ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 2, Issue 4 (Oct. - Dec. 2014)
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014)

Harmonic-Aware Scheduling for Fixed-Priority Real-Time Systems

Ashwini G. Ingale
Dept. of CSE, Masters of Engineering, India

Abstract
This paper discusses a semi-partitioned fixed priority scheduling of sporadic tasks on identical multi-processors based on Rate Monotonic scheduling. A well known fixed-priority algorithm is the rate-monotonic algorithm. This algorithm assigns priorities on their periods: shorter the period, higher the priority. The rate of a task is the inverse of its period. Hence, the higher is the rate, higher its priority. Performed work deals with the following. First, we use the harmonic property for a task set. Harmonic property means task with periods being integer multiples of each other that has been widely studied on uniprocessor system. We developed one example with harmonic property in semi-partitioned scheduling. Two semi-partitioned approaches are developed. The first scheduling approach-1 for the tasks with utilization factor of each task no more than 0.5. The second one, scheduling approach-2, is developed for more general task sets, i.e. the utilization factor of each task is no more than 1. Twenty simulation results are carried out. In each simulation, different numbers of task are generated. From obtained results it is found that all the generated tasks are successfully scheduled for first 20 simulation runs.

Keywords
Harmonic, Real-Time Semi-Partitioned Scheduling, Fixed Priority, RMS.

I. Introduction

Systems are referred to as real-time when their correct behaviour depends not only on the correctness of operations which they perform but also by their accomplished time. Nowadays, these system present in a vast variety of embedded systems such as automotive/avionic control system, military applications and environmental monitoring systems.

A real Time System (RTS) is a computer-controlled mechanism in which there are strict timing constraints on the computer's actions. In other words, the correctness of the system depends not only on the logical computational result, but also on the time at which the results are produced. Such systems can be simple like alarm clock or can be the most complex systems built. Real-time systems are classified in two categories: soft and hard. Hard-Real-Time system, which must meet its deadline, otherwise it will create undesirable damage or a fatal error to the system. Soft-Real-Time System has an associated deadline that is desirable but not mandatory, it still o schedule and complete the task even if it has passed its deadline. It is considered hard if it can guarantee that a certain event will always be carried out in less than a specific time and failure to respond to the event may cause a catastrophe, which is intolerable. The important part of such a system is the scheduling algorithm that decides on the timing, preemption, and resuming of requests.

Liu and Layland discovered the famous utilization bound N ($2^{1/2}$-1) for fixed priority scheduling on uniprocessor in the 1970's [1]. The purpose of utilization bound is to test the scheduling of real time task sets with simple and practical way. Utilization bound can also be used as a metric to evaluate the quality of scheduling algorithms. A hard real-time system must execute a set of concurrent real-time tasks in such a way that all time-critical tasks meet their specific deadlines. Every task needs computational and data resources for job completion. The scheduling problem is concerned with the allocation of the resources to satisfy the timing constraints.

In multi-core processor platforms, there are various processors available upon which these jobs may execute. In uniprocessor platforms, there is exactly one shared processor available. Two approaches traditionally considered for scheduling algorithms are Rate monotonic Scheduling (RMS) and earliest Deadline First (EDF) scheduling, which have been proven to be optimal for uniprocessor scheduling [2]. Hence when the problem comes towards multi-core platforms, these uniprocessor scheduling algorithms are no longer optimal [3] because in multi-cores systems, there is need to decide where to execute a real- time task.

There have been large numbers of researches published on real-time scheduling for multi-core systems [4-8]. These scheduling algorithms consist of two parts [9, 10]: the partitioned approach (e.g. [4]) and the global (or non-partitioned) approach (e.g. [5]). In the partitioned scheduling approach, each task is assigned to a processor beforehand at runtime each task can only execute on this particular processor. All instances from the same task will be executed solely on that particular processor. In the global scheduling approach, each real-time task is assigned to available processor at run time. Both approaches have their own advantages and disadvantages, and none of them dominates the other in terms of schedulability [9].

Recently a multi-core scheduling approach [4, 5, 11-13] i.e. is called semi-partitioned scheduling approach for the purpose of improving schedulability with succeeding the advantage of partitioned scheduling as much as possible. In semi-partitioned scheduling, most tasks are assigned to one particular processors to reduce runtime overhead, and a few tasks are split into several subtasks to be assigned on the different processors. Semi-partitioned approach has been shown to be sound and practical in the real implementation and theoretically, have the greatest achievement compared to the global and the partitioned approaches [2, 14, 15]. According to the prior work, a class of semi-partitioned scheduling offers a significant improvement on schedulability, as compared to a class of partitioned scheduling, with less preemptions and migrations than a class of global scheduling. However, little work has studied on fixed priority algorithms.

