Journal of Computer Science 8 (8): 1225-1234, 2012
ISSN 1549-3636
© 2012 Science Publications

Performances of Partition Utility Accrual Real Time Scheduling Algorithm

Idawaty Ahmad, Shamala Subramaniam,
Mohamed Othman and Zuriati Zulkarnain
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: This study proposed a TUF/UA real time scheduling algorithm known as Partition Preemptive Utility Accrual Scheduling (PUAS) also known as PPUAS algorithm. This algorithm addressed the overloaded problem that was identified in a uniprocessor scheduling environment and the necessity to design the scheduling algorithm in a multiprocessor environment. **Approach:** The PUAS algorithm was enhanced into the partitioned multiprocessor environment. The comparison of PUAS and PPUAS were made by using a discrete event simulation. **Results:** The proposed PUAS algorithm achieved a higher accrued utility for the entire load range as compared in the uniprocessor environment. **Conclusion:** Simulation results revealed that the proposed algorithms PPUAS are more efficient than the existing PUAS algorithm, producing a higher utility ratio and less abortion ratio making it suitable and efficient for real time application executed in multiprocessor environment.

Key words: TUF/UA scheduling, real time system, partitioned, multiprocessor

INTRODUCTION

Real-time scheduling is fundamentally concerned with satisfying application time constraints. In an adaptive real time system an acceptable deadline misses and delays are tolerable and do not have great consequences for the system. One of the scheduling paradigms in adaptive real time system environment is known as Time Utility Function/Utility Accrual (TUF/UA) scheduling paradigm (Li *et al*., 2006).

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. The value of utility for each executed task is accumulated and the total attained utilities are measured.

The latest trend of TUF/UA scheduling algorithm is moving towards the multiprocessor and distributed environment (Dellinger *et al.,* 2011; Ji *et al*., 2010). One of the existing uniprocessor TUF/UA scheduling algorithms is known as PUAS (Preemptive Utility Accrual Scheduling) algorithm (Idawaty *et al*., 2011). PUAS is a uniprocessor scheduling algorithm that manages the independence tasks in preemptive environment. In PUAS, a task that is currently executed in a lower PUD is temporarily suspended and a new task with a higher PUD is given the highest priority to hold a resource.

Problem statement: In the presence of extremely overloaded tasks traffic, it is observed that PUAS implemented in uniprocessor environment is inefficient due to the limited resources available in the system. A real time system requires a multiprocessor environment with larger number of resource capability to accommodate the surplus load. For ensuring that the system provides a higher utility under the highly loaded conditions, the TUF/UA scheduling algorithms operated in multiprocessor platform are proposed which are essential for providing an efficient real time system. The proposed algorithm known as PPUAS (Partitioned PUAS) has enhanced the existing PUAS algorithm.

Objective: The scheduling objective of PPUAS in this research is to maximize the total accrued utility from all executed tasks in the system by implementing the scheduling algorithm into the partitioned multiprocessor environment.

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

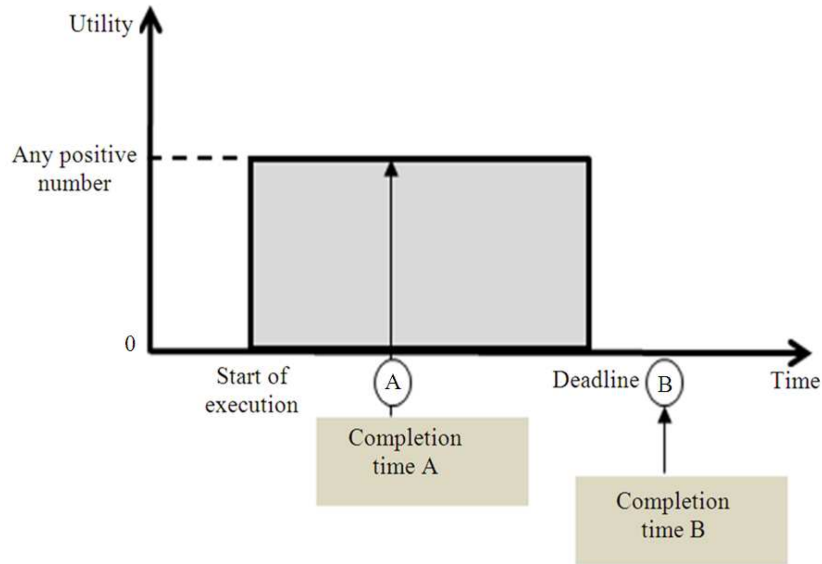

Fig. 1: The step TUF (Idawaty *et al*., 2011)

Approach: In order to evaluate and validate the performance PPUAS, a simulation model for the TUF/UA scheduling environment for the partitioned multiprocessor environment is deployed. In partitioned scheduling, once a task is generated it is assigned to a single processor and permanently located and scheduled at the respective processor.

