# A Preemptive Utility Accrual Scheduling Algorithm for Adaptive Real Time System

**Idawaty Ahmad, S.Shamala, M.Othman[†] and Muhammad Fauzan Othman[††]**

[†]Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 Serdang Malaysia
[††]Motorola Multimedia Sdn Bhd, 3507 Prima Avenue,Jalan Teknokrat 5, 63000 Cyberjaya Malaysia

**Summary**
In this paper, we propose a preemptive utility accrual scheduling (or PUAS) algorithm as an enhancement to General Utility Scheduling ( or GUS) algorithm proposed by Peng Li [1]. These scheduling algorithms are designed for adaptive real time system environment where undesirable effects such as overload and deadline misses are tolerable. We consider independent task models that are subject to deadline constraints specified using step time/utility functions (or TUFs). The basic idea of our algorithm is to reduce the number of unnecessary abortion that occurs in GUS by preemption instead of abortion. We consider the scheduling objective of maximizing the utility that is accrued by the completion of all tasks. Simulation results revealed that the proposed algorithm outperforms GUS algorithm. By reducing the total number of task aborted together with lower abortion ratio, this in effect produced a higher utility and reduced the average response time, making it more suitable and efficient in time-critical application domain.

***Key words:***
*Adaptive real time system, Utility Accrual Scheduling, Time/Utility Functions (TUFs), Accrued Utility Ratio (AUR), Abortion Ratio (AR).*

## 1. Introduction

A real time system is a system where the time at which events occur is important. Real-time scheduling is fundamentally concerned with satisfying application time constraints. While early research on real time scheduling was primarily focused on complying avoidance of undesirable effects such as overload and deadline misses, adaptive real time systems are designed to handle such effect dynamically by softly degrading performances. In adaptive-soft real time system, an acceptable deadline misses and delays are tolerable.

During resource overloads, meeting the deadlines of all tasks is impossible as the demand exceed the supply. The urgency of task is typically orthogonal to the relative important of the task. The most urgency task can be least important, and vice versa. When overloads occur, it is often desirable to complete tasks that are more important than those which are more urgent. Thus, a clear distinction has to be made between the urgency and the importance of a task. A deadline by itself cannot express both urgency and importance. In this paper, we consider the time/utility functions (or TUFs) that express the utility of completing a task as an application-specific function of when a task completes. This formulation was initially proposed by Jensen in [2], [3]. In this paper we specify the deadline constraint of a task as a binary-valued, downward step shaped TUF. As illustrated in Fig 1, a TUF decouples importance and urgency- i.e., urgency is measured as a deadline on the X-axis and importance is denoted by utility on the Y-axis.

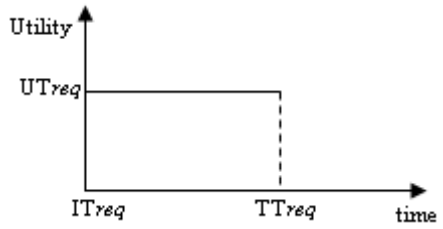


Fig. 1 The Step TUF that specify deadline and importance [1].

The scheduling optimality criteria are based on maximizing accrued utility from those tasks i.e., maximizing the sum of tasks' attained utilities. These criteria are named as Utility Accrual (or UA) criteria. Scheduling algorithms that consider UA criteria are classified as UA algorithms. A UA algorithm that maximizes the sum of tasks' attained utilities will seek to meet all task deadlines when under loads and during overloads it naturally tend to favor task that are more important (from whom higher utility can be accrued) than those that are more urgent.

As suggested in the recent overview of the UA real time scheduling domain [4], the current algorithm that provides general assurance on timeliness behavior is GUS (Generic Utility Scheduling) algorithm that is presented in Peng Li's PhD thesis [1]. However, it is observed that this algorithm is inefficient for independent task model in the sense that it simply aborts any task with lower utility

Manuscript received May 5, 2008
Manuscript revised May 20, 2008

without accounting that those tasks just need to be pre-empted (i.e., suspended) instead of being aborted. Task that has been aborted will not contribute any positive utility to the system. Therefore, we speculate that more unnecessary abortions occurred in GUS that could possibly reduce the tasks' attained utility.