This paper presents a semi-partitioned fixed priority scheduling of sporadic tasks on identical multi-processors based on RMS. A well known fixed-priority algorithm is the rate-monotonic algorithm. This algorithm assigns priorities on their periods: smaller the period, the higher the priority. The rate of a task is the inverse of its period. Hence, the higher its rate, higher its priority. Performed

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

work deals with the following. First, we use the harmonic property for a task set. Harmonic property means task with periods being integer multiples of each other that has been widely studied on uniprocessor system. We developed one example with harmonic property in semi-partitioned scheduling. Two semi-partitioned approaches are developed. The first scheduling approach-1 for the tasks with utilization factor of each task no more than 0.5. The second one, scheduling approach-2, is developed for more general task sets, i.e. the utilization factor of each task is no more than 1.

The rest of paper is organized as follows. Section II describes the formulation of problem and other background information required. First and second proposed approaches are discussed in section III and IV. Experiments results and discussion are presented in section V. Finally, the conclusion is summarized in section VI.

II. Problem Formulations

This section discusses the formulation of problem. The problem of semi-partitioned scheduling of sporadic
tasks on multi-core platform based on RMS policy. First, we explain problem model with their objectives and then background information.

Problem model:

We consider a multi-processor platform consisting of M processor i.e. P= $\{P_1, P_2, ...P_M\}$.

A task set T= $\{t_1, t_2...t_N\}$ consist of N sporadic tasks. Each task t_i is characterized by a tuple (e_i, p_i) ,where e_i is the worst-case execution time of t_i and p_i is the minimum interarrival time between any two consecutive requests of t_i. p_i is also called period of t_i.

We make some assumptions:

1. The deadline of each task is identical to its period;
2. T is sorted with decreasing priority order, i.e. task t_i has higher priority than t_j if i<j
3. We use RMS strategy to assign the priorities: task with shorter periods have higher priority.
4. The utilization of a task set is less than or equal to Liu&Layland bound.

The utilization factor of a task t_i is denoted as u_i where

$$u_i = \frac{e_i}{p_i} \quad (1)$$

The total utilization of a task set T is denoted as U (T) where

$$U(T) = \sum_{ti \in T} u_i \quad (2)$$

The system utilization of task set T on a multi-core platform with M processors is denoted as $U_M(T)$, where

$$U_M(T) = \frac{U(T)}{M} \quad (3)$$

Liu and Layland [5] showed that a task set T can be feasibly scheduled by RMS on a uniprocessor if

$$U(T) \le \theta(N) = N(2^{1/N} - 1). \quad (4)$$

Θ (N) is also traditionally referred to as the Liu&Layland bound.

The utilization of a task set is less than or equal to Liu&Layland bound.

Example:

A task set with five real-time tasks and their scheduling is shown in Table 1 and Figure 1 respectively.

Table 1: A task set with five real-time tasks

T_i	e_i	p_i	u_i
1	4	9	0.44
2	6	10	0.6
3	3	18	0.16
4	2	20	0.1
5	15	30	0.5

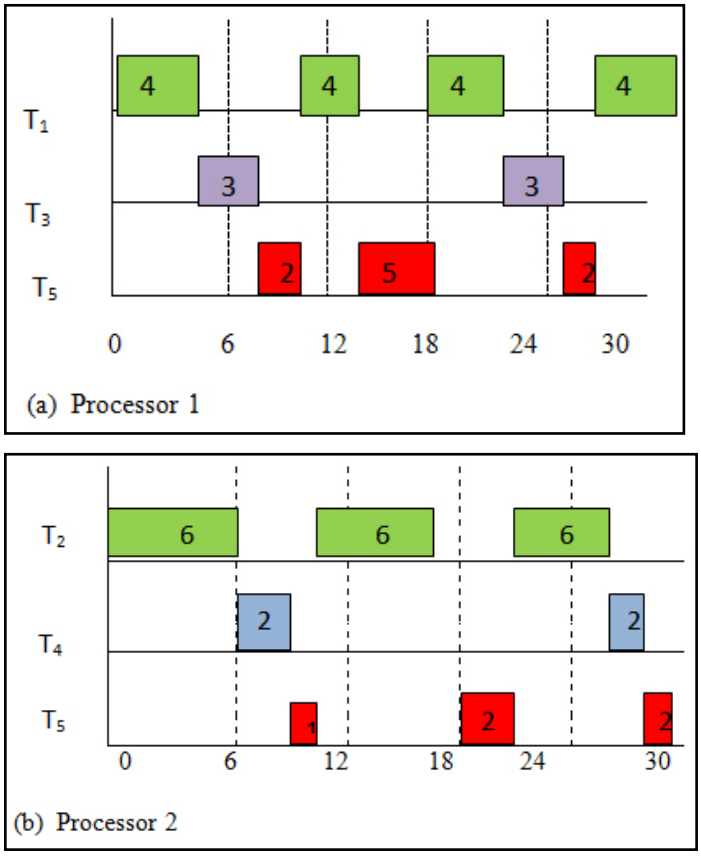

Fig. 1: Allocation fails when simply grouping harmonic tasks and assigning them to the same processor.

