Journal of Computer Science 5 (11): 783-787, 2009
ISSN 1549-3636
© 2009 Science Publications

Enhanced Utility Accrual Scheduling Algorithms for Adaptive Real Time System

[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]Motorola Multimedia Sdn Bhd 3507 Prima Avenue, Jalan Teknokrat 5, 63000 Cyberjaya Malaysia

Abstract: Problem statement: This study proposed two utility accrual real time scheduling algorithms named as Preemptive Utility Accrual Scheduling (PUAS) and Non-preemptive Utility Accrual Scheduling (NUAS) algorithms. These algorithms addressed the unnecessary abortion problem that was identified in the existing algorithm known as General Utility Scheduling (GUS). It is observed that GUS is inefficient for independent task model because it simply aborts any task that currently executing a resource with lower utility when a new task with higher utility requests the resource. 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 UA scheduling algorithms are design for adaptive real time system environment where deadline misses are tolerable and do not have great consequences to the system. **Approach:** We eliminated the scheduling decision to abort a task in GUS and proposed to preempt a task instead of being aborted if the task is preemptive able. We compared the performances of these algorithms by using discrete event simulation. **Results:** The proposed PUAS algorithm achieved the highest accrued utility for the entire load range. This is followed by the NUAS and GUS algorithms. **Conclusion:** Simulation results revealed that the proposed algorithms were more efficient than the existing algorithm, producing with higher accrued utility ratio and less abortion ratio making it more suitable and efficient for real time application domain.

Key words: Adaptive real-time system, utility accrual scheduling, accrued utility ratio, 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 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)[1]. 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.

As illustrated in Fig. 1, completion of a task within the deadline (i.e., within the StartTime and TerminateTime) will accrue some positive utility (i.e., MaxAU) or zero utility otherwise.

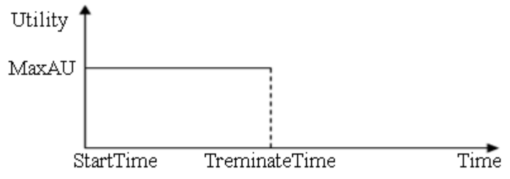

Fig. 1: The step TUF[1,2]

Objective: The scheduling objective of this research is to maximize the accrued utility from all executed tasks in the system. These criteria are named as Utility Accrual (UA) criteria[2]. A 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) when the system is overloaded.

As suggested in the recent overview of the UA scheduling domain[3], one of the existing algorithms

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

that provide general assurance on timeliness behavior is General Utility Scheduling (GUS) algorithm[4].

Problem statement: It is observed that GUS algorithm is inefficient for independent task model because every time a new task with higher utility requests a resource, the GUS simply aborts any task that is currently using the resource if the task produces lower utility. Figure 2 illustrates this inefficiency scenario. There is two tasks currently involved in the scenario i.e., task Towner and Treq. Table 1 summarizes the characteristics of these tasks. Task Towner request for a resource at time 1.0. After executing the resource for 0.10 sec, a new request from task Treq for the same resource arrived into the system. The Potential Utility Density (PUD) of both tasks is calculated. The PUD of a task measures the amount of utility that can be gained per unit time by executing the task[2]. Task Treq produced larger PUD (i.e., 36) than Towner (i.e., 25). GUS then decides to abort Towner for 0.075 sec before it releases the resource. GUS then allows Treq to execute the available resource. Execution of aborted task will accrue zero PUD and zero utility to the system. Clearly, sequencing tasks using the GUS algorithm accrued 9 utility (i.e., zero for Towner that has been aborted plus 9 for Treq).

