Efficient Processing Power Reservation Approach to Imporove Real-Time Task Schedulability and Relaibaility

Ahmed E. Youssef[1] Abeer Hamdy[2] and Reda Ammar[3]

**[1] Faculty of Engineering-Helwan, Helwan University
Cairo, Egypt, and
Dept. of Information Systems, King Saud University
Riyadh, 11543, KSA**

**[2] Dept. of Computers and Systems, Electronic Research Institute, and
Faculty of informatics and computer science, British University in Egypt
Egypt**

**[3]Dep. of Computer Science and Engineering, University of Connecticut
Storrs, CT 06269-3155, USA**

Abstract

Efficient utilization of the computational resources is an urgent demand especially if real time constraints should be guaranteed. Moreover, an acceptable level of reliability should be provided due to the critical nature of some real time applications. This paper proposes a new approach for processing power reservations that efficiently utilizes *all* the available processing power to improve reliability and schedulability of independent real time tasks on a uni-processor. The basic idea of the proposed approach is to use all of the available processing power in the time interval between the arrivals of two successive tasks or when an existing task departs. The advantages of this mechanism are: 1) it reduces the execution time required for each task and hence increases its reliability. 2) At the arrival of a new task; the processing power requirements for the executing tasks to meet their deadlines are smaller, which gives the new arriving tasks higher chances to be accommodated with the existing ones. 3) Efficient processing power utilization may reduce the power consumption in processors with dynamic voltage scaling. An illustration example and simulation experiments showed that our approach provides a more reliable scheduling scheme with higher acceptance rate compared to the traditional approach based on Rialto operating system.

Keywords-*Processing Power, Processor Utilization, Scheduling, Real-Time Tasks.*

1. Introduction

In recent decades, real-time systems have been employed in many application domains including banking systems, autonomous robots and control systems. A real time application is composed of one or more tasks that are dependent in most cases and are required to perform their functions under strict timing constraints. A task missing its deadline may result in a dominant effect, causing other tasks to miss their deadlines which may cause a system failure. Emerging computing paradigms, cloud, grid, cluster and multi/many core systems provide suitable platforms for real time systems to satisfy their timing requirements. Each of these computing paradigms requires a middleware called scheduler to manage the allocation of the computing resources to the admitted applications in such a way that certain performance metrics are met. These metrics depend on the application areas and are used to guide the scheduling decisions. However, in real-time systems, the main metric is to satisfy timing constraints with maximum acceptance rate.

Scheduling real time tasks on multiprocessor and distributed platforms is usually achieved using a two-level hierarchical scheduler: 1) A high level scheduler (partition algorithm) which is concerned with how to partition the applications and assign them to the different processors. 2) Low level scheduler (CPU reservation algorithm) that ensures an efficient and predictable scheduling of real-time independent tasks on each processor individually. By and large, in real-time computing, a task is submitted along with a statement of its start, finish and computation times. Depending on the available processing power, the scheduler either admits the task, guaranteeing the task will be completed on time, or rejects the request if it is impossible to satisfy the desired deadline of the task. Thus, in order to accommodate as many tasks of different applications as possible while satisfying the required deadline of each application an efficient utilization of the CPU processing power is necessary.

In this paper we introduce a new approach for processing power reservation that improves real-time task scheduling in terms of both acceptance rate and reliability. Our approach utilizes all of the processing power in the execution mode. Thereby; 1) the processing power requirements for the current loaded tasks are smaller when new tasks arrive. This gives higher chances for the new arriving tasks to be accommodated with the existing ones. 2) The execution times of the admitted tasks are reduced and consequently the system reliability is increased. 3) Power consumption is reduced in processors with dynamic voltage scaling.

The rest of the paper is organized as follows: Section 2 reviews some work related to real-time task scheduling. Section 3 introduces Rialto processing power reservation approach which we used as a baseline to evaluate the performance of our proposed approach. Section 4 describes our proposed scheduling algorithm. Section 5 illustrates our algorithm by an example and shows its advantages by comparing it to Rialto approach. Sections 6 discuss the simulation experiments and results. Finally, section 7 concludes the paper.

