Partition Oriented Affinity-Aware ERfair Scheduler

Journal:	*Transactions on Computers*
Manuscript ID:	TC-2012-07-0498
Manuscript Type:	Brief Contribution
Keywords:	D.4.1.e Scheduling < D.4.1 Process Management < D.4 Operating Systems < D Software/Software Engineering, D.4.1.c Multiprocessing/multiprogramming/multitasking < D.4.1 Process Management < D.4 Operating Systems < D Software/Software Engineering, D.4.7.e Real-time systems and embedded systems < D.4.7 Organization and Design < D.4 Operating Systems < D Software/Software Engineering

Partition Oriented Affinity-Aware ERfair Scheduler

Anshuman Tripathi, Arnab Sarkar, and P. P. Chakrabarti *Sr. Member, IEEE*

Abstract—Limiting overall scheduling overheads (primarily combining task selection overheads and task migration overheads) is of utmost importance in today's resource constrained multiprocessor real-time systems because it provides premium spare processor bandwidth that may be useful in various situations. This paper presents *Partition Oriented Affinity-Aware ERfair Scheduler (POAES)*, a hard real-time ERfair multiprocessor scheduler that incurs significantly lower overheads compared to all the known ERfair schedulers under most workload scenarios. POAES employs a two level scheduling technique in which the outer level consists of a load balancer with an online partitioning / merging mechanism that maintains the work load mapped onto disjoint groups of processors. At the inner level, within each task-processor group a task-to-processor affinity aware ERfair scheduler is used to execute tasks in ERfair manner while simultaneously attempting to minimize inter-processor task migrations. Experimental results show that POAES incurs much lower overall scheduling related overheads as compared to the current migration aware ERfair schedulers in most scenarios.

Index Terms—Proportional fairness, ERfair scheduling, Real time scheduling, Task migration, Task partitioning, Low overhead

1 Introduction

Guaranteeing responsiveness to all time-critical events in software intensive multi-processor resource constrained environments has been the ever cherished goal of the real-time systems community for architectures ranging from big multi-cluster servers to multi-core high-performance embedded systems like hand-held devices. Additionally, because many of these systems run a mix of different independent applications such as real-time audio processing, streaming video, interactive gaming, web browsing, telnet, etc., often their scheduling requirements not only demand meeting deadlines, but CPU reservation to ensure a minimum Quality of Service (QoS). These demands are generally of the form *reserve X units of time for application A out of every Y time units*.

Attempts to satisfy such conflicting demands led to the development of the global ERfair scheduling methodology [1]. ERfair scheduling is a work-conserving version[1] of the proportional fair scheduling methodology (*PF* [2], *PD* [3], *PD*^{2} [4]). Given a set of periodic tasks $\{T_1, T_2, T_3, \ldots, T_n\}$, with each task T_i having a computation requirement of e_i time units, required to be completed within a period of p_i time units from the start of the task, ERfair schedulers need to manage their task allocation and preemption in such a way that not only are all task deadlines met, but also each task is executed at a consistent rate proportional to its weight $wt_i = \frac{e_i}{p_i}$. ERfair algorithms generally consider discrete time lines and divide the tasks into equal sized subtasks such that a single subtask may be executed within the time period of a slot. ERfairness is formally defined in terms of a parameter called *lag* [4]. At any given time t, $lag(T_i,t)$ denotes the difference between the amount of time that has been actually allocated to a task T_i and the amount of time that should have been ideally allocated to it in the interval $[0,t)$ (denoted by $allocated(T_i,t)$). Thus, $lag(T_i,t) = (e_i/p_i) * t - allocated(T_i,t)$. A scheduling algorithm is ERfair iff $\forall(t,T_i),\ 0 \leq |lag(T_i,t)| < 1$. Each subtask st_{ij} of a task T_i has a *pseudo-deadline* time pd_{ij} before which it must complete st_{ij}. pd_{ij} is defined as:

$$pd_{ij} = \lceil \frac{j * p_i}{e_i} \rceil \quad (1)$$

Thus, ERfair's salient features include its ability to meet all hard real-time deadlines, allowing full CPU resource utilization and also guaranteeing a specified execution rate for all applications. However, along with all these desirable features, an efficient practical scheduler must also keep scheduling overheads like its *own task selection time* and *inter-processor task migrations* to a minimum. Limiting these overheads is of paramount importance especially in resource constrained real-time systems because it provides premium spare processor bandwidth that may be useful in various situations like: completing tasks during transient overloads, implementing power management strategies like processor slowdown and processor shutdown, fault tolerance, executing non-real-time and aperiodic tasks on a best effort basis, etc. Simulation based experiments showed us that the global Basic-ERfair scheduler severely lacks in this front and may consume almost 30% of the allocated time for task executions even on a moderately large sized 16 processor system (Refer table 1).