To overcome the unnecessary abortion characteristic of GUS, we proposed a pre-emption enabled version of GUS algorithm named as PUAS (Preemptive Utility Accrual Scheduling) algorithm. Instead of abort, we suspend (preempt) task with lower utility that currently holding a resource and allow task with higher utility to execute and hold the resource.

## 2. The PUAS Algorithm

This section briefly describes the PUAS algorithm, and extension of GUS algorithm. For comparison purposes, most of the definition and assumptions of PUAS algorithm are similar to the GUS model. We apply the Jensen's TUFs [3] to define the time constraints of a task. As shown in Fig 1, each task Treq has an initial time ITreq and a termination time TTreq. Initial time is the earliest time for which the function is defined and termination time is the latest time for which the function is defined. As illustrated in Fig 1, UTreq is defined in the time interval of [ITreq, TTreq]. Beyond that, UTreq is undefined. If the termination time of a task is reached and the task has not completed its execution, it will then be aborted. Aborting a task will change the task from Normal to Abort mode. Completion of a task before the deadline in Normal mode accrues some uniform utility and accrues zero utility otherwise. Thus, finishing a task in Abort mode will accrues zero utility.

Following [1], our proposed algorithm measures the metric called Potential Utility Density (or PUD) that was originally developed in [3]. As shown in Fig 2, the PUD of a task measures the amount of utility that can be gained per unit time by executing the task. Thus, executing task in Abort mode will accrues zero PUD. It is important to observe that by reducing the number of tasks in Abort mode, it is very likely that we would gain higher utility.

A description of PUAS after accepting a task request is shown in Fig 3. When the scheduler accepts a request from task Treq, it will first check the availability of the requested resource. If the resource is in idle mode, task Treq can be schedule immediately to use the resource. For the case when the resource is in busy mode and currently being used by the owner task Towner, we compare the PUD for both tasks. If requesting task Treq produced higher PUD, then the owner task Towner will stop executing the resource and let the T*req* to be scheduled for execution and hold the resource.

**Calculation of PUD for a requesting task T*req***

Case : **NORMAL** mode:

$$T_{req}.PUD = \frac{\text{Utility gained at time t , UT}_{req}}{T_{req}\text{ remaining holding time, }T_{req}.holdtime}$$

Case: **ABORT** mode:
$T_{req}.PUD = 0.0000$

Fig. 2 The calculation of Potential Utility Density (or PUD).

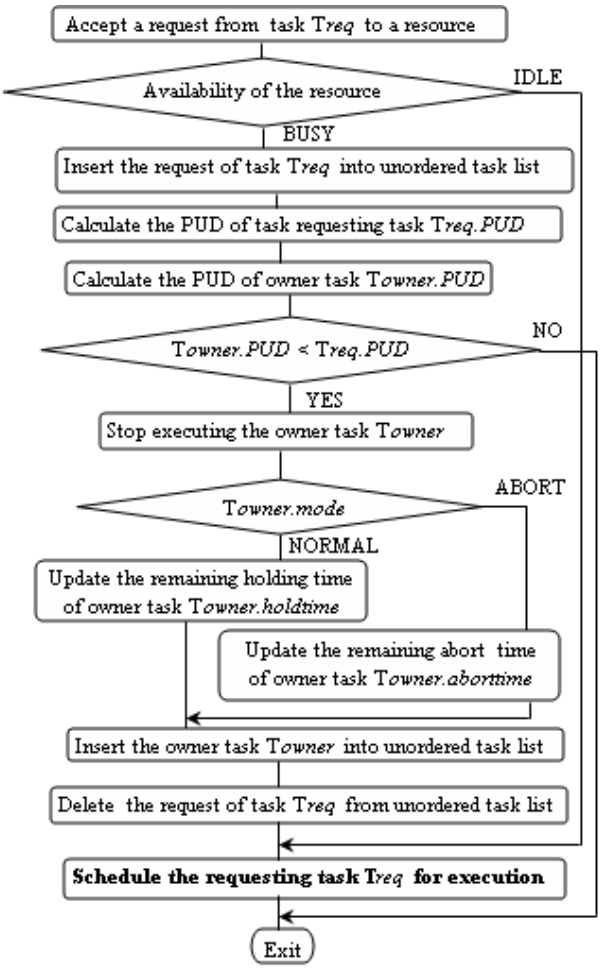


Fig. 3 The PUAS Algorithm.