MATERIALS AND METHODS

A Discrete Event Simulation (DES) is used as methodology to verify the performance of PPUAS algorithm. The rationale of using DES lies in the fact that the PUAS algorithm was based on the DES written in C language, thus the best method to achieve this objective. Fig. 2 shows the deployed simulation model. The multiprocessor infrastructure consists of a source and tasks entities, an array of *utlist* queues to represent the various numberS of processors and resources in the system.

Source model: 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 random execution time with mean of 0.50 seconds. For the purpose of implementing the multiprocessor environment into the simulation model, two steps are taken after a task has been generated as follows:

Step 1: Assign a task to its specific processor by using a task assignment algorithm. All tasks are assigned to processors by task assignment algorithm as shown in Fig. 2 and 3. The *cpuid* parameter is used to identify the assigned processor ID of a task.

Step 2: Execute the PPUAS scheduling algorithm.

Task model: Each task is associated with an integer number, denoted as tid. Each task is also associated with a start of execution time (i.e., Initial time) and a deadline (i.e., Termination time) as shown in Fig. 1. The arrival time of the task into the system is denoted as the Initial time. The Termination time represents the absolute deadline of a task. This research considered the step and arbitrary TUF task model as shown in Fig. 4.

The step TUF model used in the simulator is shown in Fig. 4a. 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. The *InitialTime* is the starting time for which the function is defined. The *TerminationTime* is the last time for which the function is defined.

The arbitrary shape TUF is represented as a continuous and derivable polynomial equation derived from the literature (Li *et al*., 2006). 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. For arbitrary TUF, the completion of a task within the *InitialTime* and *TerminationTime* interval will yield a random positive utility denoted as *Utility* as shown in Fig. 4b.