Traditionally, a completely partitioned approach have generally been adopted to avoid migrations. Here, once a task is allocated a processor, it is exclusively executed on it [5], [6]. Partitioning also has the added advantage of reducing the average scheduling complexity at each time slot because unlike global schedulers, separate schedulers that run independently in parallel may be employed for each partition. In this respect,

Authors are with the Computer Science & Engineering Department, Indian Institute of Technology, Kharagpur, WB 721 302, India. Email: {anshu.g546, arnabs}@gmail.com, ppchak@cse.iitkgp.ernet.in

1. A work conserving scheduler never allows a processor to idle when there are ready tasks available to run

global schedulers generally have to incur the extra overhead of communicating to all processors the tasks they should execute at each time slot.

One of the major problems faced by partitioning is that no more than half the system capacity may be utilized in order to ensure that all deadlines are met in the worst case [7]. However, this worst-case condition may be relaxed either by bounding the maximum weight of any individual task under a certain value [8] or by allowing individual tasks to be split across multiple processors [9]. Levin et. al. in [10] and Kimbrel et. al. in [11] discuss migration aware algorithms which provides a trade-off between the amount of deviation from perfect scheduling fairness and the number of migrations.

Lately *Sticky-ERfair* [12] and *Partition Oriented ERfair Scheduler (POES)* [13], two optimally fair (ERfair) scheduling approaches with reduced migrations and / or context-switches have been developed. Sticky-ERfair follows an overall global scheduling policy. The algorithm controls migrations by keeping track of the processor where a task last executed and by using the system's slack capacity in underloaded ERfair systems. Employing these techniques, it attempts to execute on a given processor its most recently executed task such that this execution do not lead to a future ERfairness violation. On the other hand, POES follows a semi-partitioned approach switching towards higher global-ism with increasing load and vice-versa. Both these techniques have been shown to achieve handsome gains in terms of migration overheads with Sticky-ERfair being more effective at high workload conditions and POES being more effective under lower workloads.

However, measurement and estimation of the actual gains in terms of the reduction in overall overheads (combining scheduling complexity and migration related overheads) for both these algorithms have been lacking. This paper does a comprehensive simulation based experimental analysis and comparison of the total scheduling related overheads of Basic-ERfair, Sticky-ERfair and the POES algorithms. The analysis considers different workload conditions, varying number of processors and different values for the overhead of a single migration (this depends on the system architecture), etc. We observed that although Sticky-ERfair achieves high gains in terms of migration overheads even under heavy workloads, the overall gains obtained may be quite moderate due to high task selection times. On the other hand, with a lower average task selection complexity, POES provides very high reductions in overall overheads under low to moderately heavy system load scenarios when it is able to partition the task set into disjoint processor groups. However, one of the drawbacks of this algorithm is that it quickly becomes as bad as Basic-ERfair when the system becomes global under high workloads.

Using the insights gained through these observations, this paper also presents the *Partition-Oriented Affinity-Aware ERfair Scheduler (POAES)*, that aims to obtain high reductions in total overall overheads under all workload conditions by effectively leveraging and combining the benefits of both the global (Sticky-ERfair) and semi-partitioned (POES) scheduling approaches mentioned above. As an example of the achieved gains, the right-most column for 16 processors in table 1 shows that on a system with time-slot size of 1 msec, while Basic-ERfair, Sticky-ERfair and POES may consume upto 31%, 14% and 6% of a time-slot as scheduling related overheads, POAES incurs only about 1%.

We first describe the working of the POAES algorithm along with illustrative examples in the next section. Section 3 presents the detailed experimental analysis results. We discuss and comparatively analyze the migration overheads of the different algorithms under various scenarios in section 3.1. Then, results for the task selection overheads of these algorithms have been discussed in section 3.2. Finally, in section 3.3 we present a comparative analysis the overall scheduling related overheads. We conclude in section 4.