Consider a two-processor platform with a task set as shown in Table1. Since T1 and T3 are harmonic, we can group T1 and T3 to one processor, i.e. Processor 1. Similarly, we can group T2 and T4 to the other processor, i.e. Processor 2. Since no processor can accommodate T5 entirely, we have to split T5 between these two processors. With this assignment, there are two problems. First, as shown in Figure 1(a), the maximum capacity that can be accommodated in Processor 1 is 9. Since the subtasks from T5 cannot be executed concurrently on two processors, at most 5 time units from Processor 2 can be utilized by T5 as shown in Figure 1(b). As a result, T5 cannot complete before its deadline. Second, in order to use all 5 time units on Processor 2, we need complicated process migration controls and synchronization mechanisms. Note that, if we assign T1 and T5 to one processor, and the other tasks to another processor, it is easy to verify that the schedule is feasible.

From this example, to take the advantage of harmonic relationship among tasks to improve the feasibility a critical problem is how to wisely select the task to split and to among different processors. To solve this problem, two scheduling approaches are proposed, namely, approach-1 and approach-2.

III. Scheduling approach-1

We introduce first approach i.e. Scheduling approach-1 with utilization is no more than 0.5. To improve the scheduling performance, we use the harmonic relationship in which it is not necessary that all tasks in the same task set are strictly harmonic. We always select the processor with minimal total utilization of tasks. Approach-1 assigns tasks to processors in increasing priority order. A task can be assigned to the current processor, if all tasks on current processor can meet their deadline. If a task cannot be accommodated entirely by any processor then we split task into two parts. The first part is assigned to the current processor and the second part is assigned to the next processor.
We used the worst-fit strategy. These approaches used the task splitting technique to get as high priority as possible on each processor. To split a task, we adopt a simple heuristic that assigns subtasks to the processor with the highest available capacity.

Scheduling Approach-1:
Require: utilization is less than or equal to 0.5, T= $\{t_1, t_2,..t_N\}$ and $P= \{P_1, P_2,....P_M\}$
1: **while** there is non_full processor and an unassigned task **do**
2: t_i : the task with the lowest priority in T;
3: P_m: the processor with minimum harmonic index in P;
4: **if** $T_{Pm} + \tau_i$ is feasible **then**
5: $t_i \rightarrow P_m$;
6: **end if**
7:**if** P_m: = the processor with the maximum capacity (greater than 0) for t_i; **then**
8: Task is assigned entirely
9: **else**
10: Split task t_i and assigned part of t_i to the processor until it is maximally utilized;
11: **end if**
12: **end while**
13**: if** there is unassigned task **then**
14: Return "Succeed if all tasks are allotted";
15: **else**
16: Return "Allocation Failed";
17: **end if**

T is the list of accommodating unassigned tasks, sorted and a multiprocessor system *P*. Approach-1 makes the assignment decision for each task through the "while" loop from line 1 to line 12. Among all unassigned tasks left in T, we select the task t_i with the lowest priority. t_i *is* assigned to the processor with the minimum harmonic index. Harmonic index i.e. to measure the harmonicity of a task sets. If this assignment fails, then split occurs. We choose the processor with the maximum execution capacity for t_i. We assigned t_i entirely since there is maximum capacity available on processor. Otherwise, we split t_i into two subtasks t_i^1 , t_i^2 and assign that part of t_i to the processor until it is maximally utilized.

IV. Scheduling approach-2

In the following, we introduce second semi-partitioned fixed-priority scheduling approach i.e. Scheduling approach-2, which is developed for general task set. Scheduling approach-2 pre-assigns exactly the heavy tasks for which pre-assigning them will not cause any tail subtask to miss deadline.
Scheduling approach-2 contains three steps:

1) First, we pre-assign the heavy tasks that satisfy a particular condition to one processor each in decreasing priority order.

2) We do task partitioning with the remaining tasks and remaining processors using Scheduling approach-1 until all the normal processors are full.

3) The remaining tasks are allocated to the pre-assigned processors; the assignment selects one processor with lowest priority pre-assigned task to assign as many tasks as possible, until it becomes full, then select the next processor.

The main idea of scheduling approach-2 is to pre-assign each heavy task whose tail subtask might get a low priority, before partitioning other tasks, therefore these heavy tasks will not be split. If one simply pre-assign based all heavy tasks, it is still possible for some tail subtask to get a low priority level on its host processor.