2. Related Work

There has been and continues to be a great deal of research that addresses the problem of scheduling real time tasks. In a broad sense, scheduling approaches can be classified in several ways. For example, they can be classified based on the computing platform; scheduling algorithms in [15, 22] address the problem of task allocation over a Grid; the algorithms in [3,13,18,19,20,21] address the problem of task allocation over a cluster; the algorithms in [2,9,10,11,12,14,16,17] address the problem of allocating tasks over the processors of multiprocessor and multi-core systems; while the algorithms in [1, 5, 6, 7, 8, 23] have been proposed to ensure an efficient and predictable scheduling of real-time independent tasks over a uni-processor. Another classification to the scheduling approaches could be based on the additional performance metrics along with satisfying timing requirements (e.g. minimizing completion time, maximizing throughput, reducing power consumption); The algorithms in [2,4,15,16,21] have been proposed for reducing power consumption in processors with dynamic voltage scaling; while the algorithms in [19, 20] are concerned with achieving effective fault-tolerant in real time systems.

Most real-time scheduling approaches on a uni-processor focus on providing proportional share CPU allocation. A task receives a CPU share that corresponds to a user specified weight or percentage. This CPU share is called Processing Power (PP). A common representative to this scheduling approach is the Rialto operating system that is developed by Microsoft research [7,8]. Rialto is designed to schedule multiple independent real time and non real time tasks using the CPU reservations on the same processor. The efficiency of CPU reservation is a result of a moderate overhead that is incurred by the CPU scheduling. The overhead is bounded by a constant and is not a function of the schedulable tasks. Also, scheduled task cannot violate other tasks' guarantees. The following section discusses the Rialto CPU reservation approach which we used as a baseline to evaluate the performance of our proposed approach.

3. Rialto CPU Reservation Approach

Rialto can schedule multiple independent tasks on a uni-processor using a CPU reservation mechanism. Processing power reservations are made by the tasks to ensure a minimum guaranteed execution rate. Request for processing power reservation is of the form reserve α% processing power out of β% ($\beta \leq 100$) available processing power at processor P_m for a certain time (task deadline). This is equivalent to reserving x time units out of every y units for the task. Based on this proposition, each processor maintains a data structure, called reservation table, of a pre-computed schedule. Each entry in the reservation table contains information such as task ID T_j, required processing power PP_j for each task, expected starting time S_j and expected completion time F_j. Table1 shows an example of the reservation table for a specific processor that accommodates tasks T_1, T_2, T_3, and T_4 and figure 1 shows the Processing Power Reservation Graph (PPRG) of the processor over the time interval {115,211}.

Table 1: Reservation table for a process

T_j	PP_j	S_j	F_j
T_1	0.20	115	130
T_2	0.10	124	142
T_3	0.30	133	159
T_4	0.40	157	211

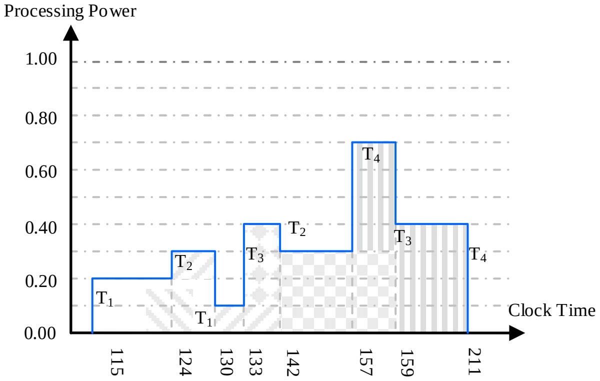

Fig.1: PPRG of a processor that accommodates tasks T_1, T_2, T_3, T_4 over the time interval 115:211.

The negotiation and reservation activities are made possible using the reservation table, when a task is admitted the minimum available processing power on the processor during the deadline of the task is computed using the reservation table or the PPRG of the processor. If the required processing power for the task is less than or equal to the minimum available processing power, the task is accepted and a corresponding entry is added to the table and automatically the PPRG is updated, otherwise the task is rejected meaning that it cannot be accommodated by the processor. On the other hand when a task finished, its entry is deleted from the reservation table and its processing power is released allowing the processor to accommodate new tasks. Algorithm1 describes the Rialto approach for real time task scheduling.

Algorithm 1: Rialto Approach
Input: a set of real-time independent tasks

$$\mathbf{T} = \{T_1, T_2, ..., T_n\}$$

Output: acceptance rate
Begin
1. counter = 0
2. For each $T_j \in \mathbf{T}$

Determine $PP_{j-\min-ava} = \min\left[PP_{ava}(t)\right]_{t=S_j}^{F_j}$

If ($PP_j \le PP_{j-\min-ava}$)) then

Increment counter

Update $PPRG$ in the window $[S_j, F_j]$ as:

$$PP_{res}(t) = PP_{res}(t) + PP_j, t \in [S_j, F_j]$$

3. Acceptance rate = counter/n
End

4. The Proposed Approach

In this section we present our new algorithm for processing power reservation and utilization. The input to this algorithm is a set of tasks $\mathbf{T} = \{T_1, T_2, ..., T_n\}$ Each task T_j is defined using three parameters (S_j,F_j,PP_j). The algorithm has to determine the acceptance/rejection status of the tasks, and updates the PPRG. The basic idea of the algorithm is to use all of the available processing power (PP_{ava}) in the time interval between the arrivals of two successive tasks or when an existing task departs. When a task departs, its processing power is released back and re-distributed among the remaining active tasks. During the scheduling process the processor alternates between two modes; Execution Mode (EM) and Reservation Mode (RM).