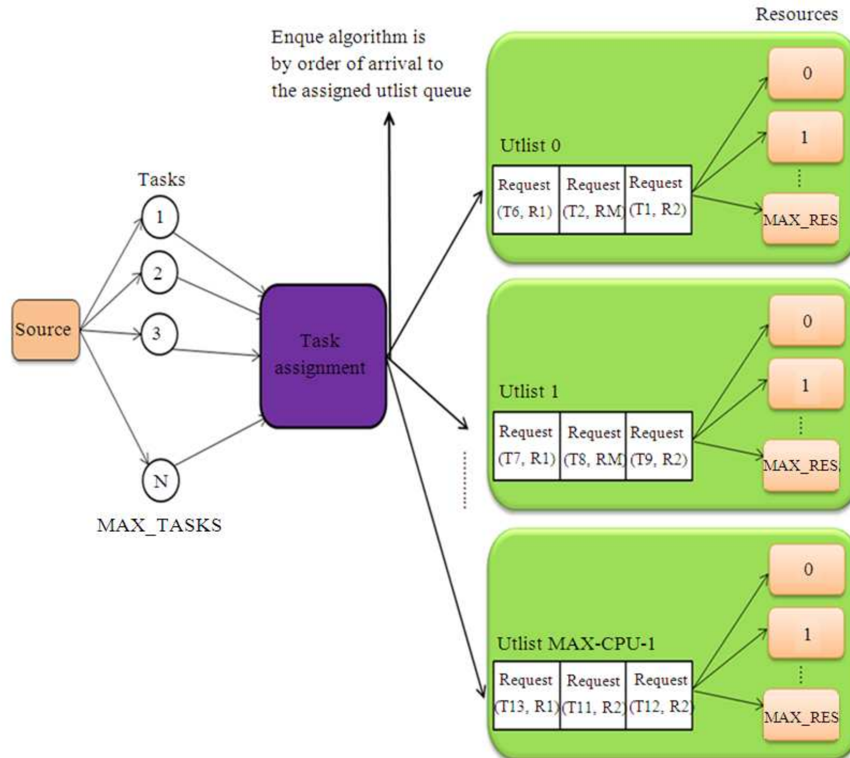

Fig. 2: Simulation Model

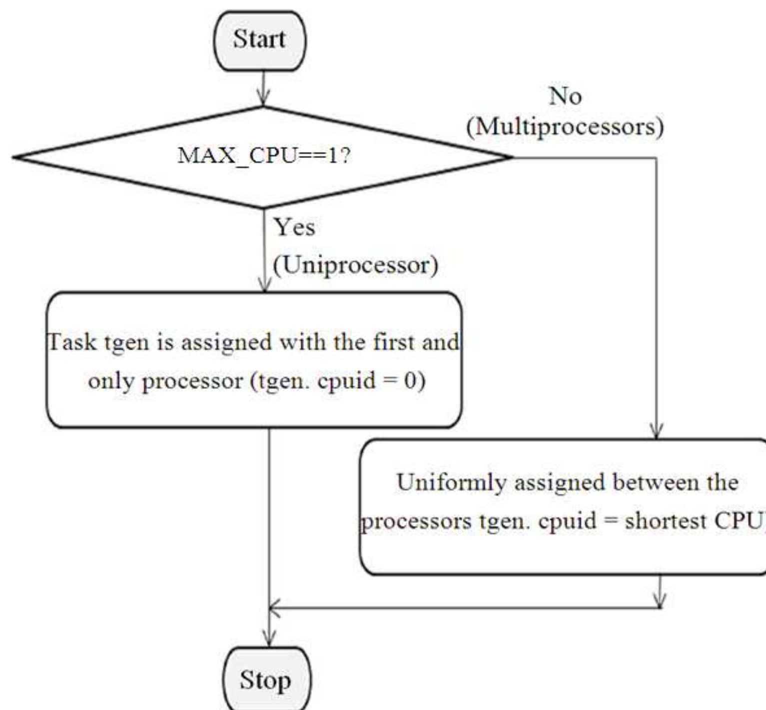

Fig. 3: Task Assignment Algorithm

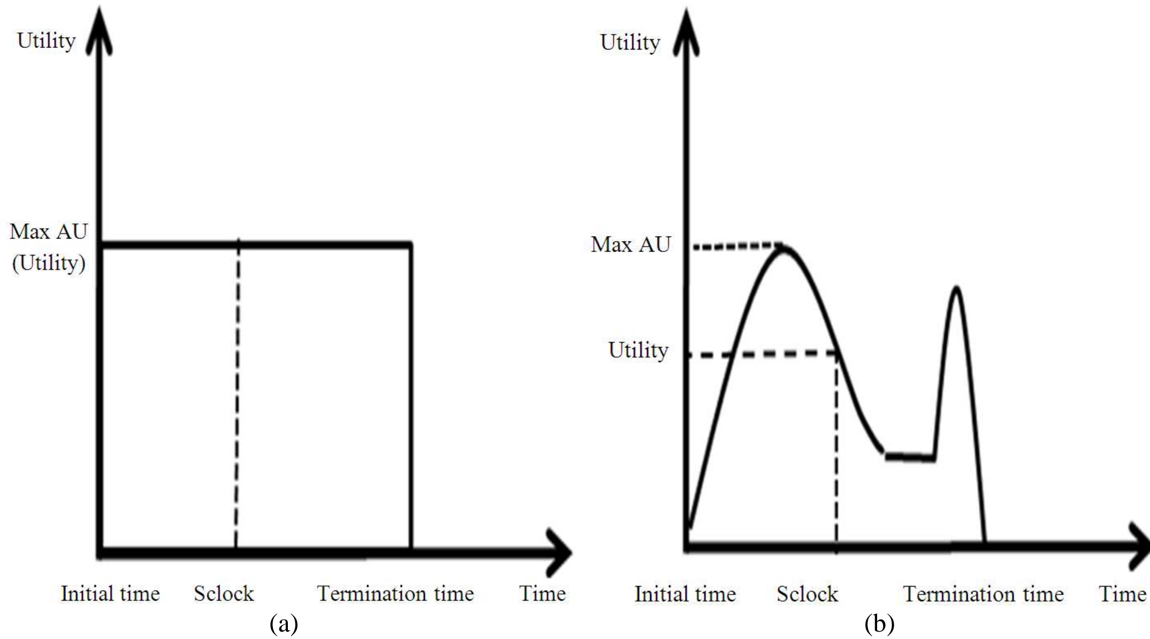

Fig. 4: Step and arbitrary TUF task set (Li *et al*., 2006) (a) step (b) arbitary

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. The number of utlist queues represents the number of processors in the system. The number of available processors is depicted in the MAX_CPU parameter as shown in Fig. 2.

Resources component: The resource model represents the physical and logical resources. The number of available resources in each processor is depicted in the MAX_RES parameter as shown in Fig. 2. The number of resources in the uniprocessor environment is limited to 5 according to the literature (Idawaty *et al*., 2011; Li *et al*., 2006). The total number of available resources in all processors is shown in the MAX_RESOURCES parameter which is calculated as the MAX_RES * MAX_CPU.

When a task request a resource, the resource request event is depicted in Fig. 5. 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.

Referring to Fig. 5, 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*.

In the case the requested resource is currently being used by the owner task *Towner*, the PUD for both tasks is compared. If the requesting task *Treq* produced a higher PUD, *Towner* is preempted and inserted into the *utlist*. Task *Treq* is granted to use the resource. The status of the resource is changed to BUSY state and the owner of this resource is assigned to the task *Treq*. The *HoldTime* i.e., the time taken to hold resource R is randomly assigned to task *Treq* in the *Treq. HeldRes[R]. HoldTime parameter*. The expected release time of resource R i.e., Treq. *HeldRes[R]. ReleaseTime* is calculated as sclock + *Treq. HeldRes[R]. HoldTime*.

When a task releases a resource, the resource release event is executed as shown in Fig. 6 and 7. The number of resources that have been released by task *Towner* is captured and represented by the *Towner.nrp* parameter which is incremented by one for every resource released made for a request in task *Towner*. This parameter is used to capture the number of requests that have been released when the termination time event for task *Towner* arrived into the system. The steps taken when a resource R is released by the owner task i.e., *Towner* is done in two consecutive phases as stated below:

Phase 1: After resource R is released by the owner task, the status of R is reset from BUSY to IDLE state.

