Evaluation of priority based real time scheduling algorithms: Choices and tradeoffs

CONFERENCE PAPER · JANUARY 2008

DOI: 10.1145/1363686.1363764 · Source: DBLP

3 AUTHORS, INCLUDING:

Biju K Raveendran
BITS Pilani, K K Birla Goa
12 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Sundar Balasubramaniam
Birla Institute of Technology and Science Pilani
13 PUBLICATIONS **122** CITATIONS
SEE PROFILE

Evaluation of Priority Based Real Time Scheduling Algorithms: Choices and Tradeoffs

Biju K Raveendran
BITS Pilani, Rajasthan, INDIA
91 – 94 – 60080115
biju@bits-pilani.ac.in

Sundar Balasubramaniam
BITS Pilani, Rajasthan, INDIA
91 – 99 – 28007679
sundarb@bits-pilani.ac.in

S Gurunarayanan
BITS Pilani, Rajasthan, INDIA
91 – 94 – 14082472
sguru@bits-pilani.ac.in

ABSTRACT
Priority based real time scheduling algorithms such as RM and EDF have been analyzed extensively in the literature. Many recent works on scheduling address energy consumption as a performance metric. In this work we analyze traditional priority scheduling algorithms RM, EDF, and LLF along with a few power-aware scheduling algorithms such as MLLF, RM_RCS and EDF_RCS. Our analysis addresses the following metrics: response time, response time jitter, latency, time complexity, preemptions, and energy consumption. –We extend past work in this direction by characterizing the behavior of the scheduling algorithms in detail using theoretical analysis as well as experimental evaluation. The results of our analysis can be used to control design choices for real time systems.

Categories and Subject Descriptors
D.4.7 [**Operating Systems**]: Performance – Measurements, Operational Analysis, and Simulation.

General Terms
Measurement, Performance, Experimentation.

Keywords
Priority Scheduling, Real Time Scheduling, Performance Evaluation, and Characterizing Scheduling algorithm.

1. INTRODUCTION
Since Liu and Layland's seminal work [1] on scheduling algorithms for periodic tasks, several real-time scheduling algorithms have been proposed and analyzed in the literature. In particular, Rate Monotonic (RM) and Earliest Deadline First (EDF) are the most commonly implemented algorithms in practice as well as the most analyzed algorithms in literature. More recently, the proliferation of mobile embedded computing devices running on limited battery power has brought to focus the issue of power-aware computing – in particular power-aware scheduling. Among the factors affecting power consumption, context switching is one that is often clearly identified as 'overhead'. A few real-time scheduling algorithms have been proposed with the aim of reducing the number of preemptions (or context switches) in a schedule [3, 4]. But there are other constraints such as Turn-Around Time (or Response Time[1]), Latency, and Response Time Jitter, which may be of equal if not of more importance than preemptions (or energy consumption) for specific application scenarios [5]. But most of the comparative evaluations so far have either focused only on RM and EDF or only on limited analysis of preemption and energy consumption. In this paper we attempt a comprehensive experimental evaluation of six different scheduling algorithms by comparing them on several metrics using simulated task sets.

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

1.1 Related Work
Butazzo [5] provides a rigorous comparative evaluation of RM vs EDF based on different metrics: Implementation Complexity, Runtime Overhead (measured in number of preemptions), Response Time Jitter, and Latency among other things. We extend this evaluation to other conventional priority scheduling algorithms such as Least Laxity First [6] as well as to scheduling algorithms that focus on preemption reduction [3, 4]. In this paper we focus on all the other performance metrics under schedulable conditions. Also, we are not considering *overload* conditions for our analysis.

Various techniques have been proposed in the literature for reducing the number of preemptions in a schedule. In [3], Oh and Yang propose a variant of the Least Laxity First Algorithm to reduce preemptions in a schedule, known as Modified LLF (MLLF). MLLF attempts to fix the "frequent preemption problem of LLF" [7] without affecting its optimality. The approach is simple and effective in addressing the limitation of LLF but it does not attempt to "minimize" the number of preemptions in any sense. Furthermore no detailed analysis on the effectiveness of the algorithm (in reducing preemptions) is available.

In [8], Wang and Saksena describe a fixed priority scheduling algorithm that reduces context switches by assigning an active priority known as preemption threshold for each job. This approach is limited to fixed priority scheduling as threshold assignment may take exponential time. Lee et. al [9, 10] address the impact of context switches on caching behavior by proposing a new algorithm known as Limited Preemptive Scheduling (LPS). LPS uses static data flow analyses techniques to determine the preemption points with small cache loading costs. The primary limitation of this approach is that it requires extensive data flow

[1] In Real-Time Systems, the term "Response Time" of a job is often used to refer to its "Turn-Around Time" [2]. Henceforth we will use the term "Response Time" in this sense.

analyses and therefore may not be suitable for dynamic scheduling.

