Optimal Scheduling Algorithms on multiprocessors: A comparative study

[1]Shreya Maolanker, [2]Prof. Padmini G. Kaushik, [3]Nidhi Thakur
Department of Electronics & Telecommunication Engineering
Jawaharlal Darda Institute of Engineering & Technology, Yavatmal, India
[1]168.shreya@gmail.com, [2]padmini.jp@gmail.com, [3]nidhithakur114@gmail.com

***Abstract* - Multicore hardware systems are proving to be more efficient each passing day and so are the scheduling algorithms for these systems. The potential speedup of applications has motivated the widespread use of multiprocessors in recent years. Optimal multiprocessor scheduling algorithms remain a challenge to the researchers. Out of the number of algorithms proposed and analyzed we here compare and examine three of them: the classic global EDF, the optimal P-fair algorithm and a newer LLREF which has worked upon the strengths of P-fair. They are compared in terms of task migrations and required number of scheduler invocations and schedulability of a variety of tasks. Results are verified on the basis of a set of randomly generated tasks.**

Index terms- scheduling algorithms, schedulability, global EDF,PF,LLREF

I. INTRODUCTION

Real-time systems are preliminarily the result of basic but complex user requirements, fundamentally defined by correctness of the systems, timeliness, accuracy, simultaneity and a high degree of predictability. Thus a scheduling algorithm becomes an indispensable part of such systems. It involves the allocation of resources and time to tasks in such a way that most of the performance requirements are met. Use of multiprocessors has increased dramatically due to reduced heat and thermal dissipation. However optimal scheduling algorithms developed for uniprocessor systems did not work satisfactorily for multiprocessors. Scheduling real time tasks consists of two sub problems: task allocation to processors and scheduling tasks on individual processors. Furthermore, there are additional sources of overhead to consider: migrating task overhead and overheads due to scheduler invocations. In this paper, we will examine three scheduling algorithms for uniform multiprocessor systems viz. global EDF, PF and LLREF. We then implement these algorithms in RTSIM, an open-source simulator. Finally, there is a comparative study on the basis of schedulability as well as the number of scheduler invocations and task migrations for each of these algorithms.

II. SCHEDULING ALGORITHMS

A. Global EDF

The Earliest Deadline First (EDF) is one of the oldest scheduling algorithms first described by Liu and Layland[5].Each instance of a task is assigned a priority on the basis of its absolute deadline. Earlier the deadline, higher will be the priority. The algorithm is optimal for uniprocessors when it is used to schedule jobs on a processor as long as preemption is allowed and jobs do not contend for resources. Global EDF is the extension of EDF for multiple processors. Similar to EDF, the jobs are ordered by earliest absolute deadline, but the P highest priority processes are executed by the P processors in every time step. Scheduling events occur only when new jobs are introduced or when a job completes.

The likelihood of migrations in global EDF depends first on how often preemptions happen. Migrations can only occur due to unfavorable scheduling events e.g. on the occurrence of sporadic tasks etc. The introduction of a new job causes only active to idle transition i.e. if a higher priority job is introduced then the state of a lowest priority job changes from active to idle. Job completion causes only idle to active transitions. If a job has completed, the highest priority job is chosen to execute, regardless of what CPU the job may have been executing on previously. If a job's execution state has an active to idle to active sequence, it is possible for the job to migrate between CPUs. This means that global task migration is allowed thus the name global EDF.

The other factor in the likelihood of migrations in global EDF is the number of CPUs. If a preempted task is not migrated it definitely means that the task has been completed before other active jobs. Suppose completion time is uniformly random. If utilization is greater than P, it is obviously not schedulable. A simple bound is

$$\sum \frac{e_i}{d_i} \leq P - \max\{\frac{e_i}{d_i}\} \quad \text{(P-1)}$$

Take for example a set of tasks that barely fit the bound. Let P = 4 and Ti = (1, 2) for $i \in [1, 5]$.The condition above evaluates to $\frac{5}{2} \leq .\frac{5}{2}$ .If any task was longer, they might not fit, as can be seen in Figure 1.

B. PF

In this algorithm all tasks are independent of each other; there is no sharing of resources besides the processors. It does not take into the account the overhead of migrating tasks when scheduling tasks and assumes that the tasks are periodic and the deadlines of the tasks equal the periods.