In Scheduling approach-2, the pre-assignment for heavy tasks follows the same strategy as introduced in [6]. Specifically, for any heavy task t_i let PR_i^E denote the set of empty processors before t_i assignment and $\left|PR_i^E\right|$ denote the number of processors in this set. Then a heavy task t_i needs to be pre-assigned to an empty processor if

$$\sum_{j>i} u_j \le \left(\left|PR_i^E\right| - 1\right)\Theta(N)$$

There is an important difference between assigning tasks to remaining (normal) processors and to pre-assigned processors. When tasks are assigned to normal processors, the approach always selects the processor with the minimal (the same as in approach-1). In the contrast, when tasks are assigned to pre-assigned processors, always the processor at the front of PR_{pre} is selected, i.e., we assign as many tasks as possible to the processor in PR_{pre} whose pre-assigned task has the lowest priority, until it is complete.
In this approach, we use grid simulation toolkit. The GridSim toolkit allows modeling and simulation of entities in parallel and distributed computing (PDC) systems-users, applications, resources, and schedulers for design and evaluation of scheduling algorithms.

Scheduling Approach-2:

Require: Task set: T = $\{t_1,t_2 ..t_N\}$, Multiprocessor: $PR = \{P_1, P_2...P_M\}$, Grid simulation.

1: Heavy tasks are pre-assigned to empty processor set denoted as PR_{Pre} ;
2: **for** i = 1 to N **do**
3: **if** utilization is less than or equal to 0.5 and

$\sum_{j>i} u_j \le \left(|PR_i^E|-1\right)\Theta(N)$ **then**
4: tasks are assigned to the processor|;
5: **end if**
6: **end for**
7: **while** there is non_full processor and an unassigned task **do**
8: t_i: the task with the lowest priority in T
9: t_j: the task with the lowest priority in T_{PRpre};
10: **if** t_i has higher priority than t_j **then**
11: Push *PR* (t_j) from PR_{Pre} to PR;
12: **end if**
13: P_m: = the processor with minimum harmonic index in PR;
14: **if** $T_{Pm}+t_i$ is feasible **then**
15: Assign τ_i to processor *PRm*;
16: Continue;
17: **end if**
18: PR_m: the processor with maximum capacity for t_i in *PR*;
19 **if** PR_m does not exist, **then** Break, **end if**
20: **if** $T_{PRm}+t_i$ is feasible **then**
21: Assign t_i to processor *PRm*;
22: **else**
23: Split task t_i and assigned part of t_i to the processor until it is maximally utilized;
24: **end if**
25: **end while**
26: **if** Task is unassigned to the processor **then**
27: Return "Succeed if all tasks are allotted";
28: **else**
29: Return "Allocation Failed";
30: **end if**

V. Simulation Results Discussion

Table 2: Twenty simulation results for total no. of tasks and total task execution time

Simulation Run	Total no. of tasks	Total task execution time
1	293	11373
2	293	11256
3	1428	57150
4	1417	57106
5	1004	39721
6	1040	41837
7	902	36271
8	988	37851
9	1510	59104
10	1242	48851
11	267	10541
12	537	19921
13	735	28178
14	1290	48798
15	1465	58648
16	325	13479
17	104	3566
18	1280	52536
19	1237	48735
20	135	4667

Twenty simulation results are carried out. In each simulation different number of task are generated. For all the tasks, the total execution time is evaluated. If all tasks are successfully assigned to the processors then we can say that a task set is successfully scheduled. Inter arrival time for first 30 tasks of first 10 simulation runs is shown in Table 3. Table 4 shows the inter arrival time for next 30 tasks of first 10 simulation runs.

Table 2 shows the total task execution time for the relevant number of tasks for first twenty simulation runs.

Inter arrival time plot for the first 30 tasks of first two simulation runs is shown in Figure 2 and Figure 3. Figure 4 and Figure 5 shows the plot of inter arrival time of next 30 tasks of first two simulation runs.