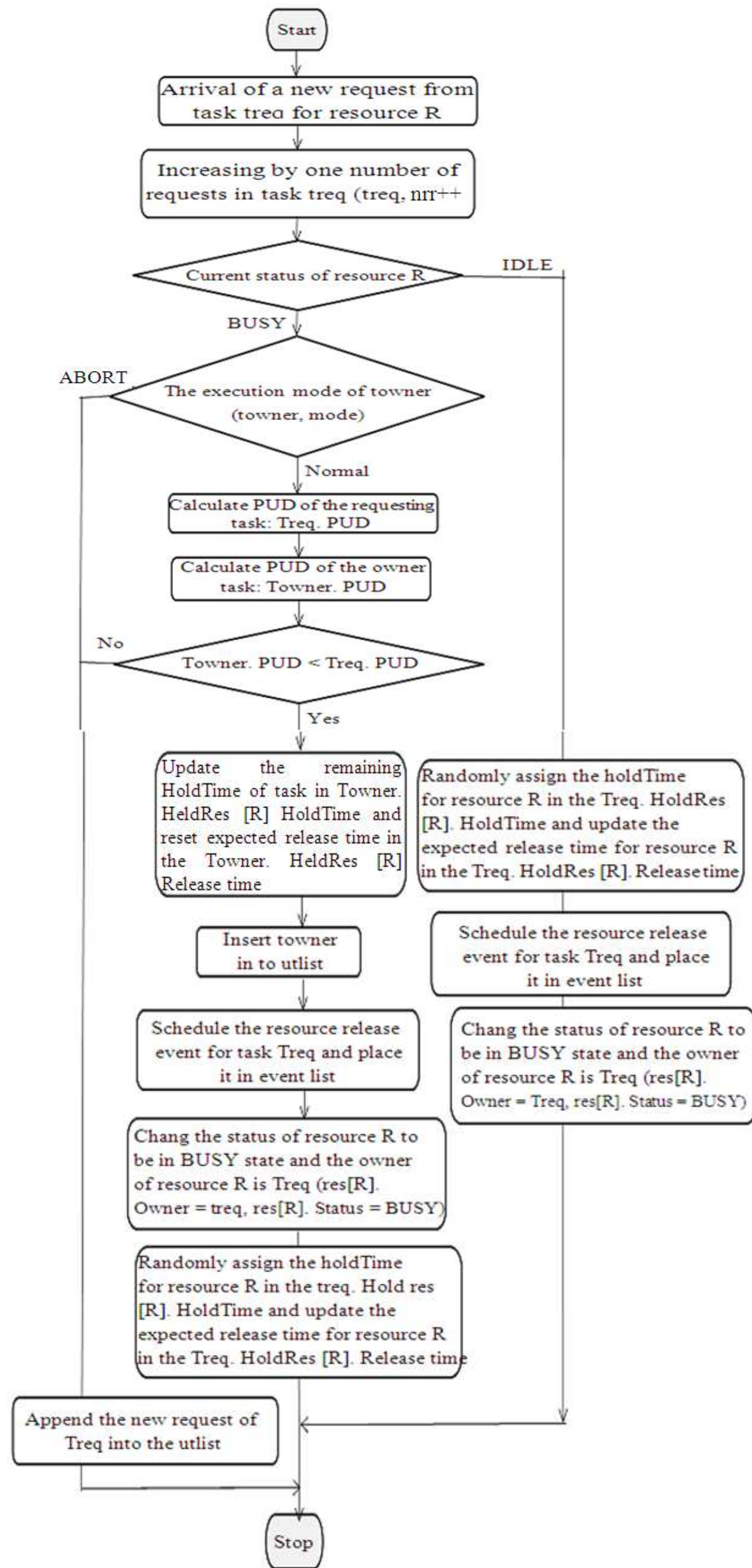

Fig. 5: A resource request event

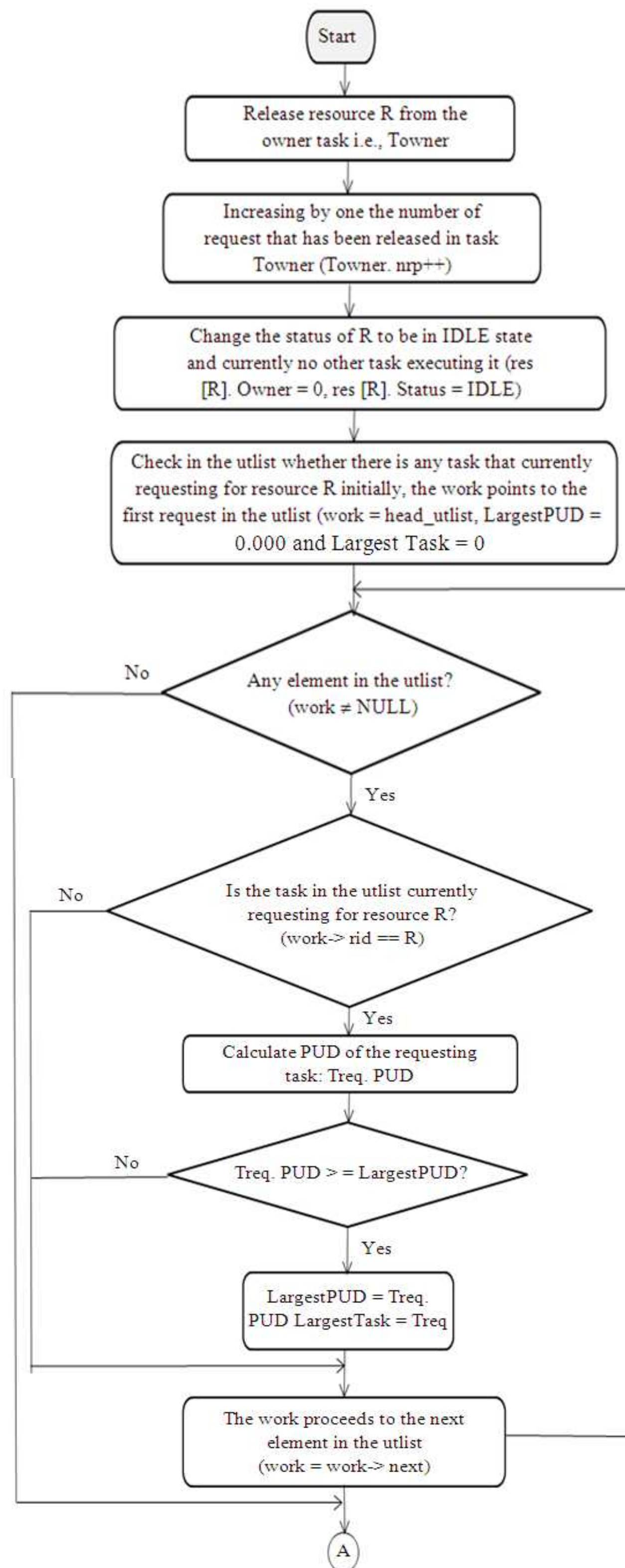

Fig. 6: A resource release event -Phase 1

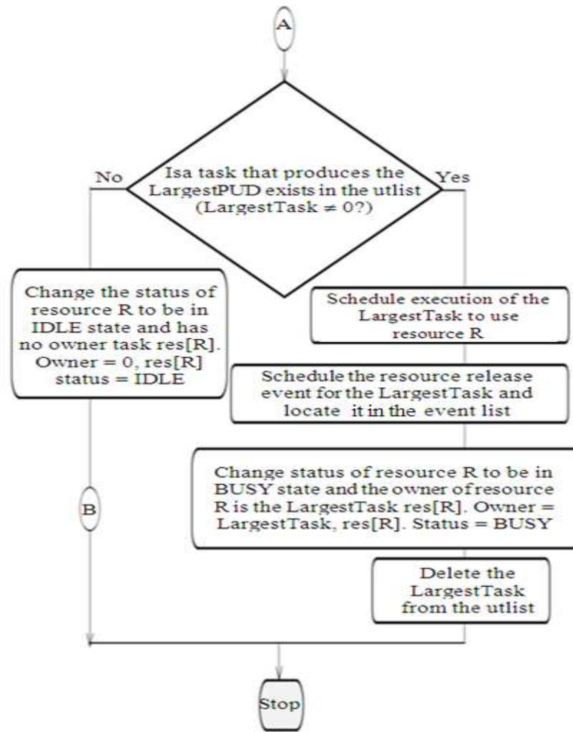

Fig. 7: A resource release event -Phase 2

The owner of resource R i.e., res[R].Owner is set to zero to indicate that R is currently not being held by any task. Next, the utlist is checked whether there is any task that currently requesting for resource R. The work pointer is used for the searching procedure. The work pointer checks the resource rid of the first element in the utlist. If it does not discover the resource R, it will proceed searching to the next element. If the work pointer discovers that a task i.e., *Treq* is currently requesting for resource R, the PUD of the requesting task i.e., *Treq*. PUD is calculated. The PUD is then compared with the *LargestPUD* parameter that contains the value that is currently producing highest PUD among the tasks in *utlist*.Initially, the value of largest PUD is set to 0.0000. If task *Treq* produces. A larger PUD than the value currently in *LargestPUD*, the *Treq.PUD* is considered as the highest PUD so far. Thus, the value of *LargestPUD* is updated to be equal to the *Treq*.PUD.