2 The POAES Algorithm

POAES actually works at two levels. The outer level consists of a load balancer with an online partitioning / merging mechanism (similar to the POES algorithm [13]) that retains the optimal schedulability of a fully global scheduler by merging processor groups as resources become critical while using partitioning for fast scheduling at other times. The principal objective is to remain only just as global at any given instant of time as is necessary to maintain ERfair schedulability. At the inner level, for any processor V_i within a particular task-processor group, a task-to-processor affinity aware ERfair scheduler (similar to Sticky-ERfair [12]) allocates the most recently executed ready task that previously executed on V_i (thus restricting migrations and preemptions) in such a way that this allocation does not cause any ERfairness violations in the system at any time during the schedule length. We now describe the online load balancing mechanism with an illustrative example.

The Load Balancer (Outer Level): Given a set of n periodic tasks $\{T_1, T_2, ..., T_n\}$ to be scheduled in a system of m processors $\{V_1, V_2, ..., V_m\}$, the task and processor sets are partitioned into k disjoint subsets or groups ($\{\tau 1, \tau 2, ..., \tau k\}$ and $\{\rho 1, \rho 2, ..., \rho k\}$ respectively) at any instant during the schedule length. The tasks in group τi are allowed to execute and migrate only within its corresponding processor group ρi. If $\{T_{\tau i_1}, T_{\tau i_2}, ..., T_{\tau i_{|\tau i|}}\}$ denote the tasks in group τi and $\{V_{\rho i_1}, V_{\rho i_2}, ..., V_{\rho i_{|\rho i|}}\}$ denote the processors in group ρi, then such a partition is feasible, only if,

$$\forall_{i=1}^{k} \sum_{j=1}^{|\tau i|} wt_{\tau i_j} \leq |\rho i| \quad (2)$$

At lower workloads, a complete partition is usually always feasible and the number of groups k is equal to the number of processors m. If a new task arrives which cannot be accommodated into any task group, two or more task-processor groups are *merged* so that the newly formed group will be able to feasibly accommodate the new task. If a task departs from any task-group and the corresponding processor-group contains more than one processor, a *split* of this group is attempted such that the feasibility condition in equation 2 is not violated.

Example: Consider ($n =$) 5 tasks, T_1, T_2, T_3, T_4 and T_5 (having weights $wt_1 = 9/10$, $wt_2 = 4/5$, $wt_3 = 5/9$, $wt_4 = 1/2$ and $wt_5 = 1/5$) to be scheduled on ($m =$) 3 processors V_1,

V_2 and V_3. Let the arrival times of these tasks be 0, 0, 0, 10, 20. Therefore, at time $t = 0$, the 3 ready tasks T_1, T_2 and T_3 get partitioned into the 3 available processors V_1, V_2 and V_3 respectively. Each processor forms a separate group. Processor V_3 having highest remaining capacity $(4/9)$ forms group ρ1. Similarly, processors V_2 and V_1 form groups ρ2 and ρ3 respectively. Figure 1 shows this situation.

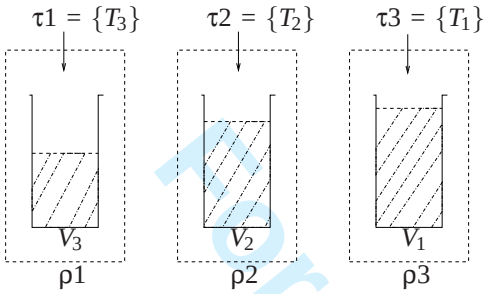

Fig. 1. Time $t = 0$; each processor individually forms a separate group

A new task T_4 arrives at $t = 10$. Task T_4 having weight $wt_4 = 1/2$ cannot be accommodated into any existing group. Hence, groups ρ1 and ρ2 are merged to accommodate T_4. After merging, we get two groups: i. ρ1 consisting of processors V_2 and V_3 with tasks T_2, T_3 and T_4 and ii. ρ2 consisting of processor V_1 and task T_1 allocated to it. Figure 2 depicts this.

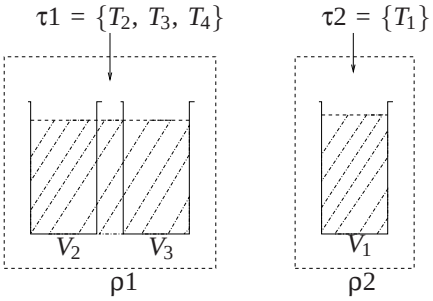

Fig. 2. Time $t = 10$. Two task-processor groups get merged.

Task T_5 arrives at time $t = 20$. Now, all the processors are merged into a single group to accommodate T_5. This situation is depicted in figure 3.

