Real-Time Workload Allocation on a Uni-processor

Abeer Hamdy
Dep. of Computers and systems,
Electronic Research Institute &
Faculty of ICS, British University in Egypt
Egypt

Ahmed E. Youssef
Faculty of Engineering-Helwan
Helwan University, Cairo, Egypt
& Dept. of Information Systems
KSU, Riyadh, KSA

Reda Ammar
Dept. of Computer Science
and Engineering
University of Connecticut
Storrs. CT. USA

ABSTRACT

The paper presents a novel allocation algorithm to allocate independent real time tasks on a processor in a way that improves the processor's throughput (Processor's throughput is the number of tasks the processor can accept for execution). The proposed approach allocates tasks' workloads (task's workload is the percentage of work required by the processor to execute the task) instead of their processing powers (Processing power assigned to a task is a percentage of the processor reserved to execute the task such that its deadline is satisfied). To achieve our objective a variable processing power is assigned to the task under consideration over its deadline to satisfy its timing requirements instead of rejecting it if a constant processing power cannot be guaranteed as in previous CPU reservation approaches. Simulation results revealed that the acceptance rate of the admitted tasks to a certain processor using the new approach is superior to that achieved using the traditional processing power reservation approach.

Keywords

Workload allocation**,** Processing power, Processor utilization, Scheduling real-time tasks.

1. INTRODUCTION

Deploying applications on multiprocessor and distributed platforms require the mapping and allocation of the application's tasks to the different computing resources of the platform such that pre-set objectives should be met. This problem is known as *scheduling problem*. Various studies have proven that finding an optimal schedule is an NP-complete problem [17]. However, a large number of scheduling algorithms which attempt to find a suboptimal solution have been proposed. These algorithms can be categorized mainly into two groups based on the type of scheduled applications. The first category deals with non real time applications and the second deals with real time applications. The main concern in scheduling non real time applications is to minimize the time required to execute all the application's tasks (makespan). While, the main concern in scheduling real time applications is satisfying the timing constraints of each task. Hence, scheduling real-time applications is more challenging. Scheduling real time applications on multiprocessor and distributed platforms is achieved using a two-level hierarchical scheduler: 1) A high level scheduler (partitioning algorithm) which is concerned with how to partition the applications and assign their tasks to the different processors. 2) Low level scheduler (CPU reservation algorithm) that determines the execution order of real-time independent tasks on each processor individually. The overall performance of the scheduler depends on the performance of its two components.

This paper proposes a new approach for processor reservation that improves the utilization of the processor and increases its throughput. In the previous approach for processing power reservation [8],[9],[13], when a task is submitted, the scheduler accepts the task if the available processing power (PP) during the task's deadline is sufficient to satisfy its deadline requirements (i.e, the min. available PP is at least equals the required PP for the task to satisfy its deadline). Otherwise the task and hence the whole application are rejected. Our new approach depends on allocating the workload of the task on the processor by assigning the task a variable processing power that guarantees its timing requirements instead of attempting reserve a constant processing power over the time spent by the task in the processor and rejecting the task if this PP cannot be provided by the processor. Thereby, we increases the chance of accepting more tasks on the processor and achieve better processor throughput and utilization than previous approaches [8],[9],[13].

The rest of this paper is organized as follows. Section 2 reviews some work related to real time task scheduling. Section 3 discusses the previous approach for processing power reservation. Section 4 discusses the proposed workload allocation algorithm. Section 5 gives a numerical example that illustrates our approach. In section 6, we provide a theoretical analysis and prove the correctness of our approach. Sections 7&8 describe the simulation experiments setup and discuss the results. Section 9 concludes the paper.

2. RELATED WORK

A great deal of research has been conducted to find solutions for the problem of scheduling real time tasks over various computing platforms ranges from uni-processor to geographically dispersed computing resources connected via the internet. Scheduling algorithms in [17],[3] address the problem of task allocation over Grid; The algorithms in [2],[13],[20],[22],[23],[24] address the problem of task allocation over a cluster; The algorithms in [10],[11],[16],[6],[[18],[20] address the problem of allocating tasks over the processors of multiprocessor and multicore systems; while the algorithms in [5],[14], [15],[8], [9],[4] have been proposed to ensure an efficient and predictable scheduling of real-time independent tasks over a uni-processor.

Algorithms for scheduling in real time systems can be classified based on various criteria. They can be classified based on the computing platforms as mentioned above, or based on the characteristics of the real time applications. Another classification to scheduling approaches could be based on additional performance metrics along with satisfying timing requirements such as minimizing number of processors as in [12], reducing power consumption in processors with dynamic

voltage scaling as in [1], [18], [17], [23] or achieving effective fault-tolerant in real time systems as in [22], [24].

