

Tasks are generated via a Task Arrival event. The details of this event are depicted in Fig. 7. Every time the Task Arrival event is executed, the system increments the counter representing the number of generated task i.e., ntg by one. Each task is associated with an Initial time and Termination time.

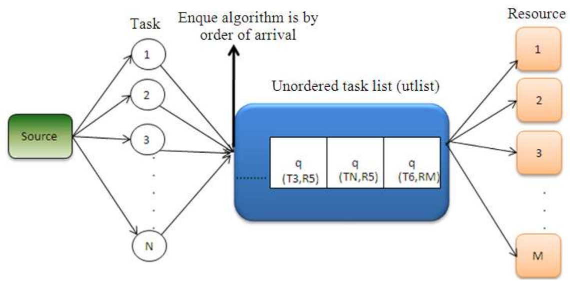

Task
Enque algorithm is by order of arrival
Resource
Source
1
2
3
N
Unordered task list (utlist)
q (T3,R5)
q (TN,R5)
q (T6,RM)
1
2
3
M

Fig. 6: Interaction of entities and resources

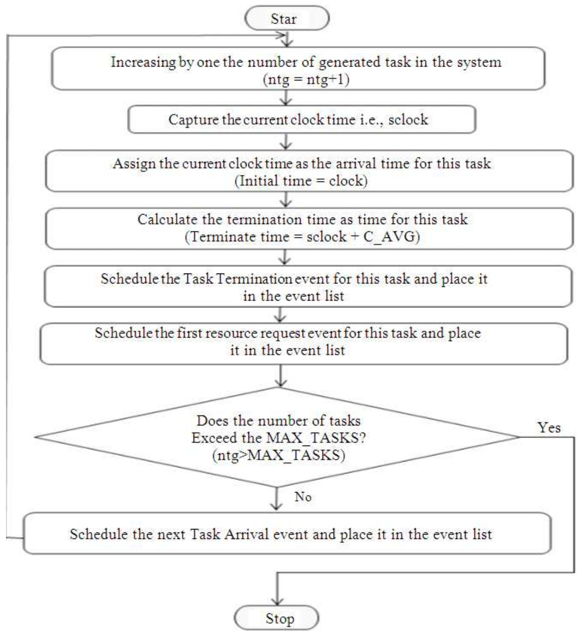

Star
Increasing by one the number of generated task in the system (ntg = ntg+1)
Capture the current clock time i.e., sclock
Assign the current clock time as the arrival time for this task (Initial time = clock)
Calculate the termination time as time for this task (Terminate time = sclock + C_AVG)
Schedule the Task Termination event for this task and place it in the event list
Schedule the first resource request event for this task and place it in the event list
Does the number of tasks Exceed the MAX_TASKS? (ntg>MAX_TASKS)
Yes
No
Schedule the next Task Arrival event and place it in the event list
Stop

Fig. 7: Task arrival event

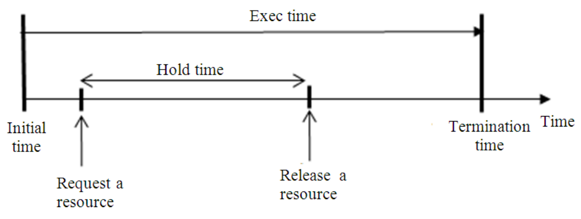


Fig. 8: Task model

The arrival time of the task into the system is denoted as the Initial time. It measures this value by capturing the current clock time denoted as schlock. The Termination time represents the absolute deadline of a task.

Task model: A micro abstraction of a source is the task model. Each task is associated with an integer number, denoted as tid. Each task is associated with an integer number, denoted as tid. Figure 8 shows a task as a single flow of execution.

During the lifetime of a task, it may request one or more resources. For each request, a task specifies the duration to hold the requested resource. This is denoted as Hold Time. The Exec Time denotes the remaining execution time of a task at a particular instant. Initially, at Initial time the value of Exec Time is equal to C_AVG. This value is reduced as the task is executed until the Termination time and the value of Exec Time becomes zero. It is assumed that a task releases all resources it acquires before it ends, complying with condition of the Hold Time≤Exec Time. The following assumptions are made for the task model implemented in this research:

- Independent task model, whereas each task has no dependency on other task during execution. The execution of a task has no correlation to the previously executed task
- Task can be preemptive, i.e., a task can be delayed or suspended to allow another task to be executed

TUF model: The timing constraint of a task is designed using the step TUF model in this research (Li *et al*., 2006). A TUF describes a task contribution to the system as a function of its completion time. The step TUF model is shown in Fig. 1. The maximum utility that could possibly be gained by a task is denoted as MaxAU. The random value of MaxAU abides normal distribution (10, 10) i.e., the mean value and variance is set 10 to conform to the benchmark. The Initial time is the starting time for which the function is defined. The Termination time is the latest time for which the function is defined. That is, MaxAU is defined in within the time interval of [Initial time, Termination time]. The completion of a task within this interval will yield positive utility i.e., MaxAU to the system. The completion of a task breaching the stipulated deadline causes the value of MaxAU to become zero. If the Termination time is reached and the task has not finished its execution, it accrues zero utility to the system.