The PF algorithm, presented in [2, 3], is optimal for scheduling periodic tasks on multiple processors thus improving on global EDF because global EDF is not optimal. This may be derived from the weighted round robin scheduling algorithm for uniprocessors where the tasks were given resources in the ratio of their priority.PF uses a "time-slicing" approach, i.e. it splits up tasks into unit intervals of execution, termed "slots", or ticks. PF uses this approach to approximate a "fluid scheduling", or constant rate scheduling. [6].

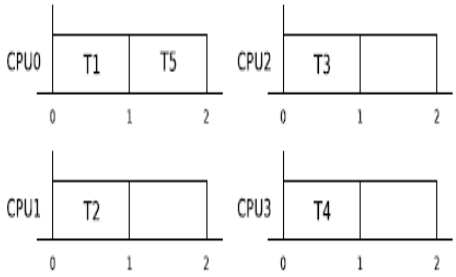

Fig 1 Global Schedule of five identical tasks with utilization1/2.

The basic idea of the PF algorithm is to assign slots to each task such that it is always scheduled proportionally to the utilization of the task. That is, if a task has a utilization of U, then if seen at any particular time slot t, the task will have been allotted U · t time slots for execution between 0 and t. The authors of [2, 3] define any schedule that satisfies this requirement as *proportionately fair* or *P-fair*. P-fair will meet all the deadlines for its tasks. The goal of the PF algorithm is to maintain the P-fair requirement for all tasks. If at all the value U · t is not an integer, then ceiling or floor function is used.

PF maintains a measure of how close the current schedule is to meeting the P-fair requirement. The authors of [2, 3] define a *lag* function at time t for a task T0 with utilization U0 as follows:

$$\text{lag}(T_0, U_0, t) = U_0 \cdot t - S(T_0)$$

where $S(T_0)$ is the number of slots allocated to T_0 thus far.

PF also makes use of a "characteristic string" that describes the resource requirements for each task at each slot. The characteristic string can be calculated offline, and it is used to make optimal decisions about the future demands of tasks on each iteration of the algorithm. The authors of [2, 3] define the characteristic string for a task T0 with utilization U_0 at time t over the characters (+, 0,−) as follows:

$$c(T_0, U_0, t) = \text{sign}(U_0(t+1) - \lfloor U0 \cdot t \rfloor - 1).$$

With the lag computation and the characteristic string in hand, PF classifies each task at a time t in the given way. A task is classified as *non-urgent* if the lag for the task is strictly less than 0 and the characteristic string at the slot for t is either − or 0. A task is classified as *urgent* if the lag for the task is strictly greater than 0 and the characteristic string at the slot for t is either + or 0. A task that is neither non-urgent nor urgent is classified as *contending*. An invariant that falls out of this classification is that at every slot t, all urgent tasks must be scheduled and all non-urgent tasks must not be scheduled. If either of these conditions cannot be met, then the task set cannot be feasibly scheduled using PF or any other algorithm, given that PF is optimal.

The order of contending tasks is defined lexicographically according to the characteristic string of each task. Specifically, the ordering is + 0 −. Additionally, when comparing the characteristic strings of two contending tasks at a particular slot t, only the subset of the characters from slot t to the first slot where the value of characteristic string becomes 0 must be compared. For example, let the characteristic string of a task be "− + − 0 − + + 0". When this task is classified as contending, only the sub string "− + − 0" is considered to order the task relative to the other contending tasks.

The mechanics of the PF algorithm are best explained with a simple example. The example, described in Figure 2 , uses PF to schedule three identical tasks on two processors;

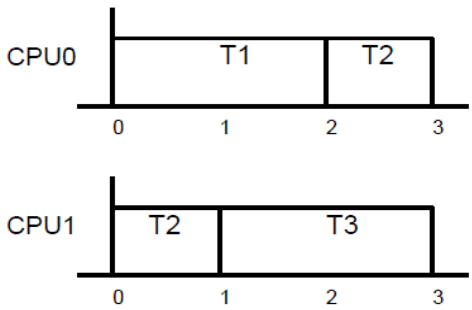

Fig 2 Schedule for PF example

Each task has a period of 3, a worst-case execution time of 2, and an initiation time of 0. Additionally, the characteristic string is the same for all three tasks, namely "− + 0 − +0...". It is important to note that this task set is infeasible with global EDF, but can be scheduled by PF.
If by chance the characteristic strings of a set of contending tasks are equal, the ties can be broken arbitrarily without affecting the fairness.