3. TRADITIONAL CPU RESERVATION ALGORITHM

The traditional approach for CPU reservation [13] is based on the operating system *Rialto* [8] that was developed by Microsoft research. *Rialto* can schedule real time and non-real time independent tasks on a uni-processor. In this approach, the processing power reservations are made for the tasks to ensure minimum execution rate that satisfies time constraints. Each submitted real time task T_i is characterized in terms of three parameters $\{S_i , F_i , PP_i \}$ where , S_i : is the task start time, F_i: is the task finish time , PP_i is the required processing power for this task. The request for reservation is of the form reserve x % processing power out of y % available processing power for a certain time (task deadline). Where, the available processing power of a processor ranges from 0 to 100%. According to this approach a task T_i is accepted if the processor can provide available processing power not less than the task's required processing power over its deadline interval.

The processor maintains a data structure called a reservation table, such that all processing power reservations can be honored continuously. Each entry in the table has information about a scheduled task such as $\{S_i , F_i , PP_i \}$. Table 1 shows a snap shot of the reservation table of a processor between time t = 115 and t = 211. Fig.1 shows the execution profile of these tasks, which we will call Reservation Graph (RG). This graph shows the reserved processing power as a function of time.

Reserved processing power at time t (i.e. $PP_{res}(t)$) is the summation of all the required processing powers by the tasks T_j allocated on the processor at t, i.e.

$$PP_{res}(t) = \sum_j PP_j(t) \quad \text{.....(1)}$$ The available processing power on the processor at any time $PP_{ava}(t)$ is given by:

$$PP_{ava}(t) = 1 - PP_{res}(t) \quad \text{.....(2)}$$

The acceptance condition for any task T_j is:

$$PP_{ava}(t) \geq PP_j \quad \text{.....(3)} \qquad \forall t \in [S_j, F_j]$$

i.e.

$$PP_{\min\_ava\_j} = \min\left[PP_{ava}(t)\right]_{t=S_j}^{F_j} \geq PP_j \quad \text{.....(4)}$$

Table 1. Reservation table for tasks T_1, T_2, T_3, and T4.

T_i	S_i	F_i	PP_i
T_1	115	135	0.2
T_2	124	156	0.1
T_3	143	172	0.3
T_4	167	211	0.4

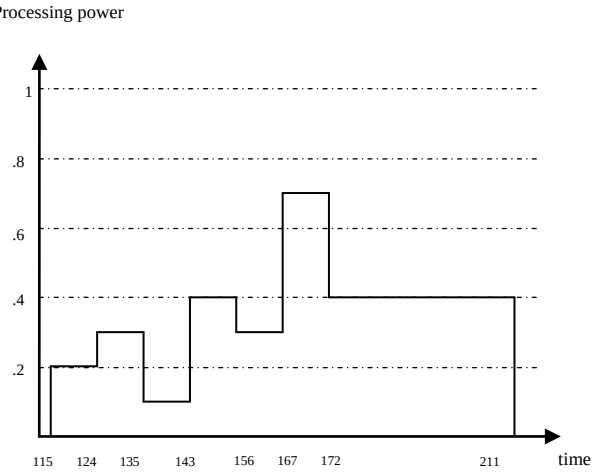

Fig. 1: Reservation graph RG of the processor

Algorithm1 specifies the steps to schedule tasks on a processor using Rialto approach.

Algorithm 1: traditional approach

Input: a set of real-time tasks $\mathbf{T} = \{T_1, T_2, ..., T_n\}$

Output: RG, acceptance rate

Begin

1. acceptance_counter = 0

2. **For** each $T_j \in \mathbf{T}$

Compute: $PP_{\min\_ava\_j} = \min\left[PP_{ava}(t)\right]_{t=S_j}^{F_j}$

If ($PP_j \leq PP_{\min\_ava\_j}$) then

Increment acceptance_counter;

Update $PP_{res}(t)$ in the window $[S_j, F_j]$

as follows: $PP_{res}(t) = PP_{res}(t) + PP_j$ (4.2)

3. acceptance rate = acceptanc_counter/n

End

The main disadvantage of this approach is that a task T_j is rejected if its constant required processing power cannot be guaranteed for its deadline. This increases rejection rate of the tasks. Consider for example the tasks in Table 1 and Figure1, assume that a task T_5 is submitted with the following parameters: {S_5=170, F_5=180, PP_5= 0.4}. Applying the traditional algorithm we find that: $PP_{\min\_ava\_5}$ = 0.3. Since $PP_5 > PP_{min\_ava\text{-}5}$,$T_5$ is rejected as shown in Fig. 2 below.