We identified that the decision to immediately abort the lower PUD task is not necessary. Naturally for tasks that are independent each other, the decision to execute one task should not result from the abortion of another task. Task that has been aborted will not contribute any positive utility to the system. Therefore,

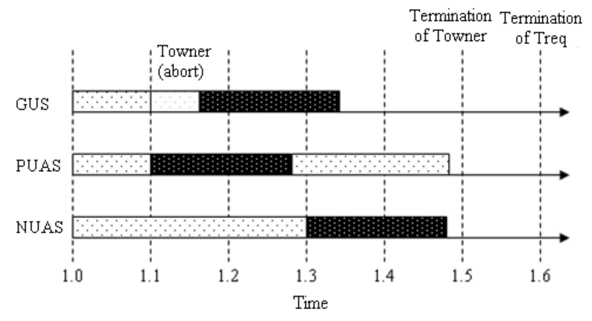

Fig. 2: Inefficiency scenario in GUS algorithm

Table 1: Task characteristics

Task Characteristic	Towner (white)	Treq (black)
Initial Holdtime	0.300	0.25
Remaining Holdtime	0.200	0.25
Aborttime	0.075	0.10
Maximum Utility (MaxAU)	5.000	9.00
PUD	5/0.20 = 25	9/0.25 = 36

we speculate that more unnecessary abortions occurred in GUS which could possibly reduce the tasks' attained utility. It is important to observe that by reducing the number of aborted tasks, it is very likely that we would gain higher utility.

Approach: To rectify the inefficiency identified in GUS, we proposed two solutions according to the preemptive nature of the task as stated below:

Preemptive Utility Accrual Scheduling (PUAS) algorithm: In this model, the owner task is preempted (i.e., suspended) temporarily instead of being aborted when a new request with higher PUD task arrived in the system. In PUAS, task with highest PUD is given the highest priority to hold the resource.

Non-Preemptive Utility Accrual Scheduling (NUAS) algorithm: In this model, the owner task continues to hold a resource without being aborted although it produces lower PUD when a new request with higher PUD task arrived in the system.

Figure 3 illustrates the scheduling decision made by the proposed algorithms after the arrival of a request from a task into the system. After the scheduler accepts a request from task Treq, it will first check the availability of the requested resource. If the resource is idle, task Treq can be scheduled immediately to use the resource. For the case when the resource is busy and

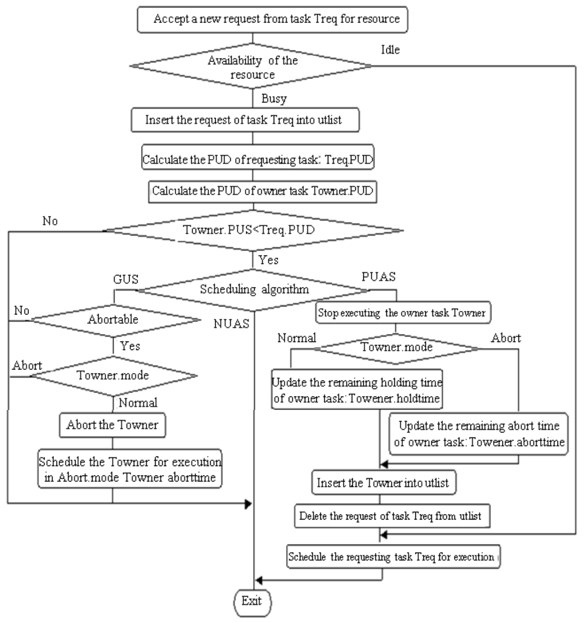
Accept a new request from task Treq for resource
Availability of the resource
Idle
Busy
Insert the request of task Treq into utlist
Calculate the PUD of requesting task: Treq.PUD
Calculate the PUD of owner task Towner.PUD
No
Towner.PUS<Treq.PUD
Yes
GUS
Scheduling algorithm
PUAS
No
Abortable
NUAS
Stop executing the owner task Towner
Yes
Normal
Towner.mode
Abort
Abort
Towner.mode
Normal
Update the remaining holding time of owner task: Towener.holdtime
Abort the Towner
Update the remaining abort time of owner task: Towener.aborttime
Schedule the Towner for execution in Abort.mode Towner aborttime
Insert the Towner into utlist
Delete the request of task Treq from utlist
Schedule the requesting task Treq for execution
Exit