Raveendran et. al. [4] describes a dynamic priority scheduling scheme that focuses on aggressive reduction of the number of preemptions without requiring extensive computations. The heuristic used in [4] is not so straightforward as the MLLF heuristic but it is much more efficient than the exhaustive search used in [8] or the data flow technique used in [9, 10]. In this paper, we compare the performance of conventional priority scheduling algorithms RM, EDF and LLF with that of "preemption reducing" dynamic scheduling algorithms described in [3, 4].

In [11], Vahid et. al. propose a modification to the Maximum Urgency First (MUF) scheduling algorithm [12] known as MMUF. MUF addresses the issue of schedulability of critical tasks in the presence of transient overloads. MMUF is a hybrid priority algorithm that is a generalization of RM, EDF, LLF, MLLF, and MUF. Like MLLF, MMUF lets the currently active process to continue running in the presence of other processes with the same priority. Thus with respect to preemption reduction MMUF behaves the same as MLLF.

Recent work on energy-aware scheduling has focused on architecture-level techniques such as Dynamic Voltage Scaling and Dynamic Frequency Scaling [13, 14]. In contrast our effort is focused on Operating System level techniques for scheduling that are not dependent on specific architectural features. There have been several recent attempts to characterize energy consumption at the operating system level. In particular, power analysis of real time operating systems in [15] identifies time and energy profiles of different operating system functions and the behavioral characterization in [16] includes energy consumption profiles. But these and other attempts do not consider the effect of task scheduling in energy consumption nor do they relate energy savings efforts to other real-time systems performance characteristics. In [17] Gopalakrishnan and Parulkar characterize the impact of preemptive scheduling on utilization. We adopt a similar approach in that we also count the number of preemptions but with the difference that we rely on experimental evaluation for comparing the different scheduling algorithms.

2. PARAMETERS FOR COMPARISON

One of our motivations is to extend the work in [5] to scheduling algorithms other than RM and EDF – in particular to those algorithms aimed at reducing preemptions. In this section, we briefly review the parameters used for our evaluation and their significance. We use the following notations in this section:

T1, T2 …. Ti … Tn represents the job corresponding to task T.

finish-time(T_i) represents the finishing time of i^{th} job of task T.

arrival-time(T_i) represents the arrival time of i^{th} job of task T.

responseTime(T_i) =finish-time(T_i) – arrival-time(T_i)

decisions(A,H, S) defines the number of decision points to be made by the scheduling algorithm A over hyper-period H and task set S.

$T_{dec}(A,H, S)$ define the time complexity of a single decision.

2.1 Response Time

For a periodic task T, the Maximum Response Time over all jobs:

Maximum Response time of $T = max_i\ (responseTime(T_i))$

We normalize the Maximum Response Time values with respect to execution times of the corresponding tasks.

2.2 Response Time Jitter

Two different jitters are usually defined - Absolute Response-Time Jitter (ARJ) and Relative Response-Time Jitter (RRJ):

$ARJ(T) = max_i(responseTime(T_i)) – min_i(responseTime(T_i))$

$RRJ(T) = max_i(responseTime(T_{i+1}) – responseTime(T_i))$

We normalize the Jitter values with respect to periods of the corresponding tasks.

2.3 Latency

For a task T, we define the maximum input-output latency [5, 19] as $L(T) = max_i(finish\text{-}time(T_i) – start\text{-}time(T_i))$

We normalize the Latency values with respect to execution times of the corresponding tasks.

2.4 Scheduling Complexity

Given a set of tasks S with hyper-period H the time complexity for algorithm A is $T_A(H,S) = decisions(A,H, S) * T_{dec}(A,H, S)$

The time taken for insertion or deletion is a function of the length of the priority queue [20]. The length of the queue is bounded by the number of tasks for any periodic task set.

2.5 Preemption Count

In this work, we ignore context switches initiated due to tasks blocking on their own – say, for resource requirements and consider only the preemptions introduced by scheduling. The impact of context switches on utilization and energy consumption is not straightforward – particularly due to data movement within the memory hierarchy. All preemptions may not cause data movement but whenever this happens this part of the overhead may be much more significant than that caused by a switching of tasks. This implies that the time spent in switching contexts may be variable and in turn may increase response time and response time jitter for tasks. The measurement of the overhead due to data movement within the memory hierarchy is beyond the scope of this paper. Thus we restrict ourselves to counting the preemptions in the schedule output by an algorithm. This is similar to the approach in [17] but we focus on experimental evaluation.

Liu [2] argues that 2*N, where N is the number of jobs, is the upper bound for number of context switches in a schedule decided by job level priorities. Of course, this does not apply for fully online algorithms like LLF. On the other hand the theoretical upper bound may not be a close estimate of the actual number of preemptions for the other algorithms. We normalize preemption counts with respect to the number of jobs i.e. we consider preemptions as a function of the total number of jobs in a schedule. This makes particular sense when we compare the preemption overhead to the overhead due to scheduling decisions as the number of scheduling decisions usually depends on the number of jobs among other factors.