## 3. Experimental Model

As suggested in the overview of the performance analysis technique used in UA real time scheduling domain [5], we develop a discrete event simulation (or DES) to verify the performance of our proposed algorithm. The rationale of using DES lies in the fact that the previous work of GUS algorithm was based on discrete event simulation tools using OMNET++ [1]. The other UA scheduling algorithms such as LBESA, DASA and EUA were also built on DES based tools [3], [6], [7]. This is simplified in Table 1. Thus, we believe that in order to precisely remodel and further enhance the GUS algorithm, DES written in C language is the best method to achieve this objective. For comparison purpose, similar experimental settings to Peng Li's PhD thesis [1] are used.

Table 1: Performance analysis technique used in the UA scheduling algorithms

| UA scheduling type | Simulation technique | Underlying Programming language |
|---|---|---|
| GUS | OMNET++ (Tool) | C/C++ ( DES ) |
| DASA | SIMCRIPT | C/C++ ( DES ) |
| LBESA | SIMCRIPT | C/C++ ( DES ) |
| EUA | OMNET++(Tool) | C/C++ ( DES |
| PUAS | GPL (General Public Language) | C ( DES) |

### 3.1 Simulation Model

Fig 4 shows the entities involve in our simulation study. It consists of a stream of 1000s task that are exponentially generated, an unordered task list, the proposed scheduler and a set of resources. The Held_Resource entity denotes the set of resources that is currently held by a task. These tasks are assumed to be independent of each other.

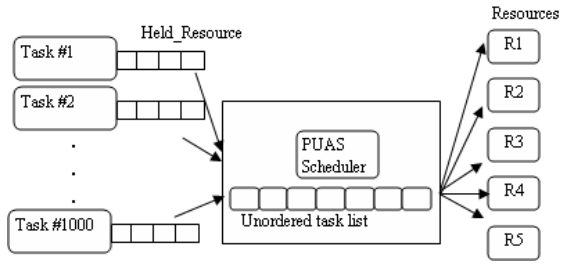


Fig. 4 Simulation Model.

During the execution time of a task, it may request one or more shared resources. Fig 5 shows the possibility of having a nested requested time interval. In Fig 5, a task request three resources, which are R1, R3 and R5. The task requests resource R5, then request resource R3 before it releases resource R5. Thus, the time interval of task holding resource R5 and R3 are considered as nested.

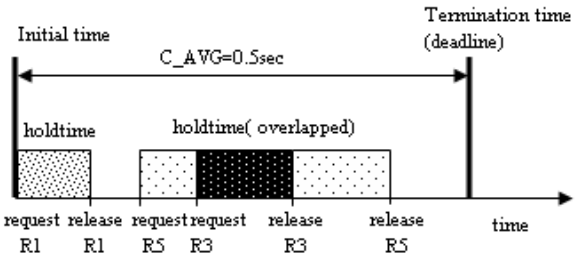


Fig. 5 Task Model

Table 2 summarized the details task settings in our simulation. Given the task execution time C_AVG and a load factor *load*, the average task inter-arrival time *iat* can be calculated as C_AVG/*load*. The *iat* is applied to exponential distribution as depicted in Table 2. The maximum utility of each task follows normal distribution.