C. LLREF

The LLREF scheduling algorithm [5] assumes that tasks are preemptible and independent (i.e. that they have no shared resources). The cost of context switches and task migration is ignored and assumed to be negligible. The deadline is assumed to be equal to the period.

Like PF, LLREF approximates a fluid scheduling model. In the *fluid* scheduling model, each task executes at a constant rate at all times as opposed to a practical schedule. In order to reduce the number of scheduler invocations LLREF only schedules on a small set of events. It makes use of an abstraction called the Time and Local Execution Time Domain Plane (T-L Plane). The entire scheduling time is composed of T-L plane of different sizes such that feasible scheduling in a single T-L plane implies feasible scheduling over all times. This abstraction is used to determine when tasks must be scheduled in order to meet their deadlines. Figure 3 shows an example T-L Plane, where the x-axis is time and y-axis is the remaining execution time for a particular task.

For every two subsequent primary scheduling events, or task arrivals, right-angled triangular T-L Planes are formed where one edge is the first scheduling event, one is the diagonal of no remaining local laxity, and the third is the horizontal at the y-value where the fluid schedule intersects the next task arrival. The line of no local laxity has a slope of -1. All T-L Planes for each time interval are superimposed, and then the algorithm is performed on each of these T-L Planes. If there is a locally feasible schedule for each plane, then all tasks can be scheduled. When scheduling locally, only the current plane is considered. On primary scheduling events, a new T-L Plane is constructed, and all tasks are rescheduled according to it. Tokens represent the execution of tasks over time. If the task is selected, it will move to the right with a slope of -1 (e.g., t1 in Figure 3), and if it is not selected, it will move with a slope of 0 or horizontally (e.g., t0 in Figure 3).

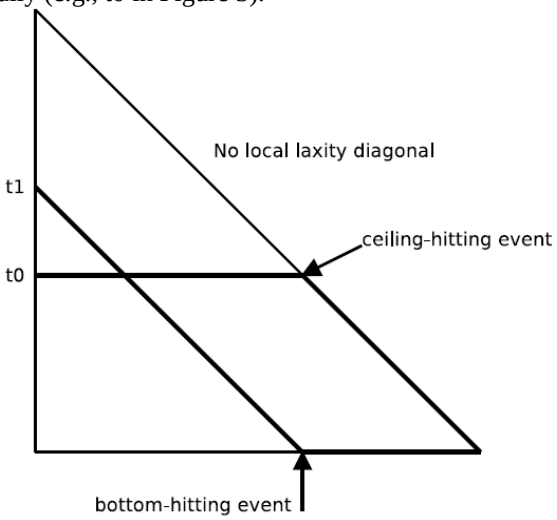

Fig 3 T-LPlane

Unlike PF, where scheduling occurs at each tick, in LLREF scheduling only happens either at primary or secondary scheduling events. There are two instances where secondary scheduling events can occur: when a task reaches the base of the triangle (a bottom-hitting event), and when a task hits the hypotenuse of the right angled triangle (a ceiling-hitting event). A bottom-hitting event indicates that the task has executed as much as it needs to for the local T-L Plane, so it should be deselected and another task should be run instead. A ceiling-hitting event indicates that the task has zero remaining local laxity, and therefore needs to be selected immediately to meet the deadline.

III. RESULTS

To evaluate and compare the algorithms, we implemented global EDF, PF, and LLREF using the RTSIM framework [8], an open-source library for simulating real-time scheduling algorithms. RTSIM builds on the simulation framework MetaSim [8], which presents a general interface to a prioritized event queue. Under RTSIM, a simulation is divided into ticks.

A. Evaluation Methodology

We empirically evaluated global EDF, LLREF and PF by randomly generating sets of tasks and extracting three parameters from the schedules generated by each of the algorithms. The first parameter evaluated the ability of the algorithm to feasibly schedule a given set of tasks. To measure schedulability, we generated task sets with a utilization less than the number of processors. This guaranteed that both LLREF and PF would be able to schedule the task sets. However, since global EDF is non-optimal, there is no guarantee that global EDF would be able to schedule these task sets[2]. Whenever a task set could not be scheduled by global EDF, this infeasibility was recorded.

The second parameter measures the total number of task migrations made by a specific schedule for a set of tasks. The third measures the total number of scheduler invocations required to produce the final schedule. The task migrations and scheduler invocations characterize the overhead of each algorithm. These parameters are ways to characterize the overhead of scheduling algorithms.