2.6 Energy Consumption

Scheduling algorithms impact energy consumption in two ways: by the time taken to schedule tasks and by the number of preemptions they induce [5, 6]. Four different components of energy consumption have been attributed to scheduling [21]: Timer Interrupt Energy, Scheduling Energy, Context Switch Energy, and Signal Handling Energy.

Signal Handling Energy and Timer Interrupt Energy are specific to the platform (i.e., the architecture and the operating system) but fairly independent of scheduling algorithms. It is important to observe that Scheduling Energy and Context Switch Energy are identified separately because each scheduler invocation may not

result in a context switch (i.e., a preemption). Isolated measurements of energy consumption are harder because data movements are dependent on many factors including nature of application, input profile, allocation policies etc. So we use a simpler, if inaccurate, measure in our analysis: the number of preemptions in a schedule. Fewer preemptions always result in less energy consumption but may or may not result in significant energy reduction as data movement may still not be reduced.

Another factor in context switch overhead depends upon the task model on a platform: whether tasks are lightweight (a la threads) or heavy weight (a la processes). Thread switching usually causes less overhead than process switching. For instance, [22] reports an average of 860 nJ as the energy consumed per thread switch in an eCOS system running on a StrongArm 1100 processor at its peak frequency. This is an order of magnitude smaller than the 12500 nJ reported in [21]. Again using preemption count as the metric abstracts away from these details and allows us to compare the scheduling algorithms.

Our comparative analysis of scheduling algorithms uses two metrics for energy consumption: the time spent by the algorithm on scheduling decisions and the number of preemptions in the resultant schedule.

3. PRIORITY SCHEDULING ALGORITHMS

In this section we present three of the traditional priority scheduling algorithms – Rate Monotonic (RM), Earliest Deadline First (EDF), and Least Laxity First (LLF) – as well as three other algorithms aimed at reducing preemptions – Modified LLF (MLLF [3]), RM with Reduced Context Switches (RM_RCS [4]), and EDF with Reduced Context Switches (EDF_RCS [4]) using an example so that the ensuing discussion and analyses are clear. All these algorithms are priority based scheduling algorithms for periodic tasks with hard deadlines. Table 1 show the example task set. We took the following assumptions in our presentation of the algorithms:

- Arrival times for first instances of all tasks are assumed to be 0
- period(J) for a job J is the same as period(T) where J is an instance of task T.
- For each job J, deadline(J) = arrival_time(J) + period(J).

Table 1. Task details

Task	Arri. Time (for the first instance)	Period	Exec. Time
T1	0	4	1
T2	0	5	2
T3	0	20	7

Figures 1 to 6 show the schedules obtained by EDF, RM, LLF, MLLF, EDFRCS and RMRCS algorithms for the task set in Table 1. EDF [2] is a job level fixed priority - scheduling algorithm with deadline as the priority. The EDF schedule for the given task set includes three context switches as shown in Fig 1. RM [1, 2] is a task level fixed priority - scheduling algorithm with period as the priority. The RM schedule for our example task set includes five context switches as shown in Fig 2. [5] Observes that in general an RM schedule results in more context switches than an EDF schedule for the same task set.

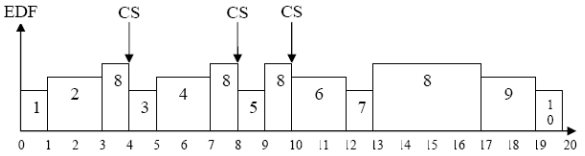

Fig 1. EDF schedule for task set in Table 1.

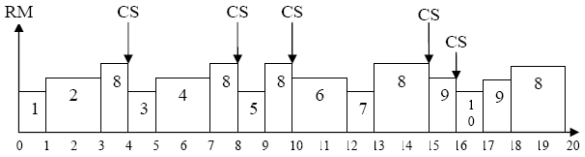

Fig 2. RM schedule for task set in Table 1

LLF [2] is a job level dynamic priority algorithm where laxity (i.e. slack time) is used as priority. Laxity is calculated by *deadline(J) – rem_time(J,t)* – t where t is the current time, and rem_time(J,t) is the remaining execution time of job J at t. [3] observes that an LLF schedule results in a significantly larger number of context switches as compared to an EDF schedule due to the frequent changes in priority. Fig 3 shows that the LLF schedule for our example task set includes five context switches.

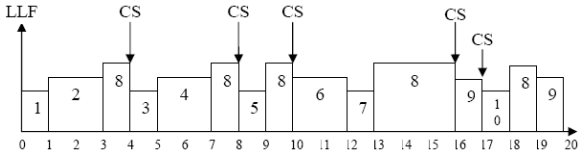

Fig 3. LLF schedule for task set in Table 1.

MLLF [3] is a modification of LLF, which focuses on reducing the number of context switches caused by LLF. Whenever more than one job has the same slack time (i.e., same priority) the one with minimum deadline will continue its execution till its completion or till scheduler encounters another job with lower deadline and lower slack. This change results in heavy reduction of preemptions in an LLF schedule. This change does not affect schedulability of LLF. Fig 4 shows the MLLF schedule for our example task set with three context switches.