Table 2: Simulation Parameters

| Parameters | Descriptions | Range |
|---|---|---|
| C_AVG | Task average execution time | 0.5 sec |
| *load* | Range of load | 0.2 to 1.5 |
| *iat* | Task inter arrival time | Exponential (C_AVG/*load*) |
| holdtime | Time for holding a resource | Normal (0.25,0.25) |
| max_au | Task's Maximum Utility | Normal(10,10) |
| MAX_TASKS | Number of task in the system | 1000 |
| MAX_ RESOURCES | Number of available resource | 5 |
| ABORTABILITY | Percentage of abortable tasks | 95% |

### 3.2 Performance Metrics

It has been stated in [8], that the performance of real time scheduling algorithm is measured by the metrics that relies on the application specs. For UA scheduling domain, the Accrued Utility Ratio (or AUR) metric defined in [2] has been used in many algorithms in [1], [6], [7] and can be considered as standard metric in this domain. AUR is defined as the ratio of accrued aggregate utility to the maximum possibly attained utility.

In addition, we proposed three new metrics that known as Success Ratio, Abortion Ratio and Average Response Time to precisely evaluate the performances of the

proposed algorithm. These performance metrics enable us to determine whether the proposed algorithm is efficient and robust in overloads situation. The Success Ratio (or SR) is the ratio of task attained positive utility to the total of task executed in the system. The SR supports the result of AUR in the sense that it measures the exact number of tasks that has contributed to AUR produced utility to the system. The Abortion Ratio (or AR) is defined as the ratio of aborted task to the total of executed task. As mentioned in the first section, we speculate that the existing algorithm GUS produced higher number of aborted task that we believed can be resurrected in our proposed algorithm. The AR metric verified the speculation. The Average Response Time (or ART) measures the average time taken for a task to complete its execution time after fulfilled its entire requests. The ART metric shows the timing effect of the proposed algorithm.

## 4. Results and Discussions

Fig 6 and Fig 7 show the AUR and SR under an increasing load respectively. As Fig 6 shows, the proposed algorithm has better performance by producing higher utility than in GUS for the entire load range. Fig 7 denoted the total number of tasks that contribute to AUR. From these figures, we observe that in underloads both algorithms performed better i.e., more than 90% of the tasks accrued utility to the system. The gap between GUS and PUAS are relatively small and insignificant (i.e., ≈3%). However, as the load increases, the AUR and SR gap increased significantly. In overloads, almost 81% of the task executed in PUAS algorithm gained utility compare to 59% in GUS. The gap between these algorithms is high (i.e., ≈22%) in overloads condition. We speculate that, the AUR and SR gap exist between GUS and PUAS is because GUS has more aborted task than PUAS. Since the aborted task produced zero utility, consequently GUS produced more zero utility tasks than in the proposed algorithm. In higher load, the numbers of preempted task are higher, which means higher number of aborted tasks can be avoided that in ultimately broaden the gap. Fig. 8 verifies our assumption, which proves that the abortion ratio in GUS is higher than in PUAS, which in turn leads to lower utility accrued. From Fig 9, we further observe that, by reducing the percentage of aborted tasks, together with the reduction of abortion ratio, this is in turn not only contribute to higher utility but also reduce the average response time making it more suitable and efficient in time critical application domain.