Phase 2: The scheduling for the execution of the task possessing the largest PUD in the *utlist* (if any). Figure 6 shows the flowchart of the second phase. The completion of Phase 1, will be followed by the determination of possible value for the *LargestTask* parameter .1. In the event that there is a request, the status of resource R is changed to the BUSY state and the *LargestTask* become the owner of resource R. The *LargestTask* consumed resource R instantaneously and the resource release event is scheduled in the event list. Since the *LargestTask* will be holding resource R, it is deleted from the *utlist*. If none of the tasks in the system currently requesting for resource R, it remain in the IDLE state without any owner task.

Experimental setting: The developed simulator has been tailored to map the characteristics of a uniprocessor scheduling. 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 1-10. The different values 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 M/M/1 queuing model is used to estimate the overloaded situations in the uniprocessor environment. In this research, the range values of load are observed 1.0-10.0 in the multiprocessor environment following the estimation load in the M/M/C queuing model.

The M/M/C queuing model is used to indicate a multiserver system with C servers that have unlimited queue capacity and an infinite population of potential task arrivals. The number of processors in the system is considered in the 2, 4 and 8 core platforms (Dellinger *et al*., 2011). Generally, the inter arrival times denoted by λ and the service times per server denoted by μ are exponentially distributed. 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. The inter arrival time, denoted as λ is defined in the unit of tasks/secsmeasures the number of tasks that arrived into the system in one sec. The service rate per processor denoted as μ measures the number of tasks that is being processed by each processor within one sec.

For multiprocessor, the maximum service rate for all processors is equal to $C\mu$. Note that inters 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., $\lambda < C\mu$.

Table 1: Parameter Estimation in the M/M/C queuing model

Number of processor C	Parameter (Cμ)	Parameter ρ ($\rho = \lambda/\mu < C$)
1	2	$\rho < 1$
2	4	$\rho < 2$
4	8	$\rho < 3$
8	16	$\rho < 4$

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 1.

From the literature, the system is considered to be stable when the arrival rate λ is less than the maximum service rate cμ i.e., $\lambda < C\mu$. 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≤ 10. For every number of MAX_CPU, the system is started to be overloaded starting when load = MAX_CPU. Referring to Table 1, 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 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. The Accrued Utility Ratio (AUR) metric defined in (Li *et al*., 2006) has been extensively utilized in the existing TUF/UA scheduling algorithms as performance metric. AUR is defined as the ratio of accrued aggregate utility to the maximum possibly attained utility.

RESULTS AND DISCUSSION

Figure 8 depicts the AUR result under an increasing load for step TUF. From the results, as the number of load is increased; a lower accrued utility is recorded. From the overall results, as the load and the number of processors increase, the higher utility accrued to the system by the PPUAS algorithm as compared to PUAS algorithm that is executed in the uniprocessor environment. The enhancement of the uniprocessor scheduling to the multiprocessor scheduling environment has tremendously improved the utility accrued to the system. The multiprocessors acquired a larger number of resources that can be used by the executed tasks.

Referring to Fig. 8, in dual core platform, the average load of 2 is estimated as the starting point of the overloaded situations in the system. At this load, PPUAS2 has achieved 75.99% and PUAS with 63.25% of the accumulated utilities. At this load, PPUAS2 algorithm has improved PUAS for 12.74% of the accumulated utilities. The superiority of PPUAS2 is also shown for the entire load range.

In four core platform, approximately the system is considered to be overloaded when the average load is equal to 4. At this load, PPUAS4 has successfully gained 75.43% of utility and PPUAS2 moderately accrued 58.00% while PUAS accrued 36.35% of the utilities. Thus, the PPUAS4 algorithm outperforms PPUAS2 for 17.43% and PUAS for 39.08% at this load. As the load increases, more incoming tasks arrived into the system and requesting for the resources in the system. PPUAS4 acquired a larger number of processors and resources to be used by the executed tasks as compared to the PPUAS2. Due to the limited resources, more tasks in PPUAS2 are overdue and therefore ending up being aborted. More aborted tasks are produced as the load increases and consequently produced more zero utility tasks to the system. This is why a sharper degradation is observed as the load increases for PPUAS2 in the dual core platform.

The PPUAS8 that runs in the eight core platform has produced the highest utility to the system as compared to the dual and quad core platforms. In eight core platform, approximately the system is considered to be overloaded when the average load is equal to 8. At this load, PPUAS8 has successfully gained 79.35% of utility, PPUAS4 achieved 61.62%, PPUAS2 moderately accrued 36.46% while PUAS accrued 24.71% of the utilities. Thus, the PPUAS8 algorithm outperforms PPUAS4 for 17.73% and PUAS2 for 42.89% at this load. The PPUAS8 algorithm outperforms the PUAS algorithm for 54.64% at this load.

Figure 9 plots the task success ratio experienced as a function of the increasing loads. Figure 9 complements the AUR results deliberated in Fig. 8. This is because it measures the exact number of tasks that has successfully contributed to AUR.