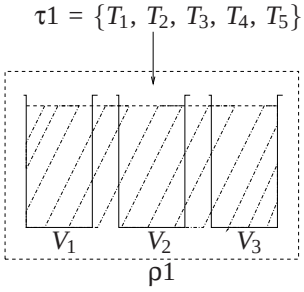

Fig. 3. Time $t = 20$. The system becomes completely global.

Now, let the task T_4 complete execution and leave the system at time (say) $t = 350$. We again obtain a fully partitioned system as shown in figure 4.

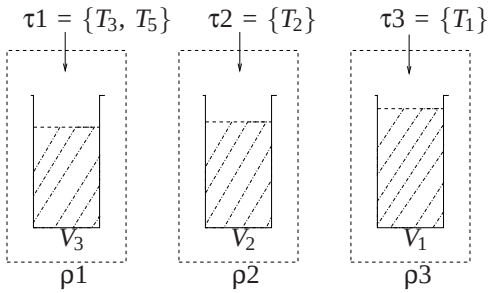

Fig. 4. Time $t = 350$. A fully partitioned system is obtained again.

The Affinity-Aware Scheduler (Inner Level): After the load balancing step at the outer level, a set of task-processor groups is obtained. Each task-processor group (consisting of 1 or more processors) acts as a separate global scheduling system on its own and employs a separate Affinity aware ERfair scheduler similar to Sticky-ERfair [12] at the inner level that attempts to minimize migrations and preemptions by: (I) Keeping track of the processor where a task last executed, and (II) Utilizing task over-allocations in under-loaded ERfair systems.. Given a task-processor group containing m' processors (say), the Affinity-Aware ERfair scheduler selects the most urgent m' tasks (those m' tasks whose pseudo-deadlines are earliest) similar to the Basic-ERfair algorithm. However, for any processor say V_i within the group, the affinity aware ERfair scheduler may postpone the execution of one of these selected tasks and replace it by the most recently executed ready task that previously executed on V_i (thus restricting migrations and preemptions) in such a way that this allocation does not cause any ERfairness violations in the system at any time during the schedule length. To ensure ERfairness, the algorithm defines a new parameter called *deadline of postponement* for each sub-task of a task. The deadline of postponement of the j^{th} subtask of a task T_i denotes the time slot upto which the execution of the j^{th} subtask of T_i may be safely postponed (suspended from ready state) without any possibility of the system violating ERfairness. It is given by:

$$\phi_{ij} = pd_{ij} - \lfloor \frac{p_i}{e_i} \rfloor - 2 \quad (3)$$

Affinity-Aware ERfair guarantees ERfairness by not allowing a task to be postponed when there exists ready tasks whose deadlines of postponement have been crossed.

Example: Consider 5 tasks, T_1, T_2, T_3, T_4 and T_5 to be scheduled in a group of 3 processors V_1, V_2 and V_3. Let the execution requirements and periods of these tasks be as follows: $e_1 = 36$, $e_2 = 40$, $e_3 = 50$, $e_4 = 64$, $e_5 = 72$ and periods $p_1 = p_2 = p_3 = 120$, $p_4 = p_5 = 80$. Hence, the wt_i $(= e_i/p_i)$ values of these tasks are $wt_1 = 3/10$, $wt_2 = 1/3$, $wt_3 = 5/12$, $wt_4 = 4/5$ and $wt_5 = 9/10$. At $t = 0$, the pseudo-deadlines (refer equation 1) and the deadlines of postponement (refer equation 3) of the first sub-task of these tasks will be: $pd_{11} = 3$, $pd_{21} = 2$, $pd_{31} = 2$, $pd_{41} = 1$, $pd_{51} = 1$ and $\phi_{11} = -2$, $\phi_{21} = -3$, $\phi_{31} = -2$, $\phi_{41} = -2$, $\phi_{51} = -2$. In the first time-slot,

the tasks T_5, T_4 and T_3 will be selected for execution because these tasks are most urgent and all of them are beyond their deadlines of postponement.

Now, let us consider another arbitrary time instant say, $t = 225$. Let, the tasks 1 through 5 have executed last on processors V_1, V_1, V_1, V_2 and V_3 respectively. Let the ϕ_i values for the next subtasks of these tasks respectively be 222, 224, 226, 226, 227 and their pd_i values be 230, 230, 233, 233 and 236. Our Affinity-Aware algorithm will first choose the three most urgent tasks, that is T_1, T_2 and T_3 in the same manner as Basic-ERfair chooses. After this, T_1 is alloted processor V_1. As $t = \phi_2 = 225$, T_2 must be executed in the current time slot to avoid possible ERfairness violation. T_2 will incur a migration. However, execution of T_3 may be postponed because the ϕ_i values of all the remaining ready tasks are greater than 225. T_4 is executed on V_2 in place of T_3, thus saving one migration with respect to ERfair in this time slot. Finally, T_2 is allotted processor V_3.