Execution Mode (EM): is activated between two successive arrivals or at the departure of an executing task. In this mode, all the available processing power is distributed among the allocated tasks proportional to their workloads WL_j. Consequently, each task is allocated with at least its reserved processing power. If a task receives a higher processing power, it terminates earlier before its deadline. Otherwise, it terminates exactly at its desired deadline.

Reservation Mode (RM): is activated at the arrival of a new task. Each of the existing tasks is assigned part of the available processing power depending on its remaining workload, guarantee no deadline violation. The remaining processing power becomes available for the new task. If it is enough to accommodate the new task, it is admitted. Otherwise, the new task is rejected.

The algorithm proceeds as follows:

a. Initialization:

1. At the arrival of the first task T_1: All the available processing power PP_{ava} (initially PP_{ava} =1) is assigned to it, instead of its required PP_1. So the task can terminates earlier than its required finish time (F_1).
2. The new finishing time (Z_1) is calculated which we will call lock time.
3. The task is added to a list called active_list (K).

b. Reservation Mode (RM): Processor converts from EM to RM at the arrival of a new task T_i. RM proceeds as follows:

1. For each existing task T_j in K compute remaining workload (WL_j) in the time interval [S_i, F_{j-EM}].

$$WL_j = PP_{j-EM} * (F_{j-EM} - S_i)$$

Where,
S_i: Task T_i arrival time,
PP_{j-EM} : is the new processing power assigned for T_j during execution mode ($PP_{j-EM} \geq PP_j$),
F_{j-EM} : Execution mode finish time of T_j, which is either equal to its required finish time F_j or smaller .

2. Compute the new reservation processing power PP_{j-RM} for each task T_j in K such that T_j finishes at its required finish time F_j)

$$PP_{j-RM} = \frac{WL_j}{F_j - S_i}$$

Where, $PP_{j-RM} \leq PP_j$

3. Use PP_{j-RM} for each task T_j in K to allocate T_j and update the PPRG in the time interval [S_i, F_j].

4. Compute the minimum available reservation processing power in the time interval [S_i, F_i]

$$\mathrm{PP}_{\mathrm{min-ava-RM}} = \min\left[PP_{ava}(t)\right]_{t=S_i}^{F_i}$$

$$PP_{ava}(t) = 1 - PP_{RM}(t)$$

$$PP_{RM}(t) = \sum_{j \in K} PP_{j-RM}(t)$$

5. If $PP_i \leq PP_{min-ava-RM}$, task T_i is accepted, added to the active list K and the PPRG is updated; Else T_i is rejected.

6. The processor converts from RM to EM.

c. Execution Mode (EM): It is activated after accepting a new task during the reservation mode or at the departure of an existing task to re-distribute the available processing power. EM proceeds as follows when accepting a new task T_i:

1. Compute the new execution mode processing power (PP_{j-EM}) for each task T_j in K as follows:

$$PP_{j-EM} = PP_{ava} * \frac{WL_j}{\sum_{i\in K} WL_i}$$

2. If $PP_{j-EM} \le PP_j$ ∀ $T_j \in$ K then
   Set $PP_{j-EM} = PP_j$,
   Move T_j from K to a temporary list L
   Set $PP_{ava} = PP_{ava} - PP_j$
   Go To step 1

3. Compute the lock time for the current execution mode $F_{j\text{-}EM}$ (N.B. All the tasks in the list K, have required finish times later than $F_{j\text{-}EM}$, will finish at the current execution mode lock time).

$$F_{j-EM} = S_i + \frac{\sum_{k\in K} WL_k}{PP_{ava}}$$

4. Move the tasks in the temporary list L to K.

5. Use $PP_{j\text{-}EM}$ for each T_j in K to plot PPRG in the time interval $[S_i, F_{j-EM}]$.

At the departure of a task T_i during EM, the previous steps will be repeated during the interval $[F_i, F_{j-EM}]$.

Algorithm 2 briefs our proposed approach.