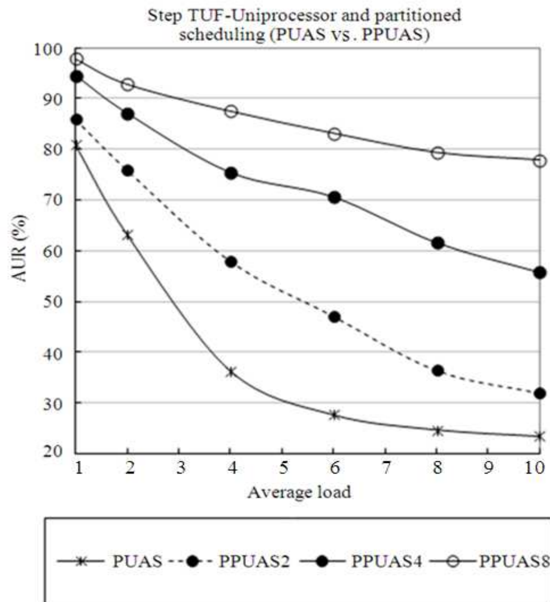

Fig. 8: AUR Results for step TUF

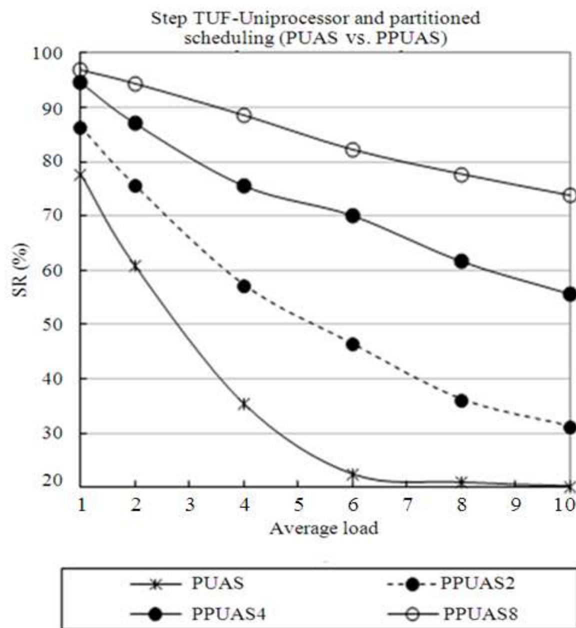

Fig. 9: SR Results for step TUF

Further, the result in Fig. 9 verifies that the reason of PPUAS acquired a higher utility compared to PPUAS is specifically because of the increases on the number of tasks that has successfully contributed to utility of the system. As the number of processor increases, a higher utility accrued to the system by the PPUAS algorithm in partitioned scheduling environment.

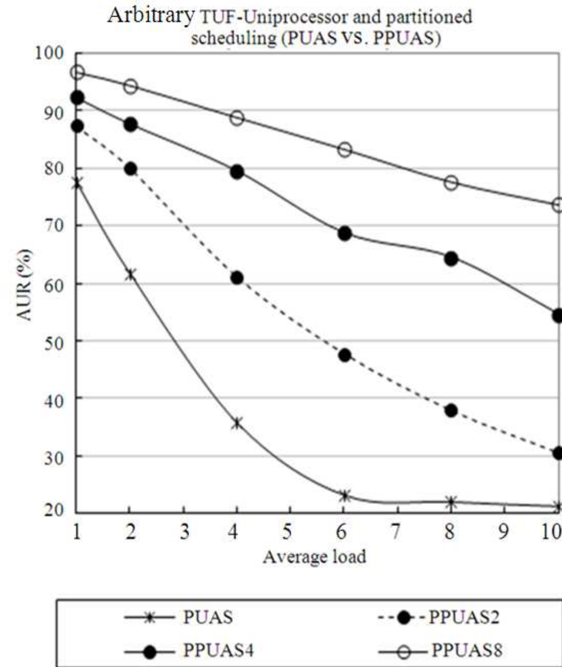

Fig. 10: AUR Results for arbitrary TUF

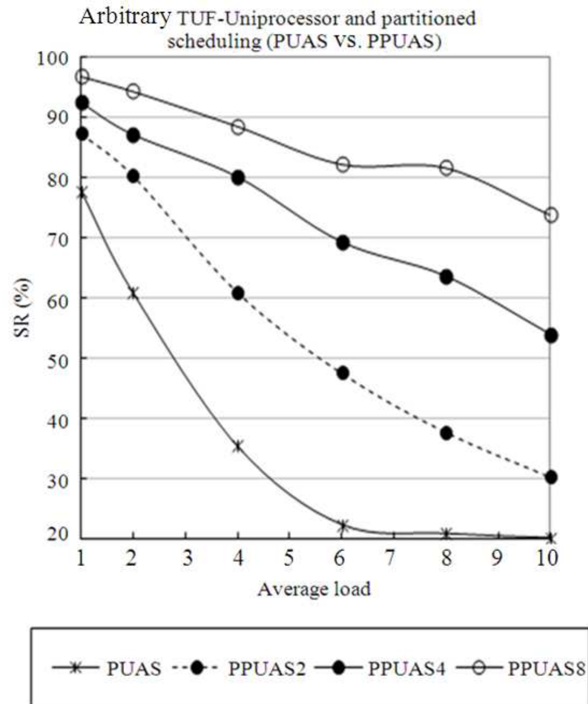

Fig. 11: SR Results for arbitrary TUF

This is because larger resources can be used by the executed tasks as the number of processor increases in the system.