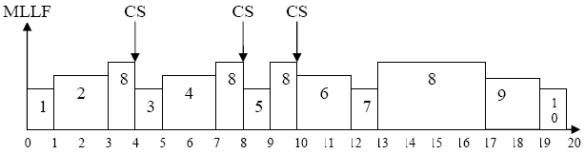

Fig 4. MLLF schedule for task set in Table 1.

EDF_RCS [4] and RM_RCS [4] are variants of EDF and RM respectively that extend the execution of the current job to the maximum possible extent while preserving schedulability of all future jobs. This decision requires computing the maximum possible extension time. Also when the current job is extended, a deferred-switching event has to be marked. This can be achieved by setting a timer. When a job is in execution and a high priority job arrives in the system, scheduler will use the extension heuristic to find whether the currently running job can continue its execution without affecting the schedulability criteria. EDF_RCS and RM_RCS preserve schedulability conditions of EDF and RM respectively. Resultant EDF_RCS and RM_RCS schedules for our example task set have only one context switch each as shown in fig 5 and 6 respectively.

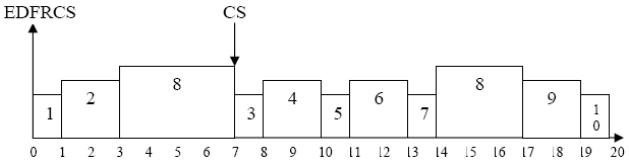

Fig 5. EDF_RCS schedule for task set in Table 1.

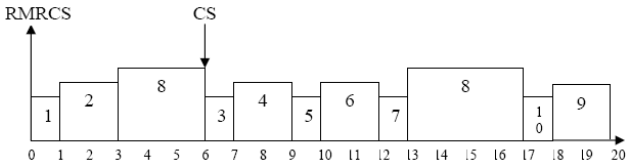

Fig 6. RM_RCS schedule for task set in Table 1.

4. COMPARATIVE EVALUATIONS

In this section, we evaluate the six algorithms presented in Section 3 with respect to the metrics described in Section 2. Our experimental setup included simulations of all the six algorithms and different test suites randomly generated under certain conditions: each test suite was characterized by a fixed number of tasks (between 2 and 20) and a fixed load i.e. utilization (from low (50%) to high (100%)). Each test suite included 100 different task sets of varying hyper-periods – from 100 to 32000. The results obtained were then averaged over these 100 test suites as appropriate.

We do not include schedulability as a metric in our evaluation because schedulability analyses have been dealt with extensively in the literature. In the cases of RM and RM_RCS, we consider only those task sets that are RM-schedulable for the following analyses. This does introduce a bias - albeit a predictable one - in favor of these two algorithms. Wherever this is an issue we explicitly address this fact in the analysis.

4.1 Response Time

We measured the response times for different tasks in a schedule for a fixed utilization of 90%. The results are shown in this chart (Fig. 7)

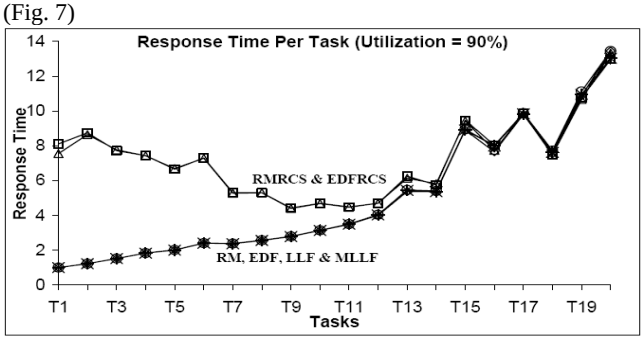

Fig 7.Response time Per Task

We observe that the traditional algorithms (RM, EDF, LLF) do not show any real difference – among themselves – in response time behavior and the response time increases with decreasing frequency of the tasks. Among the preemption reduction algorithms MLLF results in schedules with same response times as those produced by LLF. But RM_RCS and EDF_RCS result in schedules where the response times of higher frequency tasks are much larger than in schedules produced by RM and EDF respectively. This increase in response times can be attributed to the fact that a lower frequency task may continue execution in preference to a higher priority task due to the preemption reduction heuristic used by RM_RCS and EDF_RCS. This difference in the response time behavior appears to be independent of utilization. The following chart (Fig. 8) shows the average response time against utilization:

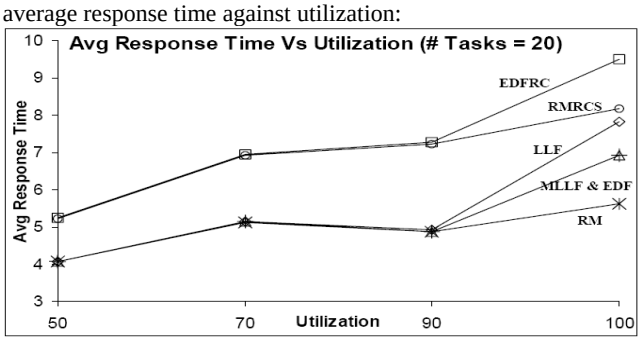

Fig 8. Average Response Time Vs Utilization

We can observe that the average response time of tasks increases with increased utilization for all the algorithms and the RCS algorithms show consistently higher average response times. The slightly anomalous trend observable between 90% and 100% load is due to the fact that the trend lines for RM and RM_RCS include only task sets that are RM-schedulable.