In the next section, we experimentally evaluate the performance of the POAES algorithm and compare it against the Basic-ERfair [1], Sticky-ERfair [12] and POES [13] algorithms. The evaluation methodology is based on simulation experiments using randomly generated task sets.

3 Experiments and Results

The simulation based experiments conducted to compare the performance of POAES against Basic-ERfair, Sticky-ERfair and POES algorithms measure both the number of migrations per time slot and total execution times of the algorithms on different types of generated data sets. All the results presented herein have been averaged over 100 simulations each having a schedule length of 100000 time slots. The ultimate objective of all these experiments is to obtain for each of the mentioned algorithms a measure of their overall scheduling overheads combining their average task selection and task migration times at each time slot and hence be able to arrive at a reasonably good comparative estimate of how fast would these algorithms execute in practice on actual workloads. Throughout the simulations, a separate check was kept on the *lag* of every task in the system to verify ERfairness of the algorithm; *no violations were detected.*

The experimental framework consists of randomly generated task sets with hypothetical task weights (e_i/p_i) and periods (p_i). The task weights have been generated using a uniform distribution within the range $[\mu_w - 0.05, \mu_w + 0.05]$ where μ_w is the required average individual task weight. Periods of the tasks are taken from a normal distribution with $\mu = 4000$ and $\sigma = 3500$. Data sets representing systems with various average individual task weights, average total system workloads and different number of processors were considered in the experimental analysis.

3.1 Migration Measurement Results

The number of inter-processor task migrations suffered by POAES and the three other ERfair algorithms mentioned above have been measured by running them on 100 different instances of each data set type. Figures 5, 7 and 8 depicts the plots of the number of migrations per time slot (on the y-axis) with respect to variation in the total system load percentage (L) (system load is computed as the fraction of sum of the weights of all the tasks to the number of processors in the system), average individual weight (μ_w) of tasks and number of processors (m) respectively (on the x-axis).

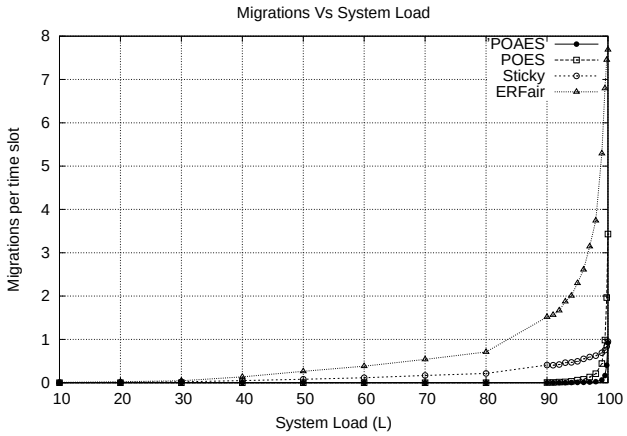

Fig. 5. Processors $(m) = 10$, Avg. Individual Task Weight $(\mu_w) = 0.3$

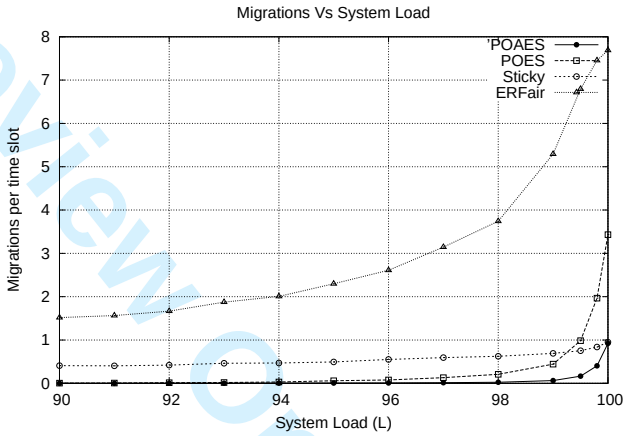

Fig. 6. $[90\% - 100\%]$ System Load (Processors $(m) = 10$, Avg. Individual Task Weight $(\mu_w) = 0.3$)

Figure 5 shows the plot of the number of migrations per time slot as the average system load is varied between 10% and 100% on a 10 processor system with mean individual weight of tasks being $\mu_w = 0.3$. It may be observed from this figure that all the algorithms exhibit a similar nature. Figure 6 which depicts the same but a magnified view of the plots in figure 5 for the range 90% to 100% load, clearly shows that POAES incurs negligible migrations until the system load reaches as high as $\approx 98\%$ and although migrations increase rapidly after this point, POAES shows better performance as system capacity gets more and more crunch. It may however be noted that at full system load (100%), with no slack available for task execution postponement and the system becoming completely global, performance of all algorithms become almost equally

poor and the number of migrations become proportional to the size (number of processors) of the system for all algorithms.