Fig. 3: Flow charts of the UA scheduling algorithms

currently being used by the owner task Towner, the PUD for both tasks is compared. If requesting task Treq produced higher PUD:

- In GUS, Towner is aborted and immediately change its state from Normal to Abort mode
- In NUAS, Towner is continuously executed without being aborted, although it produced a lower PUD than Treq
- In PUAS, Towner is preempted instead of being aborted and Treq is granted to use the resource because it produced higher PUD than Towner

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 the previous research (i.e., GUS) was based on the discrete event simulation tools[2,4]. Therefore, in order to precisely remodel and further enhance the GUS algorithm, DES written in C language is the best method to achieve this objective. We used experiment settings that are similar to those proposed in[4].

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.

The task model is shown in Fig. 5. The average execution time 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.

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. However, some tasks cannot be aborted.

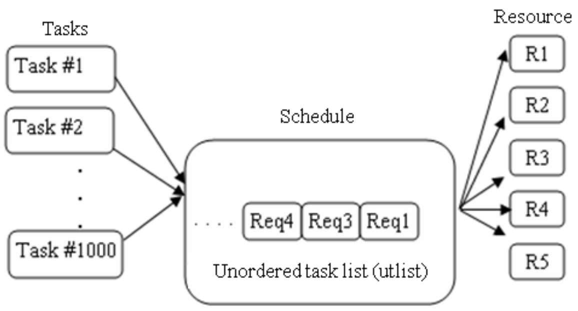

Fig. 4: Simulation model

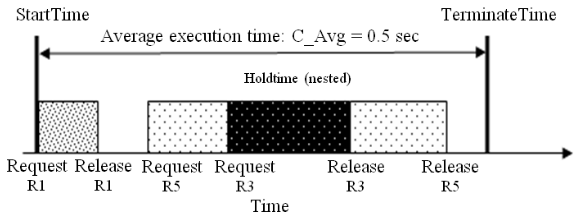

Fig. 5: Task model

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
Abortability	95%	Percentage of abortable tasks in the system

We refer to this aspect of a task as its Abortability. It is assumed that 95% (i.e., Abortability) of the executed tasks are abortable in the system. For those tasks that can be aborted, aborting 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.

RESULTS

The performances of UA scheduling algorithms are measured by the metrics that relies on the application specifications. For UA scheduling domain, the Accrued Utility Ratio (or AUR) metric defined in[3] has been used in many algorithms[1,3,4] and can be considered as a standard metric in this domain. AUR is defined as the ratio of accrued aggregate utility to the maximum possibly attained utility.

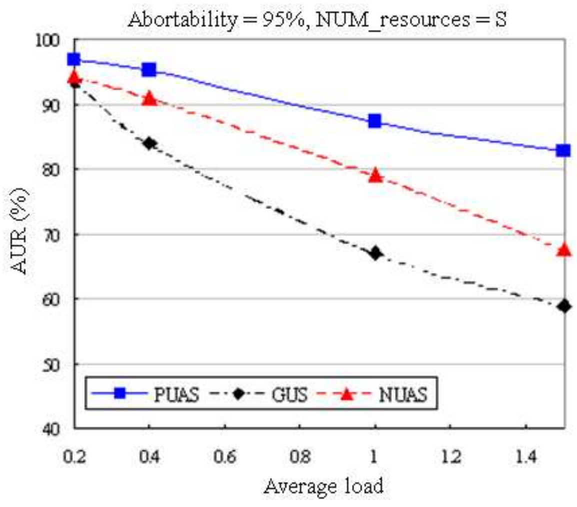

Fig. 6: AUR Vs average loads

In addition, we consider two other metrics to precisely examine the effectiveness of our proposed algorithms. The Success Ratio (or 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. The Abortion Ratio (or AR) is defined as the ratio of aborted tasks to the total of executed tasks. The results presented are intended to illustrate the characteristics of the proposed algorithms towards variation of the load in the system.