Figure10 depicts the AUR results for execution of the arbitrary TUF tasks in the system. The nature of the curves indicates clearly that PPUAS has produced the highest utility accrued to the system as compared to the PUAS algorithm for the entire load range in all the number of processors involved.

Overall, the patterns of the curves from the results in the arbitrary TUF tasks set is similar to the step TUF tasks set. In the case of arbitrary TUF, a task may not be able to accrue its maximal possible utility even though the execution is completed before its termination time. Although these algorithms guarantee that the highest PUD task to be selected, it does not necessarily represent that the maximum possible utility gained by the executed tasks.

Referring to Fig. 10, the average load of 2 is estimated as the starting point of the overloaded situation in the system in the dual core platform. At this load, PPUAS2 has achieved 80.09% and PUAS with 61.81% of the accumulated utilities. At this load, PPUAS2 algorithm has improved PUAS for 18.28% of the accumulated utilities. The superiority of PPUAS2 is also shown for the entire load range.

Referring to Fig. 10, in four core platform, approximately the system is considered to be overloaded when the average load is equal to 4. At this load, PPUAS4 has successfully gained 79.55% of utility and PPUAS2 moderately accrued 61.17% while PUAS accrued 35.82% of the utilities. Thus, the PPUAS4 algorithm outperforms PPUAS2 for 18.38% and PUAS for 43.73% at this load. As the load increases, more incoming tasks arrived into the system and requesting for the resources in the system. PPUAS4 acquired a larger number of processors and resources to be used by the executed tasks as compared to the PPUAS2. Due to the limited resources, more tasks in PPUAS2 are overdue and therefore ending up being aborted. More aborted tasks are produced as the load increases and consequently produced more zero utility tasks to the system. This is why a sharper degradation is observed as the load increases for PPUAS2 in the dual core platform.

Referring to Fig. 10, the PPUAS8 that runs in the eight core platform has produced the highest utility to the system as compared to the dual and quad core platforms. In eight core platform, approximately the system is considered to be overloaded when the average load is equal to 8. At this load, PPUAS8 has successfully gained 77.63% of utility, PPUAS4 achieved 64.51%, PPUAS2 moderately accrued 37.92% while PUAS accrued 22.04% of the utilities. Thus, the PPUAS8 algorithm outperforms PPUAS4 for 13.12% and PUAS2 for 40.43% at this load. The PPUAS8 algorithm outperforms the PUAS algorithm for 55.59% at this load.

Figure 11 plots the task success ratio experienced as a function of the increasing loads for arbitrary TUF task set. Figure 11 complements the AUR results deliberated in Fig. 10.

CONCLUSION

With the obtained results, this study has proven that the proposed PPUAS partitioned multiprocessor scheduling algorithm have tremendously outperformed the uniprocessor scheduling algorithm i.e., PUAS in the highly overloaded situations. Overall, the PPUAS accrued the highest utility to the system due to the highest resource consumption by employing multiprocessor environment in dual, quad and eight core platforms. The contribution of PPUAS algorithm that achieved the highest accrued utility and success ratio making it suitable and efficient scheduling algorithm for real time application.

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

- The PUAS algorithm can be deployed in the global multiprocessor scheduling environment considering the migration attribute of the executed tasks
- The implementation of the fault tolerance in the TUF/UA partitioned multiprocessor scheduling environment

ACKNOWLEGMENT

This research is supported by Research University Grant (RUGS) 05-05-10-1115RU (Ministry of Higher Education, Malaysia) The authors are also grateful to Universiti Putra Malaysia for providing the excellent research facilities.

REFERENCES

Dellinger, H., P. Garyali and B. Ravindran, 2011. ChronOS Linux: A best-effort real-time multiprocessor Linux kernel. Proceedings of the 48th Design Automation Conference, Jun. 5-10, ACM Press, USA, pp: 474-479. DOI: 10.1145/2024724.2024836

Idawaty, A., S. Shamala, O. Mohamed and Z. Zuriati, 2011. A discrete event simulation framework for utility accrual scheduling algorithm in uniprocessor environment. J. Comput. Sci., 7: 1133-1140. 10.3844/jcssp.2011.1133.1140

Ji, L., G. Ruifeng and S. Zhixiang, 2010. The research of scheduling algorithms in real-time system. Proceedings of the International Conference on Computer and Communication Technologies in Agriculture Engineeeing, Jun. 12-13, IEEE Xplore Press, USA., pp: 333-336.

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. Computer., 55: 454-469. DOI: 10.1109/TC.2006.47