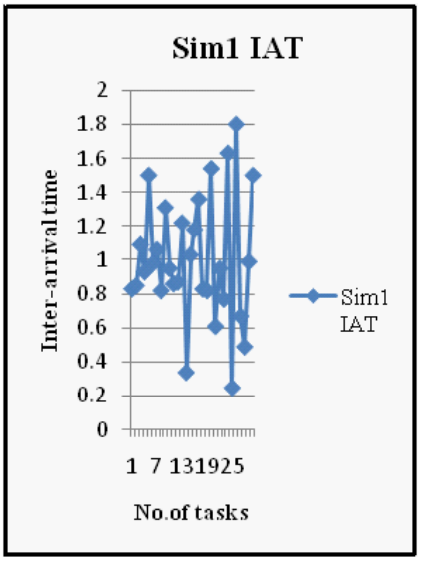

Fig.2: Simulation result 1 for first 30 tasks

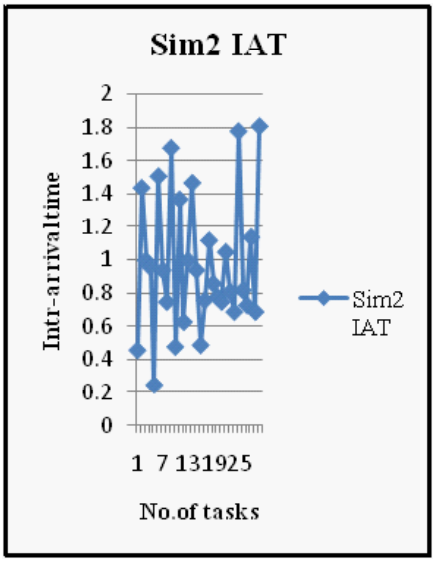

Fig. 3: Simulation result 2 for first 30 tasks

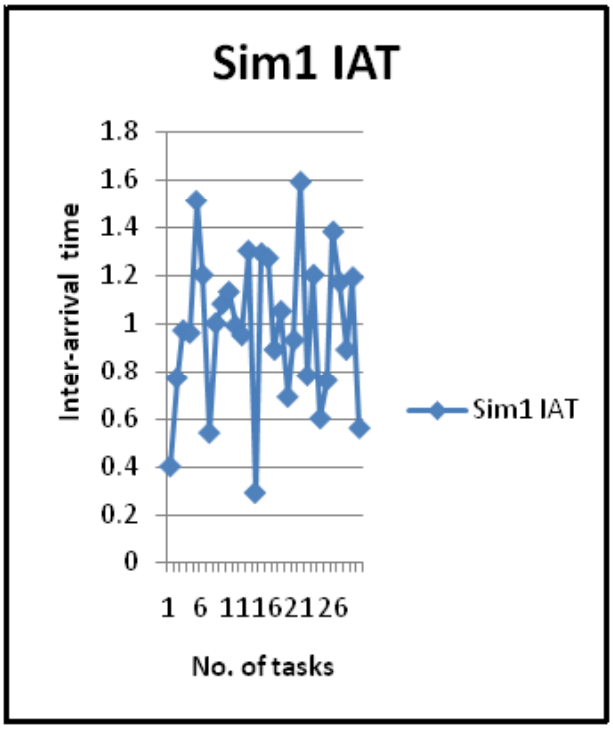

Fig. 4: Simulation result 1 for next 30 tasks

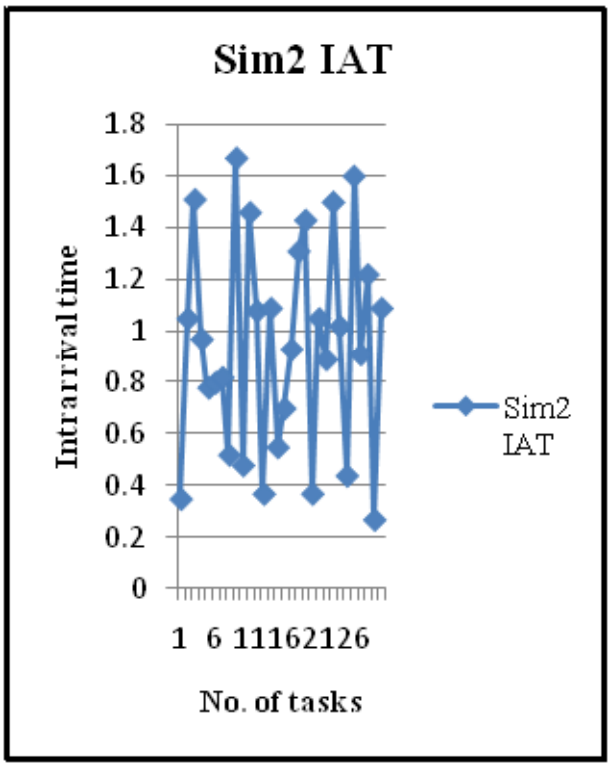

Fig. 5: Simulation result 2 for next 30 tasks

Table 3: Inter arrival time for first next 30 tasks

No. of tasks	Sim1 IAT	Sim2 IAT	Sim3 IAT	Sim4 IAT	Sim5 IAT	Sim6 IAT	Sim7 IAT	Sim8 IAT	Sim9 IAT	Sim10 IAT
1	0.83	0.46	0.77	0.72	0.80	0.95	0.45	0.75	0.92	0.10
2	0.85	1.44	1.57	1.68	1.03	0.35	0.93	0.82	0.95	1.29
3	1.09	1	1.01	0.74	0.56	1.5	1.54	0.77	1.08	1.37
4	0.93	0.96	0.8	0.72	1.32	1	0.81	0.91	0.76	0.55
5	1.5	0.25	0.64	1.16	0.61	0.9	0.76	0.98	0.88	1.14
6	0.98	1.51	0.9	1.13	0.88	0.52	1.41	1.57	1.45	1.3
7	1.06	0.94	1.17	0.72	1.09	1.61	0.84	0.31	1.1	0.45
8	0.82	0.75	1.37	1.41	1.49	0.91	0.52	1.77	1.13	0.99
9	1.31	1.68	0.58	0.99	1.11	0.44	1.30	0.26	0.33	1.31
10	0.95	0.48	1.38	1.11	0.39	1.07	1.03	1.18	1.38	1.21
11	0.86	1.37	1.21	1.03	0.94	1.23	0.88	1.58	1.29	1.2
12	0.87	0.63	0.96	1.01	1.44	1.41	1.40	0.36	0.78	0.28
13	1.22	1	0.36	0.55	0.84	0.61	0.75	1.63	1.15	0.95
14	0.34	1.47	0.9	1.35	0.67	0.76	0.92	0.32	0.57	1.72
15	1.03	0.94	1.36	0.3	1.22	1.01	0.79	1.38	0.98	1.05
16	1.18	0.49	1.18	1.16	0.82	1.54	1.38	0.63	1.48	1.09
17	1.36	0.76	1.04	1.72	1.22	0.79	0.57	0.97	0.24	0.99
18	0.83	1.12	0.73	0.35	1.32	0.78	1.22	1.55	1.16	1.52
19	0.82	0.86	0.93	1.43	0.98	0.93	1.27	0.8	1.43	0.91
20	1.54	0.78	1.43	1.21	0.86	1.3	1.71	1.24	1.33	1.2
21	0.61	0.75	0.53	0.65	1.29	1.24	1.35	0.72	0.82	0.83
22	0.95	1.05	0.96	1.23	0.45	0.65	0.97	1	1.21	0.6
23	0.77	0.81	1.03	0.7	1.17	1.14	0.97	1.2	0.48	1.02
24	1.63	0.69	1.44	1.54	0.92	0.88	1.08	0.65	1.48	1.23

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

25	0.25	1.78	1.04	0.51	1.68	1.08	0.43	0.92	0.93	1.13
26	1.8	0.82	1.14	0.63	0.9	1.2	1.56	1.23	0.27	0.81