Figure 6 depicts the AUR result under an increasing load. The proposed PUAS algorithm achieved the highest accrued utility for the entire load range. This is followed by the NUAS and GUS algorithms. In lower loads, all algorithms performed better i.e., more than 90% of the tasks, accrued utility to the system. The gaps between these algorithms are relatively small and insignificant (i.e., 0.94-3%). However, as the load increases, the AUR gap widen significantly. In highest load, almost 81% of utility accrued in PUAS, 67.8% in NUAS and 59% in GUS. These gaps exist because GUS in nature has more aborted tasks compared to NUAS and PUAS. Since the aborted tasks produced zero utility to the system, consequently GUS produced more zero utility tasks that ultimately contributed to lowest accrued utility.

Figure 7 plots the task success ratio experienced as a function of the increasing loads. Figure 7 supports the AUR results in Fig. 6 because it measures the exact number of tasks that has successfully contributed to AUR.

In Fig. 8, we can see the abortion ratio results in the system. As mentioned in the first section, we speculate

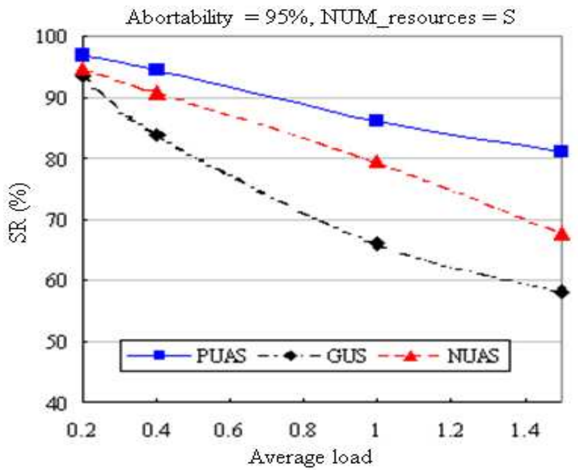

Fig. 7: SR Vs average load

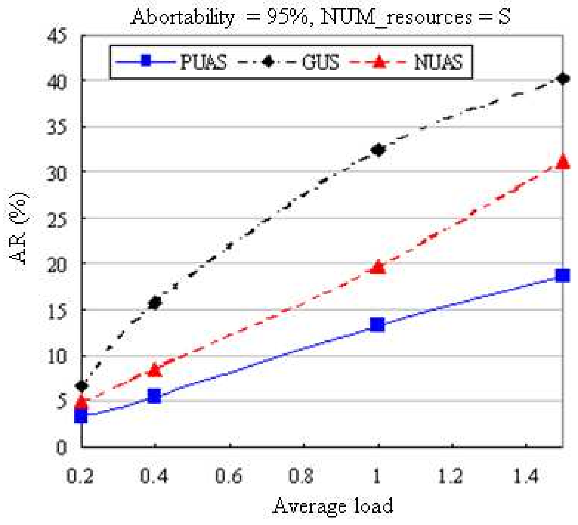

Fig. 8: AR Vs average load

that the existing algorithm GUS produced high number of aborted task that we believed can be resurrected in our proposed algorithms. Figure 8 verified the speculation. It can be observed that the proposed PUAS and NUAS algorithms are able to reduce the number of abortion compared to GUS algorithm. This justifies why higher utility can be accrued in the proposed PUAS and NUAS algorithm compared to GUS.

DISCUSSION

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

CONCLUSION

In this study we proposed an efficient UA real time scheduling algorithms called PUAS and NUAS that considers task subjected to deadline expressed using step TUFs. The proposed algorithms are compared with the existing UA algorithm known as GUS[4]. Simulation results reveal that PUAS outperform the NUAS and GUS with highest accrued utility and lowest abortion ratio making it more suitable 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 PUAS and NUAS on real-time POSIX-compliant operating system using the meta-scheduling framework can also demonstrates the effectiveness of these algorithms

REFERENCES

1. 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
2. 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
3. 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. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01420952
4. 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