Algorithm2: The proposed approach

Input: a set of real-time tasks $\boldsymbol{T} = \{T_1, T_2, ..., T_n\}$
Output: acceptance rate
Begin
1. counter = 0
2. Arrange T in an event (task arrival/departure) queue, Q
3. Get an event e from Q
4. While (Q is not empty)
  IF (e is an arrived task)
    * Convert $PPRG$ from EM to RM
    * Check acceptance of the arrived task
    If (task accepted)
      - Increment counter
      - Allocate task on $PPRG$
      - Convert $PPRG$ from RM to EM
  ELSE
    * Remove departed task from $PPRG$ in EM
    * Redistribute PP among remaining tasks
5. acceptance- rate = counter/n
End

5. An Illustration Example

This section presents an example that illustrates the idea of our proposed approach. The set of the arriving tasks are described in Table 2. Figure 2 shows that based on Rialto approach T_4 cannot be accommodated by the processor since its required processing (0.3) exceeds the minimum available processing power (0.1) in its time interval [50,100], i.e., $PP_4 > PP_{4\text{-}min\text{-}ava}$

The PPRGs for these tasks during the reservation mode and the execution mode using our approach are shown in Figs. 3-7. A quick inspection for these PPRGs revealed that while task T_4 is rejected using Rialto approach, all tasks including T_4 were accepted for scheduling using our approach for processing power utilization. This shows that by utilizing all the available processing power during the execution mode after T_4 arrives we reduce the PP requirements for T_1, T_2 and T_3 so that T_4 can be accommodated. Table 3 shows the acceptance/rejection status of each task using Rialto approach and using our approach. It can be noticed that using our approach, tasks T_3 and T_4 finish earlier than their desired deadlines.

Table 2: Tasks Reservation Table.

T_i	S_i	F_i	PP_i
T_1	10	60	0.3
T_2	20	70	0.2
T_3	25	80	0.4
T_4	50	100	0.3

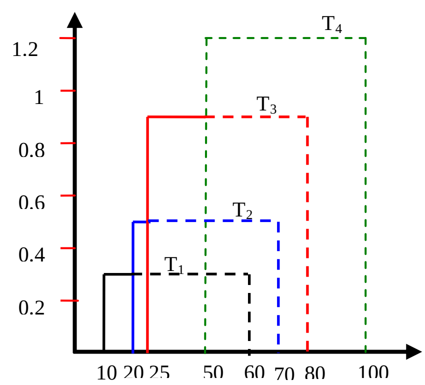

Fig.2: T_4 is rejected using Rialto approach

IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2, September 2012
ISSN (Online): 1694-0814
www.IJCSI.org

Table 3: Acceptance and rejection status of the tasks when using Rialto and using our approach

Task	Rialto approach		Our approach	
	Finish time	Acceptance status	Finish time	Acceptance status
T_1	60	accepted	60	Accepted
T_2	70	accepted	70	Accepted
T_3	80	accepted	71.876	Accepted
T_4	100	rejected	71.876	Accepted

The reliability of each task can be estimated using the following equation:

$$R = \exp[-\beta * t]$$

Where: t: execution time of T_j
β: failure rate of the processor

Table 4 shows the reliability of each task when executed on a processor of failure rate β = e-3. It can be noticed that an improvement of 1% and 3% in the reliabilities of the tasks T_3 and T_4 respectively when executed using our approach over their reliabilities when executed using Rialto approach. So, we conclude that our approach will also provide higher level of reliability due to the reduction in the execution time.

Table 4: Acceptance and rejection status of the tasks using Rialto approach and using our approach

T_j	Rialto approach		Our approach	
	Execution time	Reliability	Execution time	Reliability
T_1	50	0.951229425	50	0.951229425
T_2	50	0.951229425	50	0.951229425
T_3	55	0.946485148	46.876	0.954205712
T_4	50	0.951229425	21.876	0.978361544

6. Simulation Experiments

6.1. Experiments Setup

In order to show the performance of the proposed approach relative to Rialto, four simulation experiments were conducted. In each experiment, sets of tasks were generated according to the following settings:

1. Each set contains 10000 tasks generated randomly.
2. Different sets have different values of 1/λ however all sets in one experiment have the same value of 1/μ (μ is the departure rate of tasks). The values of 1/μ are 10, 15, 20, and 25 in the four experiments respectively. The value of 1/λ ranges between 1/μ and 100 sec.
3. In each set, a uniform probability distribution is used to generate random values for execution time of the tasks.
4. In each set, an exponential probability distribution is used to generate random values for inter-arrival time of the tasks.

The ratio (λ/μ) is called traffic intensity (it expresses processor utilization) and cannot exceed one since λ is always smaller than μ. If this ratio is close to one it means that tasks have relatively large λ (fast arrival). Consequently, their scheduling on the processor will be more difficult than if the ratio is close to zero (relatively small λ or slow arrival).