27	0.67	0.73	1.14	1.2	0.83	0.64	0.56	1	1.57	1
28	0.49	1.14	0.26	0.89	0.71	0.86	1.33	1.07	1.12	1
29	0.99	0.69	1.66	1.19	0.89	1.08	0.58	0.65	0.86	0.84
30	1.5	1.81	0.74	1.09	1.02	0.9	0.96	1.06	0.6	0.91

Table 4: Inter arrival time for next 30 tasks

No. of tasks	Sim1 IAT	Sim2 IAT	Sim3 IAT	Sim4 IAT	Sim5 IAT	Sim6 IAT	Sim7 IAT	Sim8 IAT	Sim9 IAT	Sim10 IAT
1	0.4	0.35	0.80	0.2	0.1	0.21	0.89	0.78	0.65	0.56
2	0.77	1.05	0.9	1.2	1.32	1.42	0.21	0.94	1.23	1.18
3	0.97	1.51	1	0.91	0.96	0.94	1.45	0.45	0.45	0.46
4	0.96	0.97	0.71	0.99	1.52	0.84	1.05	0.98	1.19	1.5
5	1.51	0.78	1.34	1.47	0.59	1.29	1.03	1.65	0.58	0.57
6	1.2	0.8	1.08	0.59	1.1	1.17	0.61	0.34	1.14	1.63
7	0.54	0.82	0.67	1.55	1.34	0.72	1.67	0.86	1.03	0.56
8	1	0.52	1.38	0.53	0.98	1.72	0.88	1.21	0.91	1.12
9	1.08	1.67	0.24	0.9	0.64	0.9	0.58	1.8	1.28	1.17
10	1.13	0.48	1.08	1.62	0.9	0.89	1.18	0.23	0.92	1.03
11	0.99	1.46	1.21	0.6	1.42	0.95	1.19	1.11	1.12	0.85
12	0.95	1.08	1.19	0.72	0.35	1.18	0.97	1.16	0.9	0.59
13	1.3	0.37	0.76	1.25	1.46	0.28	1.26	0.82	1.07	1.48
14	0.29	1.09	1.4	0.63	1.05	1.83	0.17	1.15	0.63	0.81
15	1.29	0.55	0.62	1.71	0.99	0.76	1.47	0.89	1.65	0.86
16	1.27	0.7	1.19	0.93	0.54	0.53	1.32	1.24	0.97	1.48
17	0.89	0.93	1.07	1.03	1.18	0.97	0.97	1.1	0.84	1.06
18	1.05	1.31	1.2	0.46	1.16	0.8	0.22	1.09	1	0.2
19	0.69	1.43	0.52	1.1	1	1.76	1.8	0.48	1.31	1.42
20	0.93	0.37	1.51	1.51	0.9	0.68	0.6	1.03	0.23	0.67
21	1.59	1.05	1.1	0.52	0.74	0.47	0.67	1.68	1.85	1.19
22	0.78	0.89	0.99	1.07	1.34	1.53	1.3	0.57	0.45	0.91
23	1.2	1.5	0.94	1.12	1.33	0.48	0.74	0.95	0.97	1.22
24	0.6	1.02	0.96	1.2	0.53	0.96	1.49	1.34	0.8	1.42
25	0.76	0.44	0.35	0.74	1.48	1	1.25	1.02	1.22	0.73
26	1.38	1.6	0.96	1.14	0.62	1.49	0.89	1.22	1.46	0.74
27	1.17	0.91	1.74	0.88	1.14	1.05	1.05	0.49	0.34	1.37
28	0.89	1.22	0.54	1.41	0.64	1.15	0.88	1.26	1.21	0.32
29	1.19	0.27	1.01	0.9	1.61	1.09	0.63	0.56	0.9	1.15
30	0.56	1.09	1.4	0.9	0.97	0.48	0.88	1.95	1.07	1.44