4.2 Response Time Jitter

We measured Absolute Response Time Jitter (ARJ) and Relative Response Time Jitter (RRJ) for different tasks for fixed utilization values. The chart (Fig. 9) below shows the ARJ values per task at a fixed utilization (of 70%).

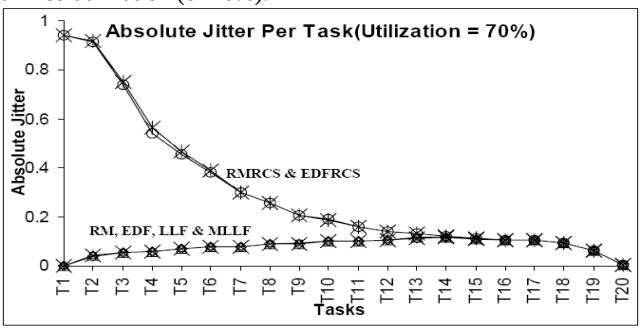

Fig. 9. Absolute Jitter Per Task

It can be observed that the normalized absolute jitter remains low for the lower frequency jobs independent of the scheduling algorithm whereas for the higher frequency jobs, the RCS algorithms exhibit very high jitter. This behavior can be explained by the fact an instance of a higher frequency task may be scheduled early due to its high priority whereas another task may be scheduled late to avoid preemption of some other lower frequency job thereby resulting in the variation (i.e. jitter) in response times.

However at a higher utilization some of the algorithms show marked deviation from the above behavior as shown in Fig 10.

We observe that in this case of a fully loaded system, all the other algorithms exhibit higher jitter compared to RM and RM_RCS for lower frequency tasks and LLF shows a pronounced increase in jitter for these tasks compared to all the other algorithms. It must be observed that our results appear to be different from those in [5] because we consider only RM-schedulable task sets for RM

and RM_RCS. If we include all the task sets in our analysis then at high loads RM and RM_RCS would exhibit higher response times for low frequency tasks as compared to EDF and EDF_RCS respectively. This behavior is in agreement with the results reported in [5] but such performance analysis may not be relevant for tasks that are missing the deadline anyway. Also in the chart above (Fig. 10) the lowest frequency task is an anomalous case, because often there is exactly one instance of such a task within a hyper-period and hence jitter (i.e. variation in response times) is not well defined.

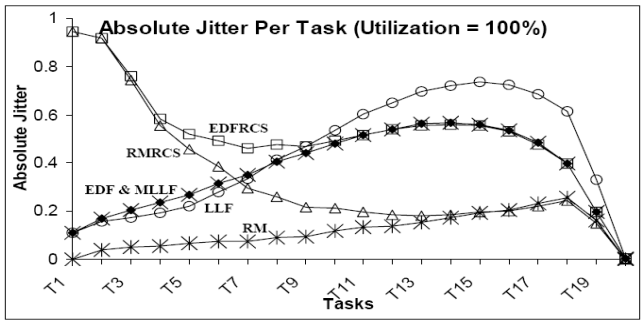

Fig. 10. Absolute Jitter Per Task

4.3 Latency

Figure 11 shows the average latencies against utilization levels.

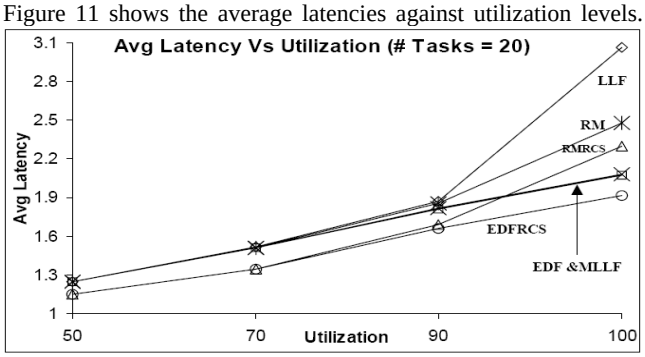

Fig. 11. Average Latency Vs Utilization

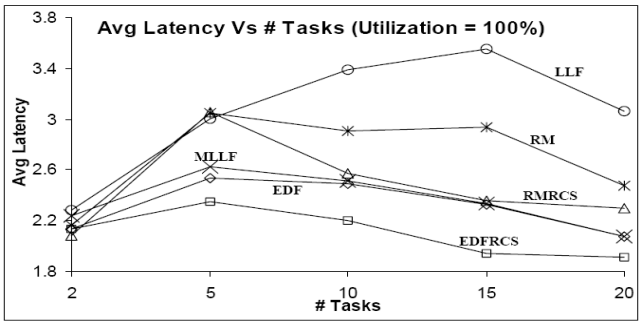

Fig. 12. Average Latency Vs Number of Tasks

We can observe that the average latency increases with load for all algorithms and that the preemption reduction algorithms perform slightly better than their counterparts in all cases. The observations can be explained by the fact that reduced preemptions increase the chances of continued execution of any job thereby reducing its latency. This difference between the traditional and the preemption-reducing algorithms in average latencies is more pronounced at higher loads because the traditional algorithms induce more preemption at higher loads. The chart in Figure 12 shows the average latency against number of tasks at full load.