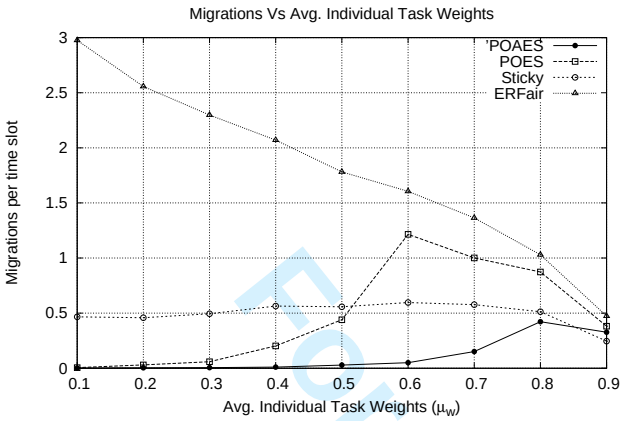

Fig. 7. Processors $(m) = 10$, System Load $(L) = 95\%$

Figure 7 shows the plots of migrations per time slot when the average individual weight of tasks is varied between 0.1 to 0.9. A 95% loaded 10-processor system was considered. A prominent feature in this graph is that while the partition oriented schedulers POES and POAES show distinct maximas at around 0.6 and 0.8 on the x-axis respectively, the global schedulers show a steady gradient with a gradual fall at higher average task weight values. The occurrence of this maxima may be attributed to the existence of two opposing factors acting concurrently. The first factor is that when the mean individual task weight increases with the system load being fixed at 95%, the total number of tasks in the system decrease contributing to a decrease in the number of migrations. The second factor is the reduction in the total number of partitions caused by higher mean task weights and this contributes to increased migrations.

A closer observation at the Sticky-ERfair plot reveals that the second factor is also feebly at work in its case and this causes migrations to slightly increase with increasing task weights with a distinct dip at very heavy mean task weight values. This happens because although Sticky-ERfair follows an overall global methodology, it exhibits micro-level partition orientation in its attempt to maximize the continuous length of time for which a task executes on a particular processor. By employing a POES like partitioning approach at the outer level with a Sticky-ERfair like scheduling mechanism at the inner level, POAES combines the benefits of both and is thus able to weaken the effect of the second factor in reducing the total number of partitions upto a much later stage and allows the maxima for POAES to be obtained at a point much further beyond the maxima for POES.

It may further be observed from this graph that the migrations per time slot for all the scheduling algorithms tend to converge as the mean task weights approach unity. This is due to the fact that on a 95% loaded system of 10 processors with average individual task weights beyond 0.9, the partitioning algorithms fail to identify feasible partitions and the systems become almost completely global. Hence, POES and POAES exhibit natures similar to Basic-ERfair and Sticky-ERfair respectively. One may notice here that Sticky-ERfair briefly outperforms POAES in the region beyond mean task weights ≈ 0.85. This happens because with such high individual task weights, multiple partitions if formed in POAES, may often consist of loads more than 95% in each partition. As depicted in figure 6, as migrations of Sticky-ERfair increase very rapidly in the $95\% - 100\%$ load range, it is possible for the POAES algorithm to perform poorer due of "higher than average" system load partitions.

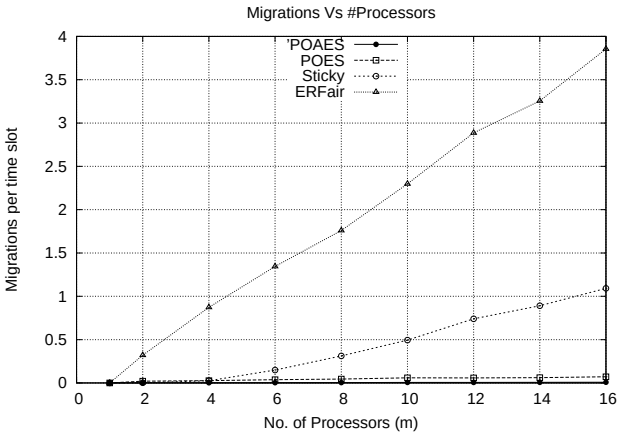

Fig. 8. Avg. Individual Task Weight $(\mu_w) = 0.3$, System Load $(L) = 95\%$

Figure 8 plots the variation in the average number of migrations per time slot versus the number of processors for an average task weight of 0.3 and system load of 95%. The obvious benefit of partitioning is clearly evident in this graph. Although all plots show a linear nature, the slope for the increase in the number of migrations per time slot with respect to the number of processors is much higher for the global schedulers as compared to their partition oriented counterparts and this makes partition oriented schedulers more scalable. Performance wise, POAES outperforms all the other three schedulers.

3.2 Time Measurement Results

We have measured the average execution times to estimate task selection overheads for the new algorithm POAES and also the existing algorithms Basic-ERfair, Sticky-ERfair and POES running them on 100 different instances of each data set type with the schedule length being 100000 time slots. Figure 9 depicts the *average task selection time per time slot* with respect to varying number of processors (1 to 16 processors) obtained for data sets having average task weight of 0.5 and system load of 95%. This graph shows that in general the partition oriented schedulers incur much less overheads in terms of the time taken to select tasks at each time slot as compared to their global counterparts. This is due to the fact that in partition oriented systems, the scheduling of each partition occurs concurrently in parallel with the other partitions. In comparison, global schedulers have to incur the extra overhead of communicating to all processors the

#Processors	Scheduler	N_{mig}	$C_{sched}(\mu s)$	$C_{total}(\mu s)$			
				$C_{mig}=1\mu s$	$C_{mig}=10\mu s$	$C_{mig}=50\mu s$	$C_{mig}=100\mu s$
4	POAES	0.0086	3.35	3.36	**3.44**	**3.78**	**4.21**
	POES	0.241	2.53	**2.76**	4.94	14.58	26.63
	Sticky ERFair	0.022	6.04	6.06	6.26	7.14	8.24
	ERFair	0.767	4.03	4.79	11.7	42.38	80.72
8	POAES	0.022	4.79	4.81	**5.01**	**5.89**	**6.99**
	POES	0.323	3.7	**4.02**	6.93	19.85	36.00
	Sticky ERFair	0.410	13.9	14.26	17.95	34.35	54.85
	ERFair	1.46	9.83	11.28	24.42	82.82	155.82
12	POAES	0.025	6.23	6.25	**6.48**	**7.48**	**8.73**
	POES	0.422	4.86	**5.28**	9.08	25.96	47.06
	Sticky ERFair	0.737	22.86	23.60	30.23	59.71	96.56
	ERFair	2.21	16.62	18.83	38.72	127.12	237.62
16	POAES	0.028	7.85	7.88	**8.13**	**9.25**	**10.65**
	POES	0.507	5.61	**6.12**	10.68	30.96	56.31
	Sticky ERFair	1.02	33.03	34.05	43.27	84.23	135.43
	ERFair	2.837	23.00	25.83	51.37	164.85	306.7

TABLE 1
Scheduling costs per time slot (Task weight $(\mu_w) = 0.5$, Load $(L) = 95\%$)

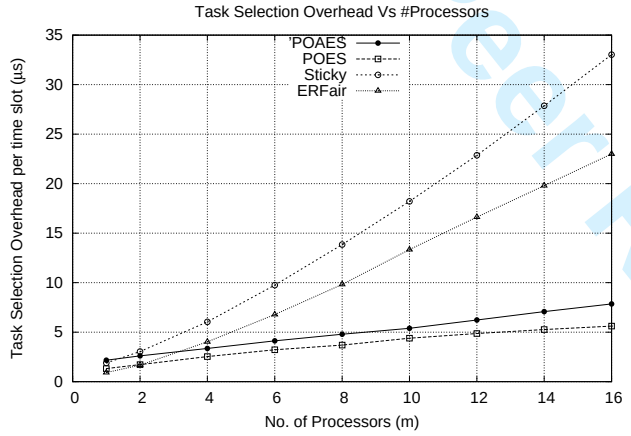

Fig. 9. Avg. Individual Task Weight $(\mu_w) = 0.5$, System Load $(L) = 95\%$

tasks they should execute at each time slot. In the POES or POAES algorithms, the *merge* and *partition* operations form their only fully or partially global parts. It may be noticed that with increase in the number of processors, the difference of execution times between the global and partitioned algorithms widen drastically. POES with a slightly lower intra-partition scheduling complexity (To schedule tasks within each partition, POES uses Basic-ERfair while POAES uses a Sticky-ERfair like methodology.) incurs the least overhead, although being closely followed by POAES.

3.3 An Estimation of Overall Scheduling Overheads

The total scheduling overhead at each time slot (C_{total}) is obtained as the sum of the average task selection time (C_{sched}) and the time spent in task migrations at each time slot (obtained as the product of the average number of migrations per time slot (N_{mig}) and the cost of a single migration (C_{mig})). Thus,

$$C_{total} = C_{sched} + C_{mig} \times N_{mig} \quad (4)$$

The cost of a single migration (C_{mig}) depends heavily on the system architecture and realistic values may typically vary from lower than $1\mu s$ in closely-coupled multi-core systems to more than $100\mu s$ in loosely-coupled multi-processor systems. Table 1 summarizes the average total scheduling overhead results for all the four algorithms evaluated here on data sets having average task weight of 0.5 and workload of 95% for different number processors and different C_{mig} values. It may be noted from the table that POAES appreciably outperforms all the other three algorithms for C_{mig} values $10\mu s$, $50\mu s$ and $100\mu s$. However, in extremely tightly coupled systems ($C_{mig} = 1\mu s$) POES performs marginally better than POAES.

Assuming a time slot size of $\approx 1ms$ (which is a typical value in many of today's real time systems) on a moderately large sized (16 processors) loosely coupled ($C_{mig} = 100\mu s$) real-time multiprocessor system, it may be observed that Basic-ERFair consumes as much as $\approx 30\%$ of a time slot while POAES consumes only about 1%. Thus, POAES is much more scalable and gives premium spare processor bandwidth which may be useful in various scenarios. Examples include, completion of tasks which misbehave at runtime by taking more time than they were stipulated to take, execution of non-real time and aperiodic tasks on a best effort basis along with the real time periodic tasks, implementing power management strategies like processor slowdown and processor shutdown, fault tolerance, etc.

4 Conclusion

We have presented a new partition oriented multiprocessor ERfair scheduling algorithm that attempts to minimize overall scheduling overheads and provide premium spare processor bandwidth which may be of vital importance especially in today's resource constrained real-time embedded systems. We have designed, implemented and evaluated the POAES algorithm. The simulation results are promising.

References

[1] J. Anderson and A. Srinivasan, "Early-release fair scheduling," in *12th Euromicro Conference on Real-Time Systems*, Jun 2000, pp. 35–43.

[2] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[3] S. Baruah, J. Gehrke, and C. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *9th International Parallel Processing Symposium*, Apr 1995, pp. 280–288.

[4] J. Anderson and A. Srinivasan, "Mixed pfair/erfair scheduling of asynchronous periodic tasks," *Journal of Computer and System Sciences*, vol. 68, no. 1, pp. 157–204, Feb 2004.

[5] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms." [Online]. Available: citeseer.ist.psu.edu/601206.html

[6] J. Malkevitch, "Bin packing and machine scheduling." [Online]. Available: http://www.ams.org/samplings/feature-column/fcarc-packings1

[7] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%," in *15th Euromicro Conference on Real-Time Systems*, Jul 2003, pp. 33–40.

[8] J. Lopez, M. Garcia, J. Diaz, and D. Garcia, "Worst-case utilization bound for edf scheduling on real-time multiprocessor systems," in *12th Euromicro Conference on Real-Time Systems*, Jun 2000, pp. 25–33.

[9] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006, pp. 322–334.

[10] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *22nd Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 3–13.

[11] T. Kimbrel, B. Schieber, and M. Sviridenko, "Minimizing migrations in fair multiprocessor scheduling of persistent tasks," *Journal of Scheduling*, vol. 9, no. 4, pp. 365–379, Aug 2006. [Online]. Available: http://dx.doi.org/10.1007/s10951-006-7040-0

[12] A. Sarkar, S. Ghose, and P. P. Chakrabarti, "Sticky-erfair: A task-processor affinity aware proportional fair scheduler," *Real-Time Systems Journal*, vol. 47, no. 4, pp. 356–377, 2011.

[13] A. Sarkar, A. Shanker, S. Ghose, and P. P. Chakrabarti, "A low overhead partition-oriented erfair scheduler for hard real-time embedded systems," *IEEE Embedded Systems Letters*, vol. 3, no. 1, pp. 5–8, 2011.