Scheduling approach-2 is used for general task sets. In this approach-2 first, we initialize the grid simulation package that allows modelling simulation of entities. Then we create grid resource with no. of machines, millions of instruction per second rating, processing elements, operating system and architecture. When we create grid resource with no of machines we need to create object of machine list, we create machine with its id, millions of instruction per second rating and processing elements. In our experiments, we use maximum processing elements 2, number of grid resources 10; machines per grid 100, maximum millions of instruction per second 400. The performances of scheduling approach-2 better when the number of processor is greater, then tasks are successfully assigned among processor.

VI. Conclusion

Two approaches are developed a semi-partitioned scheduling approach for scheduling real-time sporadic tasks on multi-core platform based under RMS policy. We used harmonic property for periodic tasks. Harmonic property is task with period being integer multiples of each other. To improve the schedulability we take harmonic relationships among the tasks. Scheduling approach-1 for task sets with utilization is no more than 0.5 and

scheduling approach-2 general task with utilization is no more than 1. In first approach, we used worst-fit strategy i.e. we select the processor with minimal total utilization of task that assigned to it. Since accommodate capacity is increased. In second approach, we used grid simulation toolkit. From obtained results it is found that all the generated tasks are successfully scheduled for first 20 *simulation runs.*

References

[1]. *C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J. ACM, 20:46-61, Jan, 1973.*

[2]. *S. Chaudhry, R. Cypher, M. Ekman, M. Karlsson, A. Landin, S. Yip, H. Zeffer, and M. Tremblay. Rock: A High Performance Sparc CMT Processor. IEEE Micro, 29(2):6–16, Mar. 2009.*

[3]. *B. Andersson and J. Jonsson. The Utilization Bounds of Partitioned and Pfair Static-Priority Scheduling on Multiprocessors Are 50%. In Euromicro Conference on Real-Time Systems (ECRTS), Pages: 33 - 40, Jul. 2003.*

[4]. *B. Andersson, S. Baruah, and J. Jonsson. Static-Priority Scheduling on Multiprocessors. In Proc. IEEE Real-Time Systems Symposium (RTSS), Pages: 193 - 202, Dec. 2001.*

[5]. *B. Anderson. Global Static-Pririty Prreemptive Multiprocessor Scheduling with Utilization Bound 38%. In Proc. ACM International Conference on Priciples of Distributed Systems (OPODIS), volume 5401, pages 73-88, 2008.*

[6]. *S. Kato and N. Yamasaki. Semi-Partitioned Fixed-Priority Scheduling on Multiprocessors. In IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Pages: 23 – 32, Apr. 2009.*

[7]. *S. Kato and N. Yamasaki. Portioned Static-Priority Scheduling on Multiprocessors. In IEEE International Symposium on Parallel and Distributed Processing (IPDPS), Pages: 1 - 12, Apr. 2008.*

[8]. *K. Lakshmanan, R. Rajkumar, and J. Lehoczky. Partitioned Fixed Priority Preemptive Scheduling for Multi-core Processors. In Euromicro Conference on Real-Time Systems (ECRTS), Pages: 239 - 248 , Jul. 2009*