4.4 Scheduling Complexity

The worst-case complexity measures (decision points, per decision complexity and scheduling complexity) of all the six algorithms are summarized in Table 2.

Table 2. (Worst Case) Time Complexity of Scheduling Algorithms

	Decision Points	Per Decision Complexity	Scheduling Complexity
RM	2*N	O(1)	O(2*N)
EDF	2*N	O(log m)	O(2*N*log|S|)
LLF	H	O(m)	O(H*log|S|)
MLLF	H	O(m)	O(H*log|S|)
RM_RCS	(2+d)*N where d grows very slowly w.r.t |S|	O(p*p)	O((2+d)*N*|S|*|S|)
EDF_RCS	(2+d)*N where d grows very slowly w.r.t |S|	O(p*p)	O((2+d)*N*|S|*|S|)

where H is Hyper-period, N is number of jobs, |S| is number of tasks, m is queue length and p is number of higher priority jobs.

The average case complexities are harder to estimate theoretically. We have experimentally measured the queue length over a large number of test cases. The average queue length for EDF is approximately 1.25*log|S|. Thus the average case time complexity of EDF is (2*N*log(1.25*log|S|)).

For LLF and MLLF, although the number of decision points is H, the number of points at which the priorities have to be updated is significantly less. We have estimated this value experimentally to be approximately 2.37*N and 2.35*N for LLF and MLLF respectively. Updating priorities takes time proportional to m, the length of the queue, and the average value for m as mentioned above is 1.25*log|S|. Thus the average case time complexities for LLF and MLLF would be (H+2.96*N*log|S|) and (H+2.94*N*log|S|) respectively.

For the RCS algorithms we experimentally estimated the average number of decision points to be 2.06*N. We also experimentally estimated the average value of p, the number of higher priority jobs, to be approximately 0.6log|S|. Thus the average case time complexity of both RM_RCS and EDF_RCS is (0.74*N*log|S|*log|S|). These average case complexity measures are summarized in Table 3.

Table 3. Estimated (Average Case) Time Complexity of scheduling as a function of N, H, and |S|

	Decision Points	Per Decision Complexity (Proportional)	Scheduling Complexity (Proportional)
RM	2*N	Constant(c)	2*N
EDF	2*N	log(1.25*log|S|)	2*N* log(1.25*log|S|)
LLF	H decisions and 2.37*N updates	1.25*log|S|	H+2.96*N*log|S|
MLLF	H decisions and 2.35*N updates	1.25*log|S|	H+2.94*N*log|S|
RM_RCS	2.06*N	0.36*log|S|*log|S|	0.74*N *log|S|*log|S|
EDF_RCS	2.06*N	0.36*log|S|*log|S|	0.74*N *log|S|*log|S|

4.5 Preemption Count

The chart below (Fig. 13) shows preemption counts against utilization for task sets with a fixed number of tasks.

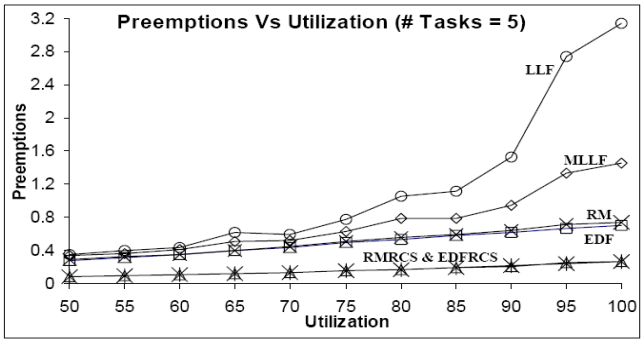

Fig. 13. Preemptions Vs Utilization

We can observe that the RCS algorithms show significant reduction in preemptions compared to their traditional counterparts, irrespective of the load. MLLF on the other hand reduces preemptions significantly with respect to LLF at high loads. But it is important to note that MLLF still results in more preemptions than EDF or RM. In [5] it is shown that EDF results in fewer preemptions than RM. Our results confirm this. Similarly EDF_RCS also results in fewer preemptions than RM_RCS.

As the number of tasks in the task set increases, preemption reduction is more pronounced for the RCS algorithms. On the other hand MLLF shows significant reductions in preemption count when the number of tasks is smaller, but this is partly explained by the fact that the preemption count for LLF is higher when the task set is smaller. The chart below (Fig. 14) shows the preemption counts per job against utilization for task sets with number of tasks at 20.