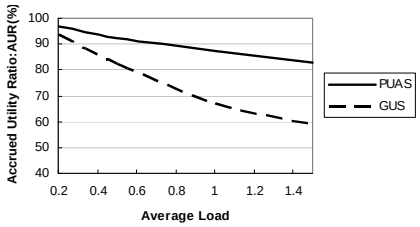


Fig. 6 Accrued Utility Ratio

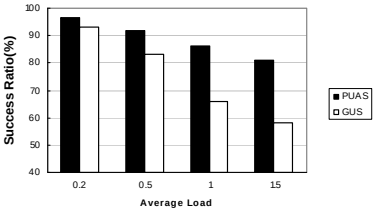


Fig. 7 Success Ratio

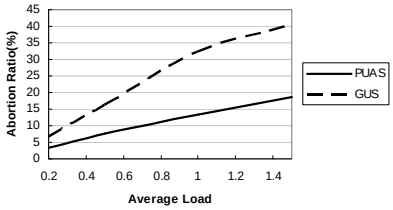


Fig. 8 Abortion Ratio

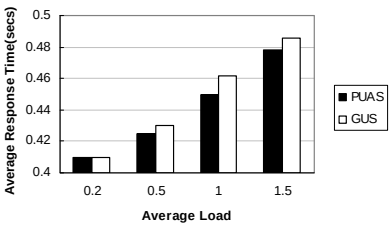


Fig. 9 Average Response Time

## 5. Conclusion

This paper presents an efficient UA real time scheduling algorithm called PUAS that considers tasks subjected to deadlines expressed using step TUFs. We developed a discrete event simulator to evaluate the performance of PUAS algorithm that target to maximize the accrued utility. We compared our proposed algorithm with the current existing UA algorithm named GUS algorithm. The PUAS algorithm outperforms the GUS with higher accrued utility, less abortion ratio and smaller average response time, making it more suitable and efficient in real time application domain.

However, this research is still at its early stage of study. There still remain several issues to be resolved such as:

(i) Applying the proposed PUAS algorithm to various shape of TUF scheduling such as soft-step, linear, parabolic, multimodal and arbitrary shapes.

(ii) Designing an error recovery algorithm to measure the proposed PUAS algorithm in error-prone environment. (In progress)

(iii) Implementing in Real Time Operating System (RTOS) to observe the actual behavior of PUAS algorithm.

### Acknowledgments

The authors would like to express their cordial thanks to Dr. Zuriati Ahmad Zulkarnain for her valuable advice.

## References

[1] Peng Li, "Utility Accrual Real Time Scheduling: Models and Algorithms", Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2004.

[2] E.D.Jensen, C.D.Locke and H.Tokuda, "A time driven scheduling model for real time systems," in Proceeding of IEEE Real-Time System Symposium, pp.112-212, December 1985.

[3] C.D.Locke, "Best-effort decision making for real time scheduling,"Ph.D. dissertation, Carnegie Mellon University CMU-CS-86-134, 1986.

[4] B.Ravindran, E.D.Jensen, P.Li, "On Recent Advances in Time/Utility Function Real-Time Scheduling and Resource Management," in Proceeding of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pp.55-60, 2005.

[5] Idawaty Ahmad, "Real-Time Task Scheduling Algorithms in Multiprocessor Environment,"in Proceeding of the National Information Communication Technology (NaICT'06), ISBN-983-42570-1-5, pp.142-150, 2006.

[6] R.K.Clark, "Scheduling Dependent Real-time Activities,"Ph.D. dissertation, Carnegie Mellon University CMU-CS-90-155, 1990.

[7] H.Wu, B. Ravindran, P.Li, "CPU Scheduling for Statistically- Assured Real-Time Performance and Improved Energy Efficiency," in Proceeding of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS), pp.110-115, September 2004.

[8] J.A.Stankovic, K.Ramamitham," Advanced in Real-Time Systems",Computer Society Press, Los-Alamitos,California,1993.

**Idawaty Ahmad** received theB.Sc. and M.Sc. degrees in Computer Science from Saga University (Japan), in 1998 and 2000, respectively. She is a full-time lecturer at the Department of Communication Technology and Network, University Putra Malaysia, pursuing her PhD at the same institution. Her current research interest includes real time system and simulation modeling. She is also an associate researcher for High Performance Computing at the Institute of Mathematical Research (INSPEM), University Putra Malaysia.

**Shamala Subramaniam** completed PhD from University Putra Malaysia in 2002. She is a lecturer at the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia. Her current research interest includes wireless and mobile network, simulation and modeling. She is an associate researcher for High Performance Computing at the Institute of Mathematical Research (INSPEM), University Putra Malaysia.

**Mohammad Othman** completed PhD from University Kebangsaan Malaysia in 1999 (with best PhD thesis awarded by Sime Darby Malaysia and Malaysian Mathematical Science Society). Currently he is Associate Professor at the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia. His research interest includes parallel and distributed algorithms, high-speed computer network, and multiprocessor system on chip.

**Muhammad Fauzan Othman** received the B.Sc in Computer Science from University Putra Malaysia in 2003. Currently as a senior principal engineer at Motorola Technology Sdn Bhd, his research interest includes clustering, virtualization and distributed computing.