[9]. *J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. In Handbook on Scheduling Algorithms, Methods, and Models. Chapman Hall/CRC, Boca, 2004.*

[10]. *Sudarshan K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. Operations Research, 26(1):127–140, 1978.*

[11]. *J.H. Anderson, V. Bud, and U.C. Devi. An EDF-Based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In Proc. Euromicro Conference on Real-Time Systems (ECRTS), Pages: 199 - 208, Jul. 2005.*

[12]. *A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Is Semi-Partitioned Scheduling Practical? In Euromicro Conference on Real-Time Systems (ECRTS), Pages: 125 - 135, July. 2011.*

[13]. *N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-Priority Multiprocessor Scheduling: Beyond Liu and Layland's Utilization Bound. In WiP Real-Time Systems Symposium (RTSS), pages: 165-174, Dec. 2010.*

[14]. *N. Guan, M. Stigge, W. Yi, and G. Yu. Parametric Utilization Bounds for Fixed-Prioirity Multiprocessor Scheduling. In IEEE International Symposium on Parallel and Distributed Processing (IPDPS), Pages: 261 - 272, May 2012.*

[15]. *M.-J. Jung, Y. R. Seong, and C.-H. Lee. Optimal RM scheduling for simply periosic tasks on Uniform Multiprocessors. In Proc. ACM International Conference on Hybrid Information Technology (ICHIT), volume 321, pages 383–389, Aug. 2009.*

[16]. *M. Fan and G. Quan. Harmonic semi-partitioned scheduling for fixed priority real-time tasks on multi-core platform. In Design, Automation Test in Europe Conference Exhibition (DATE), pages 503 –508, Mar. 2012.*

[17]. *C.-C. Han and H.-Y. Tyan. A Better Polynomial-Time Schedulability Test for Real-Time Fixed-Priority Scheduling Algorithms. Real-Time Systems Symposium (RTSS), IEEE International 0:36–45, 1997.Dec. 1997.*

[18]. *N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-Priority Multiprocessor Scheduling with Liu and Layland's Utilization Bound. In IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Pages: 165 - 174, Apr. 2010.*

[19]. *D. Muller. Accelerated Simply Periodic Task Sets for RM Scheduling In Embedded Real Time Software and Systems, May 2010.*

[20]. *M. Fan and Gang Quan. Harmonic-fit partitioned scheduling for fixed-priority real-time tasks on the multiprocessor platform. In Embedded and Ubiquitous Computing (EUC), IFIP 9th International Conference on, pages 27–32, Oct. 2011.*

[21]. *A. Kandhalu, K. Lakshmanan, Junsung Kim, and R. Rajkumar. pCOMPATS: Period-Compatible Task Allocation and Splitting on Multicore Processors. In IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Pages: 307 – 316, Apr. 2012.*

[22]. *W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor System-on- Chip (MPSoC) Technology. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(10):1701 –1713, Oct. 2008.*

[23]. *B. Anderson and E. Tovar. Multiprocessor scheduling with few preemptions. In Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pages-322-334, 2006.*

[24]. *B. Andersson, K. Bletsas, and S. Baruah. Scheduling Arbitrary- Deadline Sporadic Task Systems on Multiprocessors. In IEEE Real- Time Systems Symposium (RTSS), Pages: 385 - 394, Dec. 2008.*

[25]. *S. Kato and N. Yamasaki. Real-Time Scheduling with Task Splitting on Multiprocessors. In IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Pages: 441 - 450, Aug. 2007.*

[26]. *S. Kato and N. Yamasaki. Portioned edf-based Scheduling on Multiprocessors. In Proceedings of the International Conference on Embedded Software. 139–148.Pages: 1 - 12, 2008.*

[27]. *Y. Zhang, N. Guan, and W. Yi. Towards the Implementation and Evaluation of Semi-Partitioned Multi-Core Scheduling. In Bringing Theory to Practice: Predictability and Performance in Embedded Systems, volume 18 of*

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

OpenAccess Series in Informatics (OASIcs), pages 42–46, 2011.

[28]. R. I. Davis, and A. Burns, "A survey of hard real-time scheduling for multiprocessor system,"ACM Computing survey, 43(4):1-44, Oct. 2011.

[29]. T.-W. Kuo and A.K. Mok. Load Adjustment in Adaptive Real-Time Systems. In Proc. IEEE Real-Time Systems Symposium (RTSS), Pages: 160 - 170, Dec. 1991.

[30]. Lauzac, R. Melhem, and D. Moss´ e. An Efficient RMS Admission Control and Its Application to Multiprocessor Scheduling. In IPPS/SPDP Parallel Processing Symposium, Pages: 511 – 518, Mar. 1998.