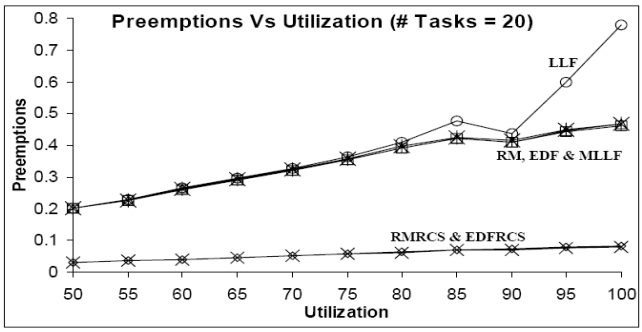

Fig. 14. Preemptions Vs Utilization

Table 4. Estimated Average Preemption Count

Algorithm	Preemption Count
RM	(0.42+2.7/|S|)*N*U
EDF	(0.45+2.1/|S|)*N*U
MLLF	(10.9/|S|)*N*U
RM_RCS	(1.9/|S|)*N*U
EDF_RCS	(1.9/|S|)*N*U

where N is number of jobs, |S| is number of tasks and U is utilization (between 0 and 1).

We observe that the average number of preemptions per job grows monotonically with increasing utilization for a fixed number of tasks for each algorithm under consideration. Except in the case of LLF, this growth is often linear. Furthermore, the average preemption count decreases with increasing number of tasks in the task set. More precisely, we were able to estimate that the normalized average preemption count is a linear function of 1/|S| where S is the task set. Based on these observations we summarize the estimated average pre-emption counts for all the algorithms except LLF in Table 4.

4.6 Energy consumption

For the purpose of a head-to-head comparison of the energy consumption impact among the algorithms, we compare the time spent on scheduling decisions and the number of preemptions both amortized over N, the number of jobs. Table 5 below summarizes these energy consumption figures for different combinations of |S| and U values.

The Table 5 can be used as a ready reckoner for energy consumption tradeoffs: for instance, for a E_c/E_s ratio of 50, one can observe that preemption energy is likely to dominate and hence the RCS algorithms are likely to offer energy savings of the order of 50% or more compared to RM or EDF; whereas if the E_c/E_s ratio is around 10, then one can observe that the RCS algorithms may still offer energy savings for small task sets whereas for large task sets they spend more energy on scheduling decisions than they save on preemptions. Given the earlier reports on energy measurements [21, 22], context-switching energy E_c is likely to be an order of magnitude larger than scheduling energy and as each scheduling decision is likely to require several instructions, we can expect an E_c/E_s value close to 100. However we admit this depends on several factors including architectural constraints, operating system implementation issues, whether thread switching or process switching is involved, and how much data movement is involved in a context switch. Thus this table is useful in practice only in conjunction with specific experimental measurements of unit energy consumption values on the target platform. But the table does give a good indication of when preemption reduction is worth considering and when it is not.

Also, it can be observed that the MLLF algorithm performs very badly with respect to energy consumption as it always spends more time on scheduling decisions than the RCS algorithms and the number of preemptions produced by MLLF is no better than that produced by EDF.

Table 5. Estimated Energy Consumption due to Scheduling Decisions and Preemptions (normalized per job)

|S|	U	RM	EDF	MLLF	*RCS
5	0.5	2Es+0.48Ec	3.04Es+0.44Ec	H'+6.83Es+1.09Ec	3.99Es+0.19Ec
5	0.7	2Es+0.67Ec	3.04Es+0.61Ec	H'+6.83Es+1.53Ec	3.99Es+0.27Ec
5	0.9	2Es+0.86Ec	3.04Es+0.78Ec	H'+6.83Es+1.96Ec	3.99Es+0.34Ec
10	0.5	2Es+0.35Ec	4.33Es+0.33Ec	H'+9.77Es+0.55Ec	8.13Es+0.10Ec
10	0.7	2Es+0.48Ec	4.33Es+0.46Ec	H'+9.77Es+0.76Ec	8.13Es+0.13Ec
10	0.9	2Es+0.62Ec	4.33Es+0.59Ec	H'+9.77Es+0.98Ec	8.13Es+0.17Ec
20	0.5	2Es+0.28Ec	5.28Es+0.28Ec	H'+12.71Es+0.27Ec	13.82Es+0.05Ec
20	0.7	2Es+0.39Ec	5.28Es+0.39Ec	H'+12.71Es+0.38Ec	13.82Es+0.07Ec
20	0.9	2Es+0.50Ec	5.28Es+0.50Ec	H'+12.71Es+0.49Ec	13.82Es+0.09Ec

where *RCS denotes RM_RCS or EDF_RCS, H' is a constant multiple of hyper-period H, E_s is energy consumption per unit time spent in scheduling decisions and E_c is energy consumption per preemption

5. CONCLUSIONS

We have experimentally evaluated six different real time scheduling algorithms with respect to various performance metrics and analyzed the results. A comprehensive summary of this evaluation is presented (see Table 6) in the form of desired performance characteristics and the corresponding choice of scheduling algorithms.

Table 6. Ready Reckoner for choice of Scheduling Algorithm

Desired Performance Characteristic	Choice of Scheduling Algorithm
Low Response Time	RM under low load and EDF under high load
Low Response Time Jitter	RM if jitter is critical only for high frequency tasks; RM under low load; EDF under high load.
Low Latency	EDF_RCS
Ease of Implementation	RM under low load and EDF under high load
Low Energy Consumption	EDFRCS for small task sets. EDFRCS under large E_c/E_s (50 or greater) irrespective of task set size RM under low load and small E_c/E_s (around 10) EDF under high load and small E_c/E_s (around 10)

Our future work will focus on evolving a systematic process for selection of scheduling algorithms based on quantitative performance requirements.

6. ACKNOWLEDGEMENTS

Our thanks to Microsoft Research India for supporting this work through PhD research fellowship and Embedded Systems Research Grant.

7. REFERENCES

[1] Liu, C, L., and Layland, J, W. "Scheduling Algorithms for Multiprogramming in Hard Real- Time Environment", Journal of ACM (JACM), vol. 20, no. 1, pages 46 – 61, Jan 1973.

[2] Liu, J, W, S. "Real Time Systems", ISBN 9780130996510, Prentice Hall, Mar. 2000.

[3] Oh, S, H., and Yang, S, M. "A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks", Proceedings of 5th International Conference on Real-Time Computing Systems and Applications, page 31 – 36, Oct. 1998.

[4] Raveendran, B., et. al. "Variants of Priority Scheduling Algorithms for Reduced Context Switches in Real Time System", International Conference on Distributed Computing and Networking, pages 466 – 478, Dec. 2006.

[5] Buttazzo, G, C. "Rate Monotonic vs. EDF: judgement day", Real Time Systems, vol. 29, no. 1, pages 5 – 26, Jan. 2005.

[6] Mok, A, K. "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment", Ph.D.Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1983.

[7] Michael, B, J., et. al. "An Overview of the Rialto Real-Time Architecture", In Proceedings of the Seventh ACM SIGOPS European Workshop, pages 249-256, Sep. 1996.

[8] Wang, Y., and Saksena, M. "Scalable real-time system design using preemption thresholds", Proceedings of the 21st IEEE Symposium on Real-Time Systems, pages 25 – 34, Nov. 2000.

[9] Lee, C, G., et. al. "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling", Proceedings of IEEE Transactions on Computers, pages 700–713, Jun. 1998.

[10] Lee, S., et. al. "Cache-Conscious Limited Preemptive Scheduling", Real-Time Systems, pages 257–282, Nov. 1999.

[11] Vahid, S., et. al. "A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks", Transactions on Engineering, Computing and Technology, pages 19 – 23, Nov. 2005. http://enformatika.org/data/v9/v9-4.pdf.

[12] Stewart, D, B., and Khosla, P, K. "Real-Time Scheduling of Dynamically Reconfigurable Systems", Proceedings of the IEEE International Conference on Systems Engineering, pages 139 – 142, Aug. 1991.

[13] R. Xu, D. Mosse, and R. Melhem, "Minimizing Expected Energy in Real-time Embedded Systems", Proceedings of the 5th ACM international conference on Embedded software EMSOFT '05, Sep. 2005.

[14] H. Chung-Hsing, U. Kremer, M. Hsiao, "Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors", Proceedings of the International Symposium on Low power Electronics and Design, page 275-278, Aug. 2001.

[15] Dick, R, P., et. al, "Power analysis of embedded operating systems", In Proceedings of IEEE Design Automation Conference, Jun. 2000.

[16] Baynes K. et. al. "The Performance and Energy Consumption of Embedded Real-Time Operating Systems", IEEE Transactions On Computers, Vol. 52, NO. 11, Nov. 2003.

[17] Gopalakrishnan, R., and Parulkar, G, M. "Bringing real-time scheduling theory and practice closer for multimedia computing", Proceedings of the International Conference on Measurement and Modeling of Computer Systems. 1996.

[18] Marti, P., et. al. "Control performance of flexible timing constraints for Quality-of-Control Scheduling", In Proceedings of the 23rd IEEE Real-Time System Symposium, 2002.

[19] Cervin, A. "Integrated control and real-time scheduling", Doctoral Dissertation, ISRN LUTFD2/TFRT-1065- SE, Department of Automatic Control, Lund, 2003.

[20] Richard Gooch, "Linux Scheduler Benchmark Results", http://www.atnf.csiro.au/people/rgooch/benchmarks/linux-scheduler.html

[21] Tan, T, K., et. al. "Embedded Operating System Energy Analysis and Macro-modeling", Proceedings of the IEEE International conference on Computer Design: VLSI in Computers and Processors, May 2002.

[22] Acquaviva, A., et. al. "Energy characterization of embedded real-time operating systems", In Proceedings of COLP, 2001.