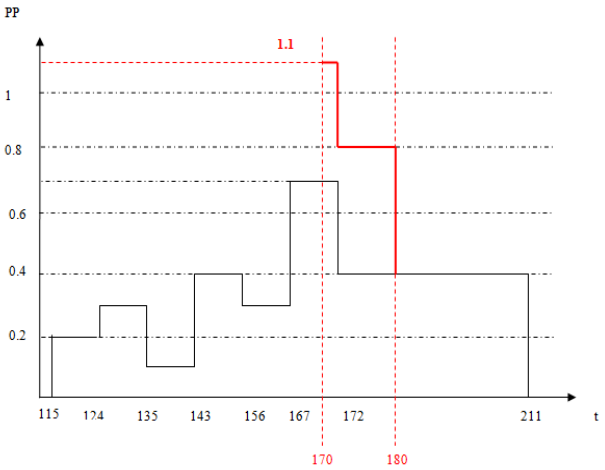

Fig. 2: T_5 is rejected using the traditional approach

Our proposed algorithm modifies this algorithm by allocating the task using its workload instead of its processing power to boost the acceptance rate.

4. THE PROPOSED WORKLOAD ALLOCATION APPROACH

In this approach, a variable processing power is assigned to the allocated task T_j to satisfy its deadline instead of rejecting it if its constant required processing power cannot be guaranteed over its deadline. A task T_j is accepted if its required workload is not greater than the available workload WL_{ava_j} in the window [S_j , F_j].

The workload WL_j of a task T_j can be computed using equation 5:

$$WL_j = PP_j * d_j \quad .....(5)$$

Where $d_j = F_j - S_j$ is the task deadline.

To calculate the available workload, the task window [S_j , F_j] is segmented into m_j segments. Each segment $g_{k,j}$ with time length equal to $l_{k,j}$ and is characterized by a constant available processing power ($PP_{ava\_k,j}$). Algorithm 2 describes this procedure in detail.

Algorithm 2: workload distribution (WL) approach

Input: a set of real-time tasks $\boldsymbol{T} = \{T_1, T_2, ..., T_n\}$

Output: RG, Acceptance rate

Begin

1. acceptance_counter = 0

2. **For** each $T_j \in \boldsymbol{T}$

Compute $WL_j = PP_j * d_j$

$$WL_{ava\_j} = \sum_{k=1}^{m_j} (PP_{ava\_k,j} * l_{k,j})$$

If ($WL_{ava\_j} \geq WL_j$) then

Increment acceptance_counter

Call *Algorithm 3* to update RG

with workload of T_j

3. acceptance rate = acceptanc_counter/n

End

If a task T_j is accepted; WL_j is distributed over T_j 's window by assigning T_j a new variable processing power (PP_j (t); $S_j < t < F_j$) enough to satisfy WL_j . To distribute WL_j , the available processing power of each segment in the window [S_j , F_j] is calculated using the Reservation Graph (RG). If $PP_{ava\_k,j}$ is greater than or equal to PP_j , then a processing power equal to PP_j is reserved for T_j over this segment, otherwise, a processing power equal to $PP_{ava\_k,j}$ is reserved for T_j and accumulate the remaining workload, $WL_{rem\_j} = (PP_j - PP_{ava\_k,j}) * l_{k,j}$. The accumulated remaining workload is then distributed over the segments that have available processing power. This procedure is described in detail in algorithm 3.

Algorithm 3: update RG (allocate accepted task on RG)

Input: a real-time task, T_j

Output: updated RG

Begin

1. Initialize $WL_{rem\_j} = 0$

2. **For** $(k = 1, k \leq m_j, k++)$

If ($PP_{k,j\_ava} \geq PP_j$) then

$PP_{res\_k,j} = PP_{res\_k,j} + PP_j$

Else

$PP_{res\_k,j} = PP_{res\_k,j} + PP_{ava\_k,j} = 1$

$WL_{rem\_j} = WL_{rem\_j} + (PP_j - PP_{ava\_k,j}) * l_{k,j}$

3. $k = 1$

4. **While** ($WL_{rem\_j} > 0$)

If ($PP_{res\_k,j} \neq 1$)

Compute the unused load:

$WL_{unused\_k,j} = PP_{ava\_k,j} * l_{k,j}$

If ($WL_{unused\_k,j} < WL_{rem\_j}$) then

$PP_{res\_k,j} = 1$

$WL_{j\_rem} = WL_{j\_rem} - WL_{uused\_k,j}$

$k = k + 1$

Else

$$PP_{res\_k,j} = PP_{res\_k,j} + \frac{WL_{rem\_j}}{l_{k,j}}$$

$WL_{rem\_j} = 0$

Else

$k = k + 1$

End

5. A NUMERICAL EXAMPLE

Consider the reservation table1 again, as we have seen task T_5 has been rejected using the traditional approach, but when applying our approach:

The required workload:

$WL_5 = 0.4 * (180-170) = 4$

The available workload $WL_{ava\_5}\Big|_{t=170}^{180}$

$= (172-170) * 0.3 + (180-172) * 0.6 = 5.4$

Since $WL_{ava\_5} > WL_5$ then T_5 is accepted

The new $PP_5(t)$ in the window $[S_5, F_5]$

is calculated as follows:

The window $[S_5, F_5]$ is partitioned into two segments:

$g_{1,5}$ (170< t <172) & $g_{2,5}$ (172< t <200)

1. Segment $g_{1,5}$ (170< t <172)

Since $PP_{ava\_1,5} = 0.3 < PP_5$, then

$PP_{res\_1,5} = 1$

$WL_{rem\_5} = (0.4 - 0.3) * (172-170) = 0.2$

2. Segment $g_{2,5}$ (172< t <200)

Since $PP_{2,5\_av} = 0.6 > PP_5$, then

$PP_{res\_2,5} = PP_{res\_2,5} + PP_5 = 0.8$

Now, the remaining workload WL_{rem_5} will be distributed over the period (172<t<180). The new processing power will be:

$$PP_{res\_2,5} = PP_{res\_2,5} + \frac{0.2}{(180-172)}$$

$= 0.8+0.025=0.825$

Figures (3a-d) illustrate the steps of our algorithm and figure 4 shows the RG after accepting and allocating T_5 on the processor.