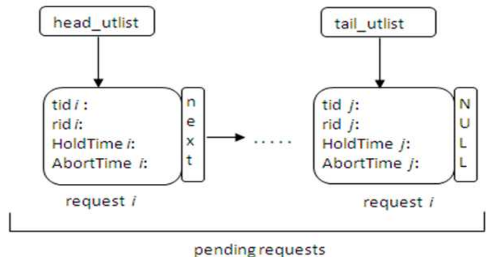


Fig. 9: Unordered task list (queuing model)

Queuing model: The constant amount of resources and surplusing demands results in resource unavailability. The simulator provides a mechanism to retain the task's requests for resources which are temporarily unavailable in an unordered task list named as utlist. A queue implementation via the pointer based single link list is used to deploy the utlist as shown in Fig. 9.

Referring to Fig. 9, the utlist consists of a sequence of pending request. A request for a resource is represented by a quadruple ReqResourceItem=<tid,rid, Hold time, Abort Time>. Thus, an element in the utlist consists of ReqResourceItem structure. A next pointer is used to link an element to the next element in the utlist. The head_utlist points to the first element and tail_utlist points to the final element in the utlist.

Resources component: The resource model represents the physical and logical resources. Logical resource can be defined as the management of physical resource whereby the "N" physical resources are treated as "M" instances. When the resource is currently being held by a task, resource is in the BUSY state. When a resource is not held by any task (i.e., in IDLE state).

When a task request a resource, the resource request event is depicted in Fig. 10.

Referring to Fig. 10, every time this event is executed, the system increments the counter representing the number of request in a task i.e., Treq.nrr by one. When a new request for a resource from a task Treq arrived in the system, the availability of the requested resource is checked. If the resource is in IDLE state which means it is available, task Treq is scheduled to immediately use the resource and the resource release event is scheduled in the event list. The status of the resource is changed to BUSY state and the owner of this resource is assigned to the task Treq. The GUS scheduling algorithm is implemented into the system for the case when the resource is currently in BUSY state being used by the owner task.

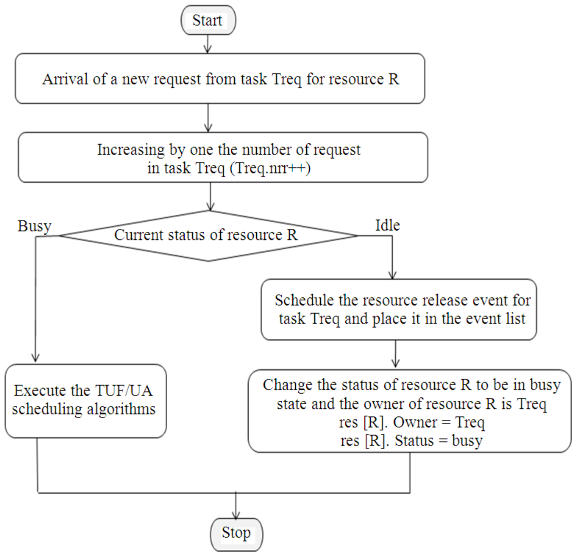

Start
Arrival of a new request from task Treq for resource R
Increasing by one the number of request in task Treq (Treq.nrr++)
Busy
Current status of resource R
Idle
Schedule the resource release event for task Treq and place it in the event list
Execute the TUF/UA scheduling algorithms
Change the status of resource R to be in busy state and the owner of resource R is Treq res [R]. Owner = Treq res [R]. Status = busy
Stop

Fig. 10: A resource request event

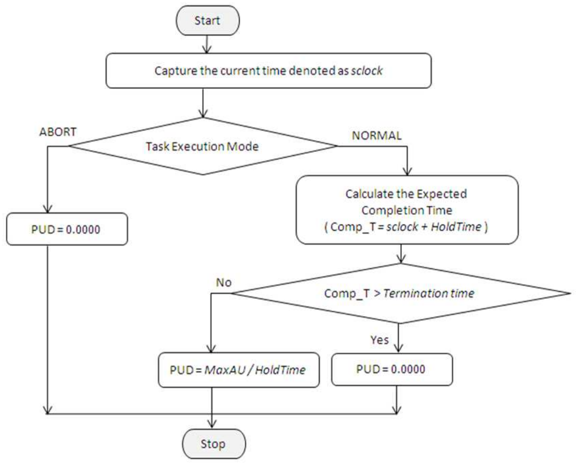
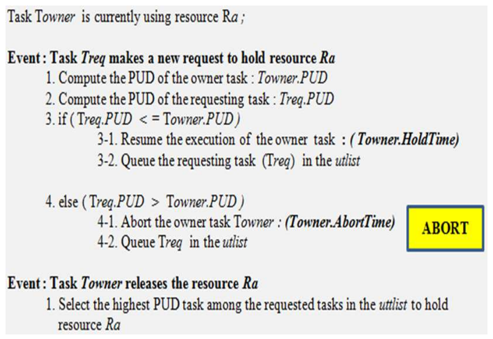