6.2. Performance Evaluation Criteria

The performance metric used in evaluating our scheduling approach is the *acceptance rate of the tasks on a processor*. *Acceptance rate* is defined as the ratio of the number of tasks that can be executed without violating their deadline requirements to the total number of tasks. We measure the acceptance rate at different values of mean inter-arrival time (1/λ) and mean execution time (1/μ) of the tasks (where λ and μ are the arrival and the departure rates of the tasks respectively).

6.3. Simulation Results

Figures 8-11 show the acceptance rate vs. traffic intensity during each of the four experiments. In all experiments, results show that the proposed scheduling approach outperform Rialto especially when the traffic is heavy. Results also show that the two approaches reject more tasks when tasks arrive faster than the processor can handle (large values of λ, heavy traffic). However, our proposed approach is still superior to Rialto. In contrast, the two algorithms perform competitively well when the tasks arrive far apart from each other (small values of λ, light traffic).

Figure 12 shows the percentage improvement in acceptance rate achieved by our proposed approach over Rialto approach at different values of 1/μ and 1/λ. As can be seen in the graphs, the improvement diminished as λ decreases. This is due to the fact that the two approaches perform very well for small values of λ (slow arrivals). The graphs also show that we achieve higher amount of improvement for higher values of λ (fast arrivals). Hence, we conclude that our approach has a major improvement when tasks arrive at high arrival rate. In this case, the scheduling process becomes difficult and Rialto-based approach performs poorly.

As can be seen, the maximum improvement is achieved at the largest value of 1/μ (slow departure) and the smallest value of 1/λ (fast arrival). These results show again that our approach has a major improvement when the processor is heavily loaded and where Rialto approach fails.