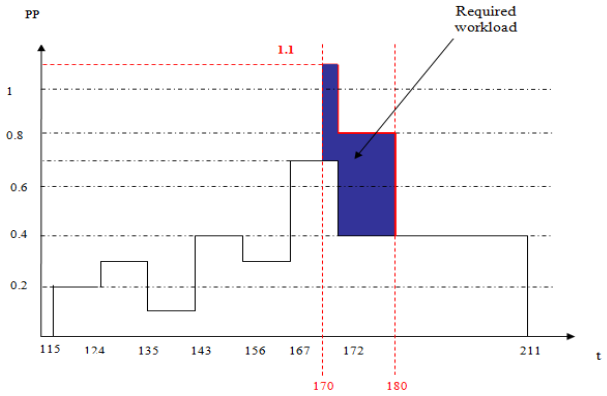

Fig. 3a: Step 1, computing required workload for T_5

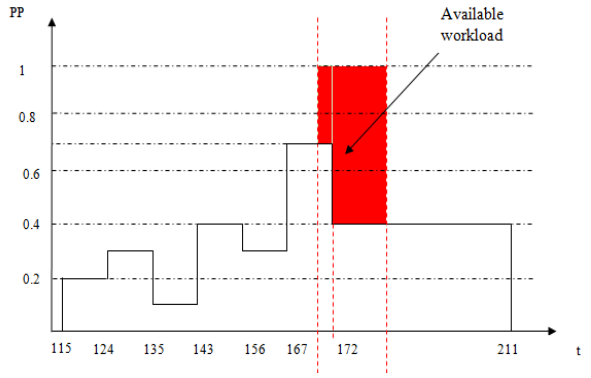

Fig. 3b: Step 2, computing available workload

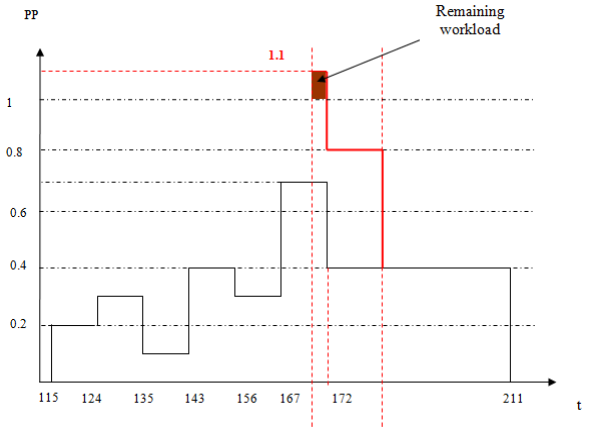

Fig. 3c: Step 3, computing remaining workload

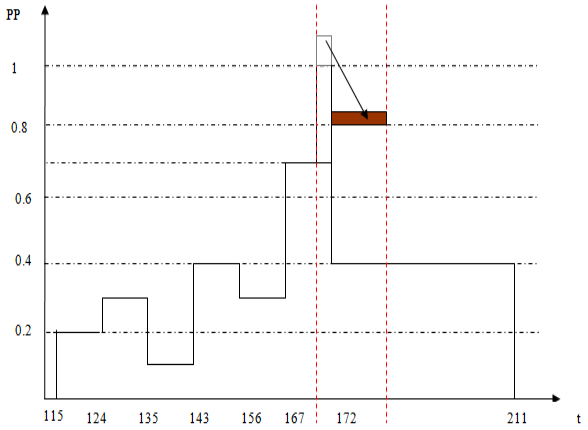

Fig. 3d: Step 4, distributing remaining workload

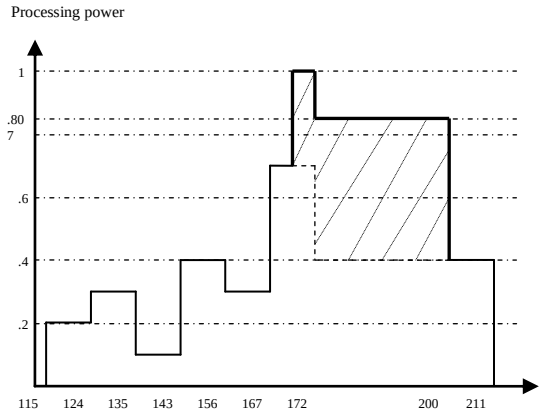

Fig. 4: RG after accepting T_5

6. THEORETICAL ANALYSIS OF OUR APPROACH

In this section we prove that our approach (WL approach) outperforms the traditional approach or in the worst case equals it by providing the following two theories:

Theorem 1: If a task is accepted on a processor using the traditional approach, then it will be accepted using the WL approach.

Proof: Suppose that T_j is a task of parameters (S_j, F_j, PP_j). If T_j is accepted using Algorithm 1 then according to inequality 4, for all segments $g_{k,j}$ in $[S_j, F_j]$, we have $PP_{ava\_k,j} \geq PP_j$ as shown in Fig. 5. From this graph we can write formulas for WL_{ava_j} and WL_j as follows:

$$WL_{ava\_j} = \sum_{k=1}^{m_j} (PP_{ava\_k,j} * l_{k,j}) \quad \ldots\ldots (6)$$

$$WL_j = \sum_{k=1}^{m_j} (PP_j * l_{k,j}) \quad \ldots\ldots (7)$$

Since $PP_{ava\_k,j} \geq PP_j \quad \forall g_{k,j}$ within $[S_j, F_j]$, from equations (6 and 7), we conclude that: $WL_{ava\_j} \geq WL_j$ which means that the task is also accepted using *Algorithm 2*.

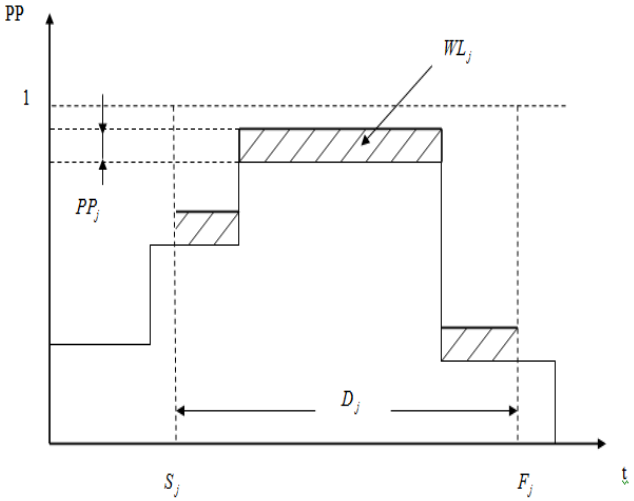

Fig. 5: A task accepted using traditional approach is also accepted using WL approach

Theorem 4.2: If T_j is rejected using the traditional approach, it can be accepted by WL, it may be accepted by WL approach.