In our simulations, a task migration occurs when a task instance is descheduled from one processor and is later rescheduled on a different processor. For each algorithm, scheduler invocations occur at different times. For global EDF, the scheduler was invoked on every arrival of a task instance and on every completion of a task instance. At these points, global EDF makes decisions on the next task subset to schedule based on the absolute deadlines. For LLREF, there are both primary and secondary scheduling events. For PF, the scheduler is invoked every clock tick; this means the number of scheduler invocations for PF is equal to the number of clock ticks in the simulation of a specific task set.

B. Task Set Generation

The idea behind the algorithm used to generate the task sets is to iteratively add tasks with a random period and worst-case execution time until the utilization exceeds the total number of processors. The task that pushed the utilization over the bound is then removed to get the final task set.

We generated 1100 task sets for each combination of these parameters, resulting in 13200 task sets. We input each of these task sets into RTSIM and invoked each of the three algorithms to create schedules for these tasks.

IV. DISCUSSION

A. Schedulability

Global EDF is far simpler than PF and LLREF, but it pays for it in schedulability. Where PF and LLREF are optimal, global EDF can only make loose guarantees. In all cases, increasing the CPU count increases the utilization level where missed deadlines may occur. However, normalizing these values shows that the system utilization does not necessarily increase with the CPUs.

B. Scheduler Invocations

Scheduler invocations are a source of overhead, and in a real time system, time spent scheduling reduces the time available to run the tasks. Even in cases where the scheduler can be run offline, a higher number of scheduler invocations will lengthen the time required to find a feasible schedule. For this reason, it is desirable to minimize the number of times the scheduler is invoked. In PF, the scheduler is invoked at every tick. Since LLREF is designed to improve on PF by minimizing the number of scheduler invocations, it only schedules on primary or secondary events. Therefore, if primary events occur on a relatively high percentage of ticks, the number of invocations will be similar to PF. This will occur if tasks have short periods, as LLREF must schedule on all primary events.

As the number of processors increases, the distributions of the difference between LLREF and PF remain fairly consistent. The exception to this is that the spike at 0 increases in size with number of processors. The reason for this is that when the tests are run for more processors, the task sets contain more tasks. The probability that one of these tasks will have a short period, and thus lead to a large number of scheduler invocations, is then higher. This may not be the case in real systems. In addition, as the number of tasks increases, the likelihood that any particular tick will be a multiple of one of the task periods increases, leading to more invocations in general. Since PF already is invoked in every time step, this only affects LLREF.

Global EDF has the fewest scheduler invocations; they only occur when a task arrives or completes. This is a subset of the invocations in LLREF, since LLREF considers task arrivals as primary events, and local task completion is only one of the possible secondary scheduling events. However, the trade-off there is that LLREF can schedule sets of tasks which are not feasible using global EDF. Our results show that, as expected, EDF always has fewer invocations than LLREF. As with the comparison of LLREF and PF, the number of cases where the schedulers have equal numbers of invocations increases with the number of processors. This has the same cause; if there is a task deadline at every tick, then both EDF and LLREF will schedule at every tick. Thus, for some sets of tasks, all three algorithms will be identical when considering number of scheduler invocations. The comparison done by RTSIM is shown in Figure 4.

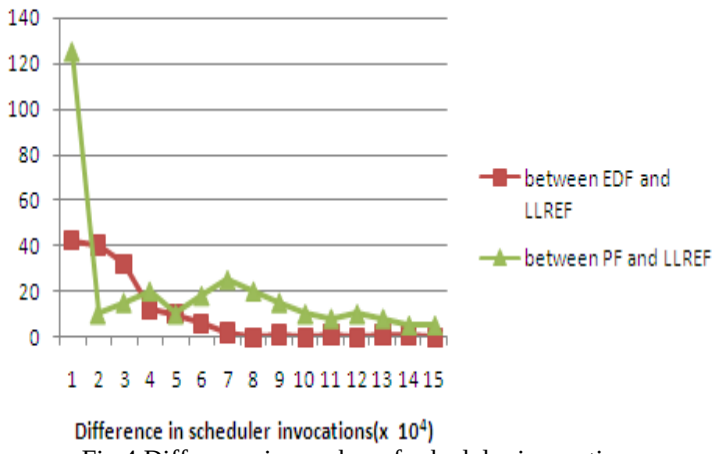

Fig 4 Difference in number of scheduler invocations

C. Task Migrations

A goal of any multiprocessor scheduling algorithm should be to minimize the number of task migrations required to schedule a task set. A task migration requires making all the resources available to the new processor. Depending on the architecture of the system, this could be quite costly.

For all three algorithms, task migrations can only occur when the scheduler is invoked. However, a task migration does not necessarily occur on every scheduler invocation. Task migrations could theoretically occur at every tick in PF. Since LLREF requires fewer scheduler invocations, it makes sense that LLREF would almost always require fewer task migrations as well. However, our simulation results show that the trends for scheduler invocations do not translate directly to the trends for task migrations when comparing LLREF and PF.

Specifically, we observe that there is a larger variation in the number of task migrations. To reduce task migrations LLREF keeps tasks that have hit the execution ceiling on the same processor while PF implementation does not make any effort to ensure that a task scheduled in two consecutive ticks remains on the same processor. The differences between these algorithms in terms of task migration performance depend partially on the implementation. As a result global EDF requires the fewest task migrations a task migrates only when preemption occurs. LLREF must sometimes migrate tasks in order to execute all of them; in some cases, a task set that has many migrations is simply not schedulable with EDF. The comparison done by RTSIM is shown in Figure 5.

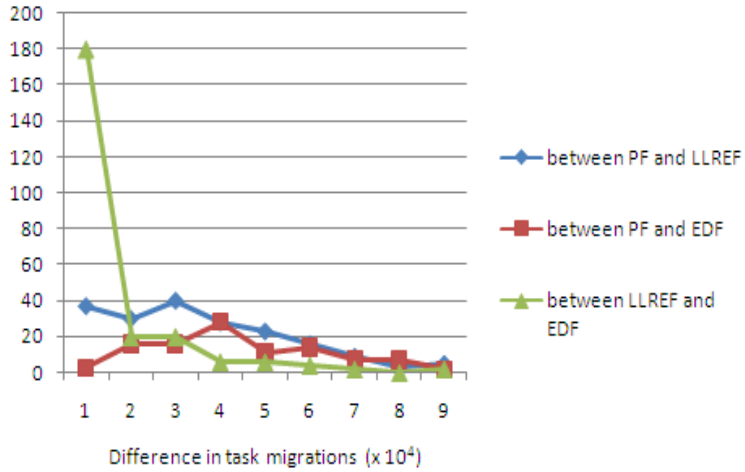

Fig 5 Differences in number of task migrations

V.CONCLUSION

We examined three scheduling algorithms for uniform multiprocessor systems: global EDF, PF, and LLREF. Although simple and straightforward, global EDF is a non-optimal multiprocessor scheduling algorithm but it demonstrated its merits with its low overhead in terms of task migrations and scheduler invocations. PF improved on global EDF by feasibly scheduling a large number of task sets that global EDF could not schedule but performed far worse than global EDF in terms of overhead. LLREF was midway between global EDF and PF. In every aspect we evaluated, LLREF performed better than PF. In terms of overhead, LLREF performed worse than global EDF, but this performance gap is considerably reduced when compared to PF. In the end, choosing whether to use one of these algorithms will require deciding whether optimality or low scheduler overhead is more important.

REFERENCES

[1] Baker, T. P. A comparison of global and partitioned edf schedulability tests for multiprocessors. Tech. rep., In International Conf. on Real-Time and Network Systems, 2005.

[2] Baruah, S. K., Cohen, N. K., Plaxton, C. G., and Varvel, D. A. Proportionate progress: a notion of fairness in resource allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 1993), STOC '93, ACM,pp. 345–354.

[3] Baruah, S. K., Gehrke, J., and Plaxton, C. G. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Symposium on Parallel Processing* (Washington, DC, USA, 1995), IPPS '95, IEEE Computer Society, pp. 280–288.

[4] Brandenburg, B. B., Calandrino, J. M., and Anderson, J. H. On the scalability of real- time scheduling algorithms on multicore platforms: A case study. In *IEEE Real-Time SystemsSymposium* (2008), pp. 157–169.

[5] Cho, H., Ravindran, B., and Jensen, E. D. An optimal real-time scheduling algorithm for multiprocessors. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 101–110.

[6] Holman, P., and Anderson, J. Adapting Pfair scheduling for symmetric multiprocessors.*Journal of Embedded Computing 1*, 4 (2005), 543–564.

[7] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20* (January 1973), 46–61.

[8] Retis Lab, Scuola Superiore Sant'Anna. *The RTSIM project*. http://rtsim.sssup.it.