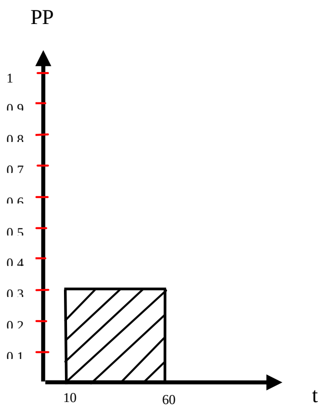

Fig.3-a: Reservation mode at the arrival of T_1

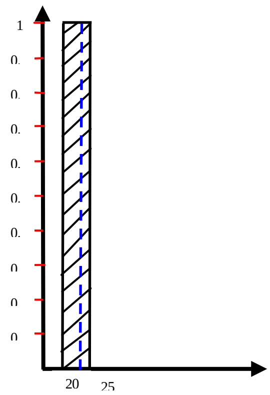

Fig.3-b: Execution mode between the arrivals of T_1 and T_2

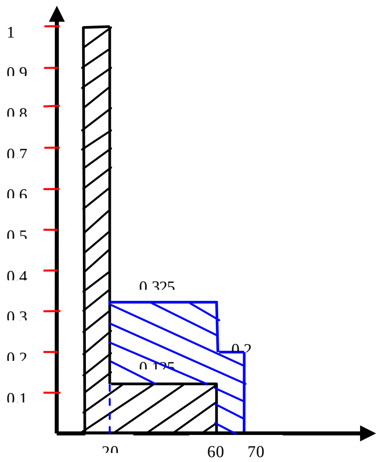

Fig.4-a: Reservation mode at the arrival of T_2

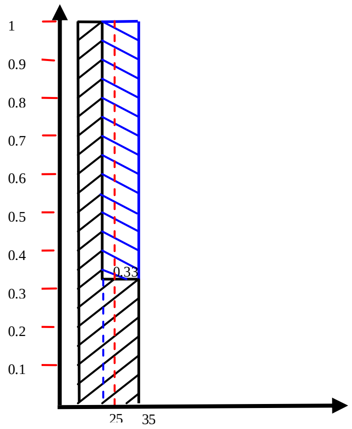

Fig.4-b: Execution mode between the arrivals of T_2 and T3

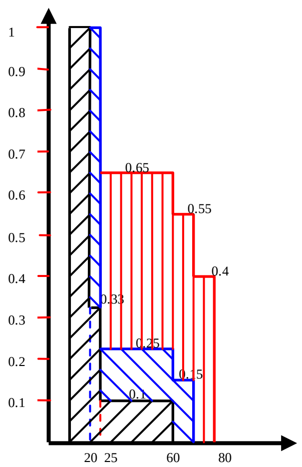

Fig.5-a: Reservation mode at the arrival of T_3

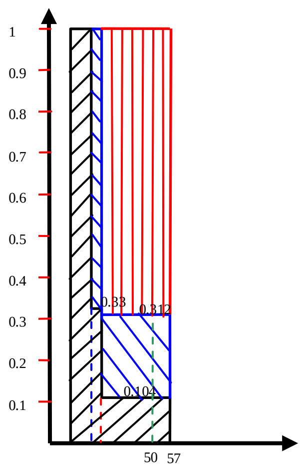

Fig.5-b: Execution mode between the arrivals of T_3 and T_4

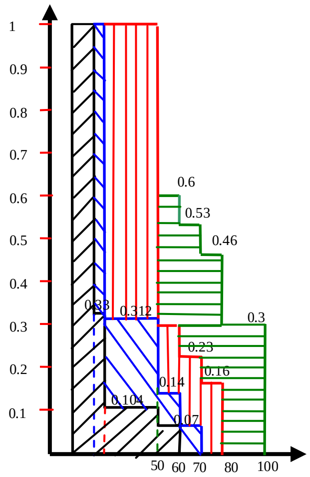

Fig.6-a: Reservation mode at the arrival of T_4

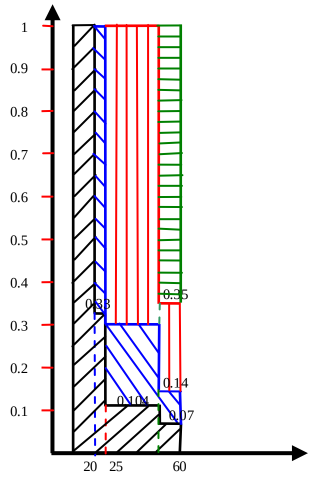

Fig.6-b: Execution mode between the arrival of T_4 and the departure of T_1

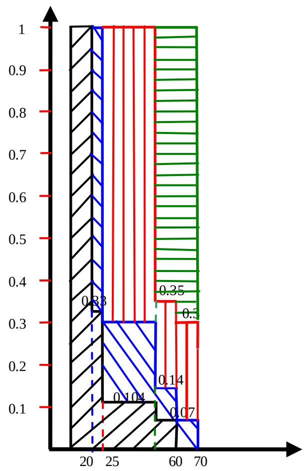

Fig.7-a: Execution mode between the departure of T_1 and the departure of T_2

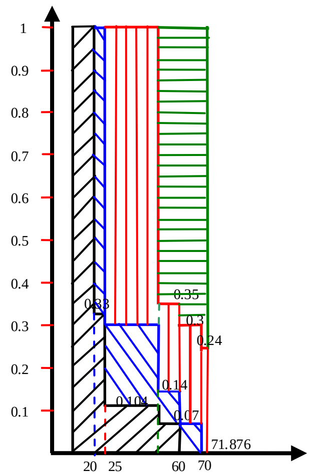

Fig.7-b: Execution mode after the departure of T_2

Fig. 8 Acceptance rate at mean execution time (1/μ) = 10 and different traffic intensity.

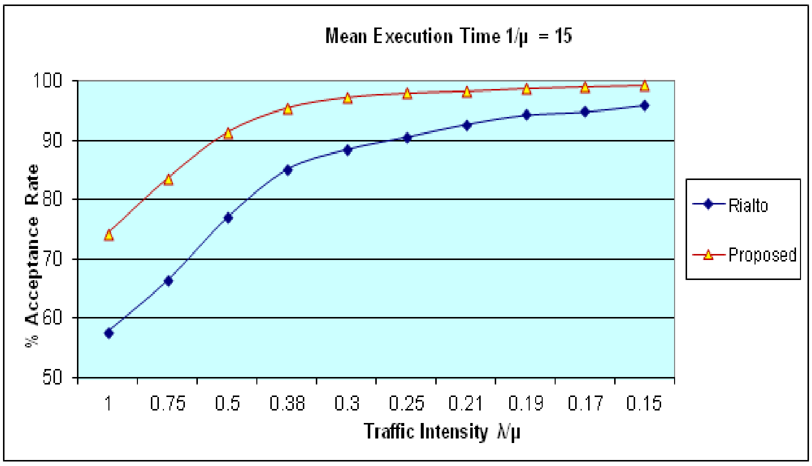

Fig. 9 Acceptance rate at mean execution time (1/μ) = 15 and different traffic intensity.