Proof: If task T_j is rejected by Algorithm 4.1 this means that the condition $PP_j \leq PP_{ava\_k,j}$ is not satisfied for at least one segment $g_{k,j}$ within $[S_j, F_j]$ as shown in Fig. 6. From this graph we can write WL_j and WL_{ava_j} as follows:

$$WL_j = WL_{add\_j} + WL_{rem\_j} \quad \ldots\ldots (8)$$

$$WL_{ava\_j} = WL_{add\_j} + WL_{unused\_j} \quad \ldots\ldots (9)$$

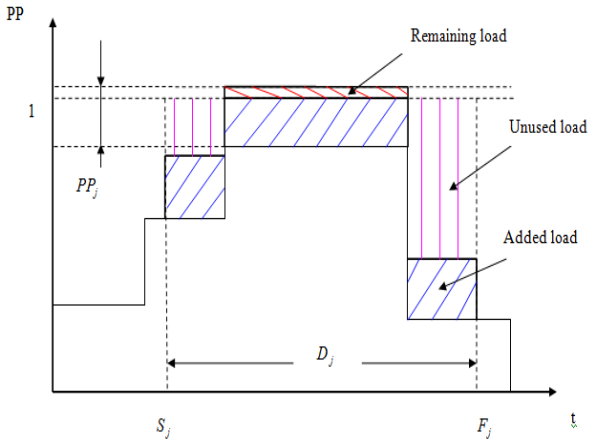

Fig 6: A task rejected using the traditional approach

We can distinguish between two cases:

Case 1: $WL_{rem\_j} > WL_{unused\_j} \Rightarrow WL_j > WL_{ava\_j}$

$\Rightarrow T_j$ is rejected by WL approach as well.

Case 2: $WL_{rem\_j} \leq WL_{unused\_j} \Rightarrow WL_j \leq WL_{ava\_j}$

$\Rightarrow T_j$ is accepted by WL approach while it is rejected by the traditional approach.

Thus, task T_j, although it is rejected using the traditional approach, it can be accepted using the new WL approach. □

The second situation is represented in Fig. 7 below.

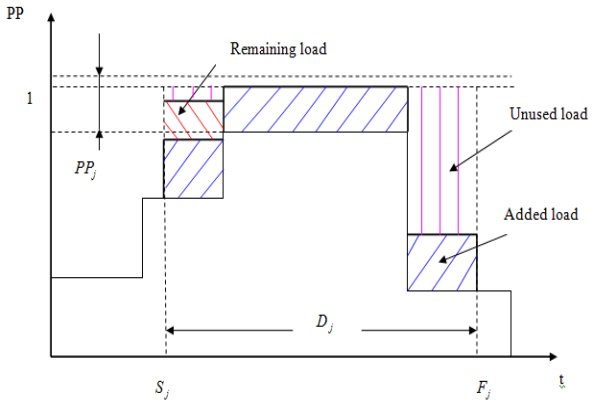

Fig 7: The same task accepted using our workload approach

7. SIMULATION AND EXPERIMENTS

7.1 Experiments Setup

A simulation study that consists of a set of four experiments have been conducted to show the performance of our proposed approach relative to the traditional one. These experiments aimed to illustrate the behavior of the two approaches, in terms of the *acceptance rate*, for different values of arrival rates (**λ**) and departure rates (**μ**) of tasks. In each experiment a set of 10000 tasks are generated randomly as follows:

1. A uniform distribution is used for the generation of the execution times of the tasks. The mean execution time (**1/μ**) is set to a fixed value during each experiment, which are {10, 15, 20, 25} for the four experiments respectively.
2. Exponential probability distribution is used to generate the inter-arrival time between consecutive tasks. For each value of **(1/λ),** a set of 10000 tasks is generated. The values of mean inter-arrival time (**1/λ**) are {10, 20, ..,120} sec during each experiment. Small values of **λ** means that the tasks arrive far apart while large values mean that the release times of the tasks are very close and the processor is loaded.
3. 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).

7.2 Simulation Results

Figures 8, 9 and 10 show the acceptance rate vs. traffic intensity for each case of the average execution time. In all experiments, results show that the new approach outperforms the traditional one. Results also show that both algorithms rejects more tasks when tasks arrive faster (large values of λ/μ) than the processor can handle but the new approach is superior to the traditional one. In converse, both algorithms perform competitively well for small values of (λ/μ) where the tasks arrive far apart from each other.

Figure 12 shows the improvement percentage in the acceptance rate achieved by the new approach over the traditional one in each experiment. As shown in the graph the improvement diminished as the inter-arrival time increases. This is due to the fact that both approaches perform very well for large values of inter-arrival time (slow arrivals). The graph also shows that we achieve higher amount of improvement for lower values of the inter-arrival time (fast arrivals). Hence, we conclude that the proposed approach has a major improvement when tasks arrive at high rate. In this later case, tasks are likely to be rejected if the traditional approach is used. Finally, results also show that the proposed approach tends to achieve better improvement for large values of mean execution time (longer tasks). This is because it is more difficult to have a constant available processing power during the task's deadline.