Start
Capture the current time denoted as sclock
ABORT
Task Execution Mode
NORMAL
PUD = 0.0000
Calculate the Expected Completion Time (Comp_T = sclock + HoldTime)
No
Comp_T > Termination time
Yes
PUD = MaxAU / HoldTime
PUD = 0.0000
Stop

Fig. 11: Calculation of PUD

Scheduling algorithm component: The scheduling algorithms component consists of the benchmark GUS algorithm. GUS is a TUF/UA uniprocessor scheduling algorithm that considers the step and arbitrary shape TUFs. The main objective of GUS is to maximize the utility accrued to represents that the most important task is to be scheduled first in the system. GUS uses a greedy strategy where task whose execution yields the maximum PUD over others is selected to determine which task to be scheduled at a particular instant.

The PUD of a task measures the amount of utility that can be accrued per unit time by executing the task. It essentially measures the Return on Investment (RoI) for executing the task at current clock time.


```
Task Towner is currently using resource Ra ;

Event : Task Treq makes a new request to hold resource Ra
    1. Compute the PUD of the owner task : Towner.PUD
    2. Compute the PUD of the requesting task : Treq.PUD
    3. if ( Treq.PUD < = Towner.PUD )
        3-1. Resume the execution of the owner task : ( Towner.HoldTime)
        3-2. Queue the requesting task (Treq) in the utlist

    4. else ( Treq.PUD > Towner.PUD )
        4-1. Abort the owner task Towner : (Towner.AbortTime)      ABORT
        4-2. Queue Treq in the utlist

Event : Task Towner releases the resource Ra
    1. Select the highest PUD task among the requested tasks in the uttlist to hold
       resource Ra
```

Fig. 12: GUS Scheduling algorithm

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

Fig. 11 shows the computation flow of PUD for a task at a particular time unit denoted as sclock. The current execution mode of a task is checked which maybe either the NORMAL or ABORT mode.

The PUD of a task that is currently executing in ABORT mode is zero as depicted in Fig. 11. This is because the maximum utility i.e., MaxAU for an aborted task is zero and consequently does not accrues utility to the system. For a task that is currently executing in NORMAL mode, the expected completion time of the task (i.e., Comp_T) is calculated. If the task is to be executed at current time i.e., sclock, the expected completion time of a task is equal to sclock + HoldTime where the HoldTime is defined as the time taken for a task to hold the respective resource. If the completion time exceeds the Termination time of the task, the utility becomes zero and consequently the PUD is equals to zero. If the task is scheduled to complete execution before Termination time, the execution of the task will yield positive utility i.e., MaxAU.

Figure 12 elaborates the GUS scheduling algorithm for the execution of an independent task model. When a new request from task Treq arrives into the system, GUS accepts the new request for resource Ra.

Referring to Fig. 12, when the resource Ra is currently being used by another task i.e., task Towner, GUS firstly calculates the PUD of both tasks. In the case that the requesting task i.e., Treq posses a higher PUD as compared to task Towner, GUS has tailored mechanism to abort the current owner task (i.e., Towner) and grant the resource to the requesting task (i.e., Treq).

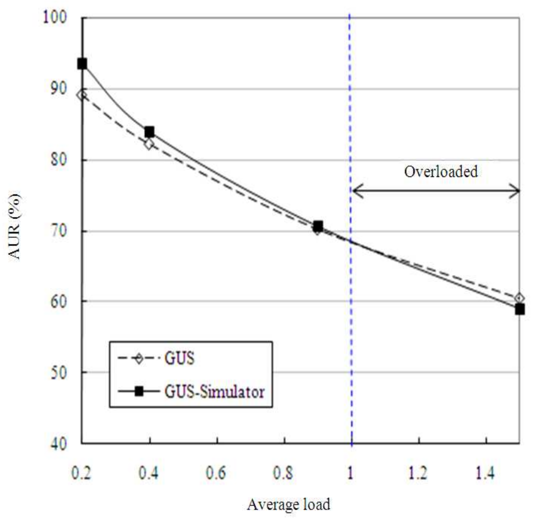


Fig. 13: AUR Results of the developed simulator and the benchmark model

The abortion procedures taken some processing time denoted as Abort Time while the request from task Treq is inserted into a utlist queue that containing the pending requests which are still waiting to be scheduled. After task Towner has releases resource Ra, GUS selects the highest PUD task among the tasks in the utlist to hold the resource Ra.

Experimental setting: The developed simulator has been tailored to map the characteristics of a uniprocessor scheduling. Table 2 summarizes the simulation parameter settings that are used throughout this research (Li *et al.*, 2006). A source generates a stream of 1000 tasks. Given the task average execution time C_AVG and a load factor load, the average task inter arrival time i.e., iat is calculated as the division of C_AVG over load and further utilized an exponential distribution to be further derived to reflect the intended system model. In all the simulation experiments, the value of C_AVG is set at 0.50 sec and the range value of load is from 0.20-1.50. The different value of load are to provide the derivation of differing mean arrival rates of tasks. The arrival of tasks is assumed to follow the exponential distribution. The system is said to be overloaded when (load >1.00) represented also as the mean arrival rate of 0.50 seconds (i.e., iat). This complementary representation of load can be utilized to show congestion as the iat is at its equal value to the execution ability to process a task.

The value of the HoldTime and AbortTime parameters are derived by the normal distribution with mean and variance is 0.25. The maximum utility of a task i.e., MaxAU is computed using normal distribution with mean value of 10 and variance of 10. It is assumed that the amount of available resources in the system i.e., MAX_RESOURCES are 5.

The performances of real time scheduling algorithms are measured by the metrics which rely on the respective application specifications. The Accrued Utility Ratio (AUR) metric defined in (Jensen *et al*., 1985) has been extensively utilized in the existing TUF/UA scheduling algorithms and is considered as the standard metric in this domain (Wu *et al*., 2004; Li *et al*., 2006).

AUR is defined as the ratio of accrued aggregate utility to the maximum possibly attained utility. Equation (1) shows that each task i has its maximum value of utility which is denoted as MaxAU(i). After a task i has completed its execution, it will yield a value denoted as Util(i). These values are then accumulated for all tasks i.e., MAX_TASKS. The AUR is calculated as:

$$\mathrm{AUR} = \frac{\sum_{i=1}^{\mathrm{MAX\_TASKS}} \mathrm{Utill(i)}}{\sum_{i=1}^{\mathrm{MAX\_TASKS}} \mathrm{MaxAU(i)}} \qquad (1)$$

RESULTS

Extensive experiments were done to ensure the developed GUS simulator is validated. The simulation model is validated by ensuring that its output data closely resemble the output data that was observed from the benchmark model i.e., GUS. A result obtained from simulator is compared with the result published in the literature by using the same assumptions and experimental setting (Li *et al*., 2006). Figure 13 depicts the AUR results of the developed GUS simulator and the original GUS. The result obtained using the simulation is comparable to the result published with the same trends.

Figure 13 depicts the AUR result under an increasing load. From the results, as the number of load is increased; a lower accrued utility is recorded. The developed GUS simulator is validated with less than 5% as compared to the benchmark model.

DISCUSSION

With the obtained result, this study has proven that the simulation of TUF/UA scheduling algorithm can be deployed in a common platform of discrete event simulator as a solution to the heterogeneity problem of the simulation tools.

CONCLUSION

In the past the research of utility accrual real time scheduling mainly uses the simulation tools as methodology to investigate the performances. With the development of a discrete event simulation this study has provided the design of the developed GUS scheduling algorithm by using DES. The aim of the developed simulation framework was not only to develop a GUS model for the research problem but also to provide a platform for future investigations involving TUF/UA real time scheduling.

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

- The GUS algorithm 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 algorithm
- The implementation of the fault tolerance in the TUF/UA scheduling domain

REFERENCES

Ahmad, I., M.F. Othman, I. Ahmad and M.F. Othman, 2009. Enhanced utility accrual scheduling algorithms for adaptive real time system. J. Comp. Sci., 5: 783-787. DOI: 10.3844/JCSSP.2009.783.787

Jensen, E.D., C.D. Locke and H. Tokuda, 1985. A time driven scheduling model for real time operating systems. Carnegie-Me//on University. http: //citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.7022&rep=rep1&type=pdf

Karatza, H., 2000. A comparative analysis of scheduling policies in a distributed system using simulation. J. Simulat., 1: 12-20. http: //ducati.doc.ntu.ac.uk/uksim/journal/issue-/HelenKaratza/HelenKaratza.pdf

Law, A., 2003. How to conduct a successful simulation study. Proceeding of the 2003 Winter Simulation Conference, IEEE Xplore Press, USA., pp: 66-70. DOI: 10.1109/WSC.2003.1261409

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

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