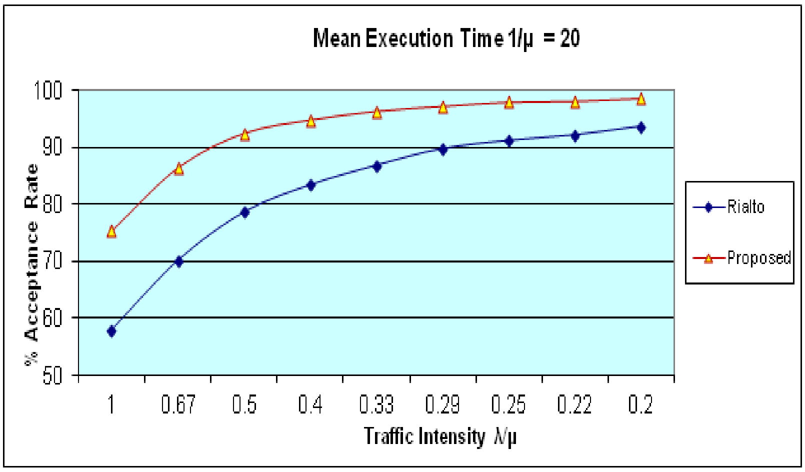

Fig. 10 Acceptance rate at mean execution time (1/μ) = 20 and different traffic intensity.

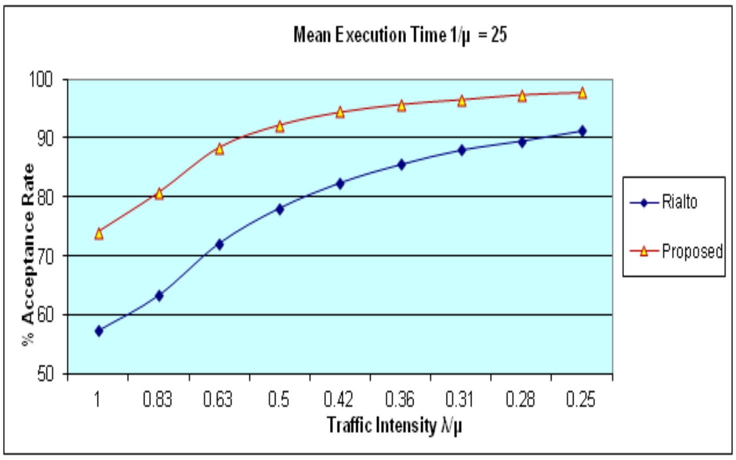

Fig. 11 Acceptance rate at mean execution time (1/μ) = 25 and different traffic intensity.

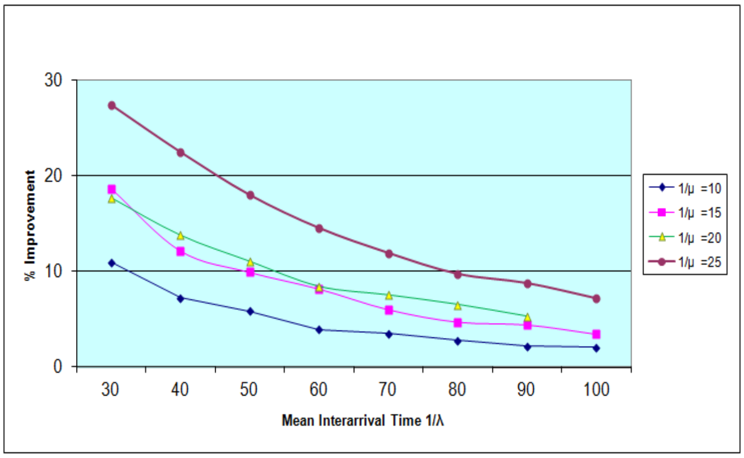

Fig.12 Improvement in the acceptance rate at different values of 1/μ and 1/λ.

7. Conclusions

This paper presented a new approach for processing power reservations that efficiently utilizes the processor and improves the schedulability of real-time independent tasks on a uni-processor. In addition it improves the reliability of the tasks by reducing their execution times. We compared our approach to a traditional one ,called *Rialto,* and it is found that our approach is superior in terms of both acceptance rate and reliability. Moreover, we expect that our approach may help in reducing the power consumption in processors with dynamic voltage scaling. As Aydin [4] and Yang [2] mentioned that optimal solution for energy efficient scheduling of periodic real time tasks; when they are executed at constant speed such that utilization is 100% or at minimum speed with utilization less than 100%. We are currently investigating energy saving issue when using our proposed approach.

References

[1] B. Ford and S. Susarla, "CPU inheritance scheduling", operating systems review, vol. 30, 1996, pp. 91-105.