8. CONCLUSIONS

This paper presented an improved processing power reservation algorithm that allocates the workloads of independent real time tasks , instead of allocating their processing powers, on a processor. The proposed algorithm improves the processor throughput (i.e. boosts the acceptance rate of the admitted tasks). Simulation results showed that the new algorithm is superior to the traditional one. In this paper, we applied the algorithm to a stream of tasks submitted to a single processor. Our algorithm is also beneficial in scheduling real-time applications represented by task graphs in a multiprocessor environment such as cluster environment. Since rejecting a task of an application leads to rejecting the whole application, it is obvious that the new algorithm will overcome this problem and hence will produce a better performance in scheduling real-time task graphs on a cluster of computers.

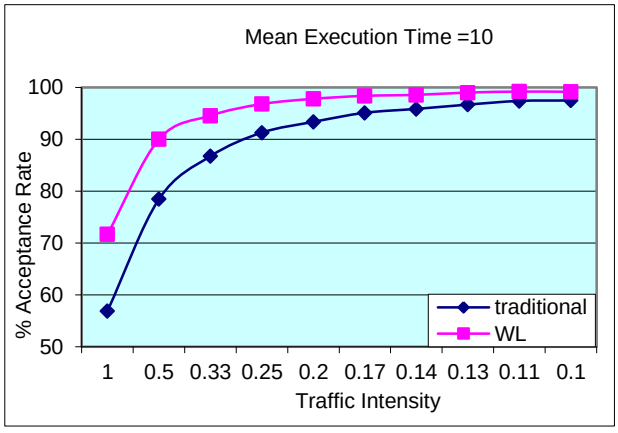

Fig 8: Acceptance rate at mean execution time (1/μ) = 10 and different traffic intensity

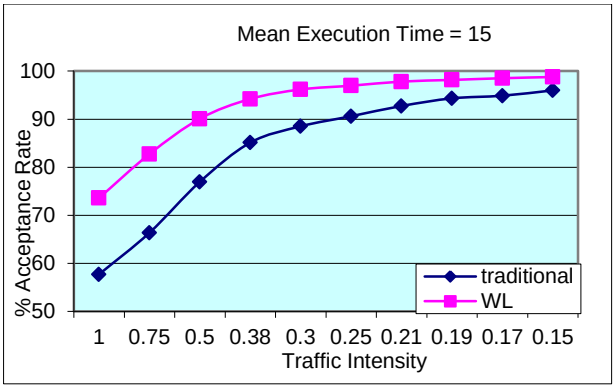

Fig 9: Acceptance rate at mean execution time (1/μ) = 15 and different traffic intensity

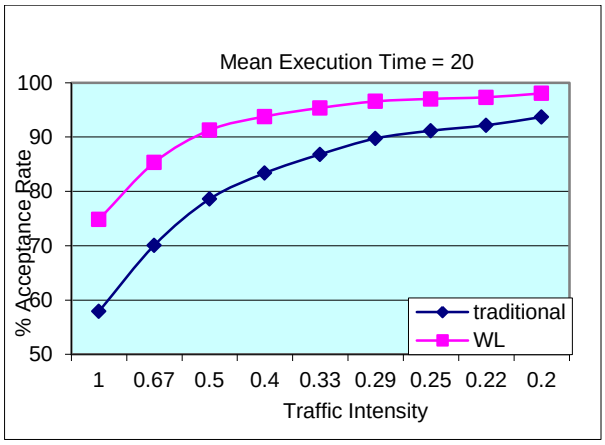

Fig 10: Acceptance rate at mean execution time (1/μ) = 20 and different traffic intensity

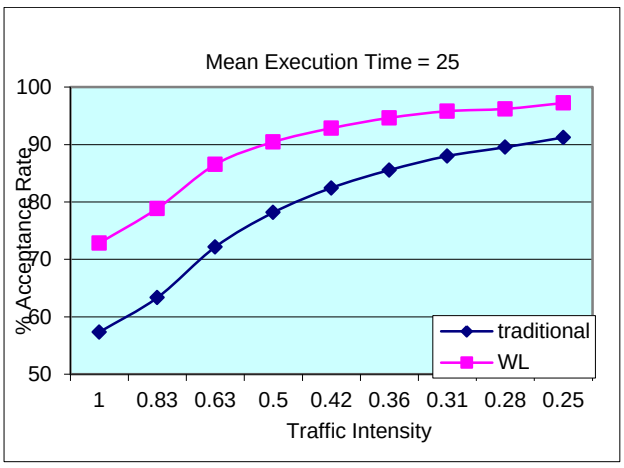

Fig 11: Acceptance rate at mean execution time (1/μ) = 25 and different traffic intensity