[2] Chuan-Yue Yang, Jian-Jia Chen ; Tei-Wei Kuo ; Lothar Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems", In Proceedings of Design, Automation &test in europe conference &exhibition, Date'09, 2009, pp. 694 - 699

[3] G. Birkenheuer and A. Brinkmann. Reservation based overbooking for HPC clusters. 2011 IEEE International Conference on Cluster computing, 2011.

[4] H. Aydin, R. Melhem, D. Mosse and P.Meja-Alvarez, "Dynamic and Aggressive Scheduling techniques for power-aware real-time systems", In Proceedings of the 22nd IEEE Real-Time systems Symposium, 2001, pp. 95-105.

[5] I. Stoica, H. Abdelwahab, K. Effay, S.K. Baruah, J.E. Gehrke, and C.G. Plaxton, "A proportional share resource allocation algorithm for real-time, time-shared systems", presented at proceedings in 17th IEEE real-time systems symposium, Los Alamitos, CA, USA, 1996.

[6] J. Regehr, J. and J.A. Stankovic. Augmented CPU reservations: Towards predictable execution on general-purpose operating systems. in Real-Time Technology and Applications – Proceedings,Taipei, Taiwan,2001.

[7] Michael B. Jones, Daniela Roşu, Marcel-Cătălin Roşu, CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proceedings* of the 16th ACM Symposium on Operating System *Principles*, St-Malo, France,Oct. 1997, pp. 198-211.

[8] Michael B. Jones, Two case studies in predictable application scheduling using Rialto/NT , In Proc. of the 7th Real-Time Technology and Applications Symposium (RTAS 2001), 2001

[9] Michle Lombardi, Michela Milano, Luca Benini, "Robust Scheduling of Task Graphs under Execution Time Uncertainty", IEEE transactions on computers, 2011

[10] Martin Niemeier, Andreas Wiese,, Sanjoy Baruah, Partitioned real-time scheduling on heterogeneous shared-memory multiprocessors, Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011) , 2011.

[11] M. Hamza, S.M. Fakhraie and C.Lucas "Soft Real Time Fuzz Task Scheduling for Multiprocessor Systems " World Academy of Science, Engineering and Technology, 2007.

[12] N. R. Satish, K.Ravindran, K. Keutzer, Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors, In EMSOFT '08 Proceedings of the 8th ACM international conference on Embedded software, 2008, pp. 149-158.

[13] R. Ammar, A. Alhamdan, "Scheduling real-time fork-join structures in cluster computing", Int. J. of High Performance Computing and Networking , Vol.3, No.4, 2005, pp.262 – 271.

[14] R. Gioiosa, S. A. McKee, M. Valero, Designing OS for HPC Applications: Scheduling, 2010 IEEE International conference on cluster computing, 2010.

[15] S.Baskaran, P. Thambidurai, Energy efficient real-time scheduling in distributed systems, IJCSI International journal of computer science issues, vol.7, issue 3, No.4, May 2010.

[16] W. Y. Lee, Energy-Efficient Scheduling of Periodic Real-Time Taks on Lightly Loaded Multicore Processors, IEEE Transactions on Parallel and Distributed Systems, Vol. 23, No. 3, March 2012.

[17] W.Y. Lee, S.J. Hong, J. Kim, On-line scheduling of scalable real-time tasks on multiprocessor systems, Journal of Parallel and Distributed Computing vol.63, No.12, 2003,pp. 1315_1324

[18] Xuan Lin, Anwar Mamat, Ying Lu_, Jitender Deogun, Steve Goddard , Real-time scheduling of divisible loads in cluster computing environmentsI, J. Parallel Distrib. Comput.Elsevier, 2010, pp. 296_308

[19] Xiaomin Zhu, Xiao Qin, Meikang Qiu, "QoS- Aware fault-Tolerant Scheduling for Real Time Tasks on Heterogeneous clusters" IEEE transactions on Computers,Volume 60, Issue 6,June 2011, pp.800-812.

[20] Xiaomin Zhu,Jianghan Zhu,Manhao Ma, Dishan Qiu, "SAQA: A self adaptive QoS-aware Scheduling Algorithms for Real Time Tasks on Heterogeneous Clusters" Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp.224-232.

[21] Xiaomin Zhu, Chuan He, Kenli Li, Xiao Kin, "Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters" , Journal of parallel and distributed computing, volume 72, Issue 6,2012, pp. 751-763.

[22] Yves Caniou, Ghislain Charrier, Frederic Desprez "Analysis of tasks reallocation in a dedicated grid environment" 2010 IEEE international conference on cluster computing, crete,2010.

Z. Deng, J.W.-s Liu, and S. Sun, "Dynamic scheduling of hard real-time applications in open system environment" presented at the real-time systems symposium, Washington, DC, 1996.