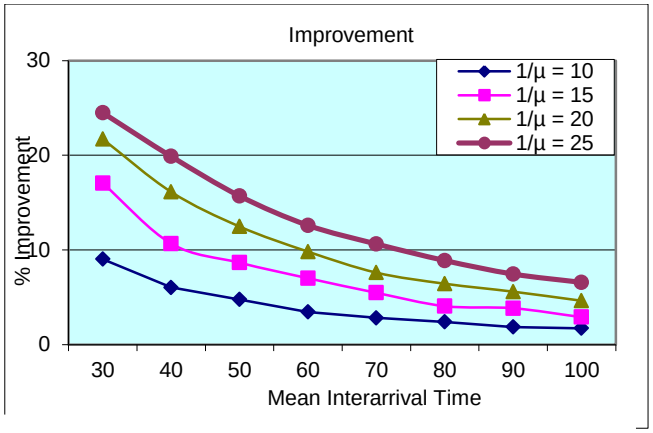

Fig 12: The improvement rate of the proposed approach over the traditional at different values of mean inter-arrival and execution times.

9. REFERENCES

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

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

[3] Caniou, Y., Charrier, G., Desprez F., 2010 ,Analysis of tasks reallocation in a dedicated grid environment, In Proceedings of IEEE international conference on cluster computing.

[4] Deng, Z., Liu, J.W.-s and Sun, S., 1996, Dynamic scheduling of hard real-time applications in open system environment, Technical Report, University of Illinois, USA.

[5] Ford, B. and Susarla, S., 1996, CPU inheritance scheduling, operating systems review.

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

[7] H. Heidari and A. Chalechale, "Scheduling in Multiprocessor System using Genetic Algorithm", International Journal of Advanced Science and Technology, june 2012.

[8] Jones, M.B., Roşu, D., Roşu, M., 1997, CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities, In Proceedings of the 16th ACM Symposium on Operating System Principles.

[9] Jones, M.B. 2001, Two case studies in predictable application scheduling using Rialto/NT , In Proceedings of 7th Real-Time Technology and Applications Symposium.

[10] M. Lombardi, M. Milano, L. Benini, "Robust Scheduling of Task Graphs under Execution Time Uncertainty", IEEE transactions on computers, 2011.

[11] Niemeier, M., Wiese, A., Baruah, S., 2011, Partitioned real-time scheduling on heterogeneous shared-memory multiprocessors, In Proceedings of the 23rd Euromicro Conference on Real-Time Systems.

[12] O. Jaewon and W. Chisu, "Genetic Algorithm Based Real Time Task Scheduling with Multiple Goals", Journal of systems and software, 2004.

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

[14] Regehr, J., J. and Stankovic, J.A., 2001, Augmented CPU reservations: Towards predictable execution on general-purpose operating systems, In Proceedings of the IEEE Real-Time Technology and Applications.

[15] Stoica, I., Abdelwahab, H., Effay, K., Baruah, S.K., Gehrke, J.E. and Plaxton, C.G., 1996, A proportional share resource allocation algorithm for real-time, time-shared systems, In Proceedings of 17th IEEE real-time systems symposium.

[16] Satish, N. R., Ravindran, K., Keutzer, K., 2008, Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors, In Proceedings of the 8th ACM international conference on Embedded software.

[17] S.Baskaran, P. Thambidurai, "Energy efficient real-time scheduling in distributed systems", IJCSI International journal of computer science issues, 2010.

[18] S. Jin, G. Schiavone · D. Turgut, "A performance study of multiprocessor task scheduling algorithms", Journal of Supercomputer, 2008.

[19] W. Y. Lee, "Energy-Efficient Scheduling of Periodic Real-Time Taks on Lightly Loaded Multicore Processors", IEEE Transactions on Parallel and Distributed Systems, 2012.

[20] 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, 2003.

[21] X. Lin, A. Mamat, Y. Lu, J. Deogun, S. Goddard , "Real-time scheduling of divisible loads in cluster computing environments", International Journal of Parallel and Distributed Computing, Elsevier, 2010.

[22] X. Zhu, X. Qin, M. Qiu, "QoS- Aware fault-Tolerant Scheduling for Real Time Tasks on Heterogeneous clusters" IEEE transactions on Computers, 2011.

[23] X. Zhu, C. He, K. Li, X. Kin, "Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters" , Journal of parallel and distributed computing, 2012.

[24] Zhu, X., Zhu, J., Ma, M., Qiu, D., 2010 ,SAQA: A self adaptive QoS-aware Scheduling Algorithms for Real Time Tasks on Heterogeneous Clusters, In Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing.