PEARLS: An Integrated Environment for Task Scheduling

Nilam Chand, Bindiya Mansharamani, Rafael Romero , Will Beazley, and Ştefan Andrei

Department of Computer Science,
Lamar Unviersity, Beaumont TX, 77710, USA
{nilamc, bindiyam, rafaelr, williamb, sandrei}@cs.lamar.edu

Abstract—The scheduling problem answers the question whether a given set of input tasks is schedulable (or feasible). It has been studied since '70s and impressive results have been revealed to the scientific community. This paper presents an implementation tool, PEARLS i.e, Pliable Earliest Deadline First, Rate Monotonic, Least Laxity Schedulers, based on some of the most significant existing schedulability analytical feasibility conditions and schedulability algorithms.If the input tasks set is feasible, our tool can simulate all the traditional scheduling methods:rate monotonic (RM), earliest deadline first (EDF) and least laxity(LL). Moreover, the tool is designed to handle both preemptive as well as non preemptive tasks. Both periodic and sporadic tasks and precedence contraints may also be considered in the input. Experimental results have been conducted for our tool implemented in the recent Java Development Kit (JDK) version 1.6 on a Pentium GHz system having 1MB memory. We tested our tool on several real-time systems specifications, and the obtained experimental results have confirmed that our tool is efficient and useful.

Index Terms—Real-Time Systems, Real-Time Scheduling, Schedulability Testing, Uniprocessor Schedulability, Multi-Processor Schedulability

1 Introduction

Decision versions of scheduling problems belong to NP and most of them are NP-hard [9], [8]. Researchers have concentrated their efforts to find efficient heuristics that can solve the scheduling problem in all circumstances: preemptable or non-preemptable, with or without precedence constraints, uniprocessor or multiprocessor, and so on. Some of the traditional methods are Rate-Monotonic (RM), Earliest-Deadline-First (EDF), and Least-Laxity (LL).

Our tool has a two-fold contribution. Firstly, it performs certain preliminary tests and, secondly, it gives the simulation of scheduling the set of input tasks.

Our paper integrates all these heuristics in only one single tool able to do analysis and simulation of an arbitrary set of tasks.

Given a set of (computer) tasks (also called processes), scheduling is to determine when to execute which task, thus determining the execution order of these tasks. For example, a set of tasks can T_1="do an expression evaluation", T_2="do a mouse move", T_3="do a document printing"excuted by the same processor". For the multiprocessors and distributed systems, scheduling means also to determine an assignment of these tasks to a specific processor, e.g., T_1 and T_3 above are done by processor P_1, and T_2 is done by processor P_2.

Scheduling is a central activity of a computer system, usually performed by the operating system. It is also necessary in many non-computer systems (e.g., assembly lines). In the case of real-time scheduling, the goal is to meet the deadline of every task by ensuring each task can complete execution by its specified deadline. The deadline is obtained from the environmental constraints imposed by the application.

Scheduling analysis is to determine whether a specific task of a set of tasks satisfying certain constraints can be successfully scheduled (completing execution of every task by its specified deadline) using a specific scheduler. These conditions are formulated as a *schedulability test*, that is to validate whether a given application can satisfy its specified deadlines when scheduled according to a specific scheduling algorithm. A schedulability test is often done at compile time, before the computer system and its tasks start their execution. If the test can be performed efficiently, then it can be done at run-time as an on-line test also called *simulation*. PEARLS is able to do both scheduling analysis and simulation.

1.1 Definitions

This next section defines the key terms used.

A *Real Time System* (RTS) is a system that can produce results in a certain time. Time is an important factor that determines that efficiency of an RTS. Simulation is often used to analyze the RTS. It is building a model (computer or physical model) of a system under study and implementing actions allowed in the system on the model. A simulator can carry out simulated executions of the simulated system and can display the outcomes of these executions[1]. A schedulability test validates whether the application under study meets a given set of constraints. The *Schedulablity utilization* is the maximum utilization allowed for a set of tasks that will guarantee a feasible scheduling of this task set.

A task T is a computer process, typically denoted thus: $T = (s, c, p, d, D)$, for simplicity, we sometimes avoid to mention all 5 of these elements, e.g., $T = (0, 1, 4, 4,)$ advoids mentioning the *absolute deadline* D. Task T has following members: *Start Time*, which defines when the task is ready for execution and denoted by s; *Computation Time*, that is the worst case execution time for a particular task and denoted by c; *Period*, which is the least required interval in the execution of a task and is denoted by p; *Relative Deadline*, which is a deadline relative to the start time and denoted by d; and Absolute Deadline, which is the absolute or the wall clock time and usually denoted by D (= s+d) [1].

2 Scheduling

Scheduling is the main activity in the implementation of a real time system. The synchronization between the processors and the task set is handled by the *scheduler*. It is the job of the scheduler to determine the performance of the tasks in the processor. For scheduling, we need a *scheduling algorithm* and a way to determine the *priority* of the tasks. There are two types of scheduling schemes, static/compile time scheduling and dynamic/runtime scheduling [5].

Scheduling can be done by using different techniques that focus upon critera such as *the number of processors*, *types of tasks* or *preemption schemes*. In a processor based scheduling where the focus is upon the number of processors one can use *uniprocessor* scheduling where the scheduling is done only once before execution at compile time. This method is inheriently static since the priorities are not reevulated overtime and thus do not change. Tasks may be preemptable with or without precedence constraints. Otherwise a *multiprocessor* scheduling scheme may or must be used, this, however, increases the complexity of the system and the scheduling. For these other types of scheduling schemes a priori knowledge of: Deadlines, Computation times, and the Start times of the tasks is necessary.

There are three main types of tasks in a RTS. A *single-instance task* is a task that executes only once. A *periodic task*, is a task that has many instances or iterations, and there is a fixed time (period) between two consecutive releases of the same task. A *sporadic task* has zero or more instances, and there exists a minimum interval between two consecutive releases of the same task.

When the scheduling allows preemption, there can be two variations, either there is *premptability or priority* based scheduling which relies on whether a process can be interrupted, prempted or otherwise is (or must be) non-preemptable, having critcal sections which are atomic. In preemptive scheduling the control is transferred to the higher priority job whenever it becomes ready, or we can have a *non-preemptive* based scheduling which is similar to the first-in, first-out (FIFO) scheme. The lower priority tasks are allowed to execute, even if the higher priority task is there in the ready queue.

2.1 Scheduling Techniques

Schedulers can be categorized into fixed-priority schedulers and dynamic-priority schedulers. In a fixed-priority scheduler, the priority of all instances of a task is the same whereas in a dynamic-priority scheduler, the priority of the instances of a task may vary. A popular example of fixed-priority scheduler is *Rate Monotonic* (RM). The tasks in this scheme are prioritized in accordance with their period. The RMS executes a task the moment an instance is ready with shortest period. A task J_i has a higher priority than task J_k if and only if the period p_i $< p_k$. *Earliest Deadline First* (EDF) and *Least Laxity* (LL) are the examples of dynamic-priority scheduler. EDF is designed to execute the task at the every instant it is ready and has the earliest absolute deadline (i.e. , $D = S + d$). If two or more tasks have the same deadline, EDF randomly selects any one for the next execution. It is a dynamic-priority scheduler since task priorities may change at run-time depending on the nearness of their absolute deadline. This scheduling can, for large task sets, achieve up to 69% of processor utilization. LL is often called Minimum-Laxity-First (MLF) or Least-Slack-Time-First (LST) algorithm. If $c(i)$ denotes the remaining computation time of a task at time i and $d(i)$ denotes the deadline of a task relative to the current time i, then the /emphlaxity (or slack) of a task at time i is $d(i) - c(i)$. In other words laxity is the maximum time a task can delay execution without missing its deadline anytime in the future. The scheduler executes at every instant the ready task with the smallest laxity. It has been proved that both LL and EDF are optimal for preemptable tasks [5], [1].

2.2 Schedulability Tests

Schedulability tests are performed to check if a given set of tasks satisfy specified constraints. There exist many sufficient and/or necessary conditions that can be used as good schedulability tests. this subsections contains a selection of ten such tests:

2.2.1 Schedulability Test 1

For an independent task set that is preemptable and periodic on a uniprocessor with

- $d_i \geq p_i$
- p_i are multiples of each other
- $U = c_1/p_1 + ... + c_n/p_n,$

The tasks are RM schedulable if and only if $U \leq 1$.

For example, the task set in Table 2.2.1 is RM-schedulable because $p_2 < p_1 < p_3,\ p_1 = 2p_2,\ p_3 = 4p_2$ and $U = c_1/p_1 + c_2/p_2 + c_3/p_3 = 1/4 + 1/2 + 2/8 = 1$.

JobId	s_i	c_i	p_i	d_i
1	0	1	4	4
2	0	1	2	2
3	0	2	8	8

TABLE 1
An Example of Task Set For Uniprocessor

2.2.2 Schedulability Test 2

For an independent task set that is preemptable and periodic on a uniprocessor, if U be the utilization then, The task set is schedulable if $U \leq \lim_{n\to\infty} n(2^{1/n} - 1) = \ln 2 \approx 0.6931$. [2]

For example, the task set in Table 2.2.1 does not satisfy Schedulability Test 2 because U $>$ 0.771

2.2.3 Schedulability Test 3

For a task set sorted in the increasing order of their period then if $w_i(t) = \sum_{\lim_{k=1}}^{i} c_k * p_t/p_k, 0 < t \leq p_i$, Task J_i is schedulable if and only if there exists a $w_i(t) \leq t$, for any time instant $t, t = kp_j, j = 1, ..., i, k = 1, ..., \lfloor pi/pj \rfloor$.

If the relative deadline $d_i \neq$ the period p_i, then p_i is replaced by $min(d_i, p_i)$ in the above formula.

For example, the task set in Table 2.2.1 is not schedulable, since at time =2 it cannot find a value of w $<$ t.

2.2.4 Schedulability Test 4

Given the value of c_i the computation time of task J_i, a set of n periodic tasks, $(d_i >= p_i)$, a necessary and sufficient condition for feasible scheduling of this task set on a uniprocessor is:

- $U = c_1/p_1 + ... + c_n n/p_p \leq 1$.

For example, the task set in table 2.2.1 is schedulable, since the deadlines are equal to the periods, and Utilization is $<= 1$.

Remark: for a task set containing some tasks whose relative deadlines d_i are less than their periods, no easy schedulability test exists with a necessary and sufficient condition.

2.2.5 Schedulability Test 5

Given a set of independent, preemptable, and periodic tasks on a uniprocessor, a feasible condition is $U = c_1/min(d_1, p_1) + ... + c_n/min(d_n, p_n) \leq 1$

For example, the task set in table 2.2.1 is schedulable, since the deadlines are equal to the periods, and Utilization is < 1.

2.2.6 *Schedulability Test 6*

States that if U be the total utilization of this tasks and the maximum relative deadline among the task deadlines be dmax and P is the least common multiple (LCM) of these tasks' periods, with the sum of computation times with absolute deadline less that t, s(t). Then this set of tasks is not EDF-schedulable if either of the following conditions holds:

- $U > 1$
- $t < min(P+d_{max}(U/(1-U))^{max_{1\leq i\leq n}}(p_i-d_i))$ such that $s(t) > t$.

For example, the task set in table 2.2.1 is not schedulable, since the second criteria fails.

JobId	c_i	p_i
1	10	50
2	15	70
3	10	40

TABLE 2
An Example Task Set with Sporadic Task

2.2.7 *Schedulability Test 7*

This test is for a task set with sporadic tasks. If pS and cS be the period and allocated time for the Differed Server. If U is the utilizationof the DS then

- $p_S < p_1 < ... < p_n < 2p_S$
- $p_n > p_S + c_S$
- This is RM-schedulable if the total utilization of this task set is at most $U(n) = (n-1)[(\frac{U_S+2}{U_S+1})^{\frac{1}{n-1}} - 1]$.

For example, the task set in table 2.2.6 is schedulable, since it satisfies all the criteria.

2.2.8 *Schedulability Test 8*

If the non preempt able task set M = the set of periodic tasks + set of sporadic tasks and the initial laxity l_i of task $T_i = d_i - c_i$, then each sporadic taskT_i be (c_i, d_i, p_i) is replaced by an equivalent periodic task$T'i = (c'i, d'i, p'i)$ as follows:

- $c'_i = c_i, d'_i = c_i, p'_i = min(p_i, l_i + 1)$.

2.2.9 *Schedulability Test 9*

For a multiprocessor, a schedule exists if the deadlines for a single instance tasks whose start times are equal, then the same set of tasks can be scheduled at runtime even if their start times were unknown or different. A priori knowledge of the deadlines and computation times in enough for LL algorithm.

For example, task set in Table 2.2.9, all the jobs have the same start time 0 and their deadlines and computation times are known in advance. Here when the number of processors is 2, EDF does not hold but LL does. Now even if we change the start time of job 3 to 2, LL does hold true.

JobId	s_i	c_i	d_i	p_i
1	0	1	2	4
2	0	2	3	4
3	0	4	4	4

TABLE 3
An Example Task Set For Multiprocessor

2.2.10 *Schedulability Test 10*

Given a set of k independent, preemptable (at discrete time instants), and periodic tasks on a multiprocessor system with n processors and $U = c_1/p_1 + ... + c_k/p_k \leq n$, $T = GCD(p_1, ..., p_k)$, and $t = GCD(T, T(c_1/p_1), ..., T(c_k/p_k)$.

If t is integral then, the given task set is schedulable.

For example, task set in Table 2.2.9, when the jobs can be preempted only at the integral time interval, the tasks can be scheduled because $T = GCD(4,4,4) = 4 \ and \ t = GCD(4,1,2,4) = 1$ which is an integral value.

2.3 Precedence Constraints

Order of execution of tasks can be specified by adding precedence constraints to the scheduling problem for a single instance tasks on a uniprocessor. A task precedence graph can be used that depicts tasks as nodes and a directed edge to indicate the precedence relationships between tasks. For example, T_i to T_j means that the job T_i must complete before job T_j can start.

3 Design

The tool was developed as project that was partially divided into 4 sub-projects and with the integration of the common elements. Given a set of tasks:

3.1 Community Project

- Find the utilization rate;
- Minimum number of processors required;
- Check the feasibility of the task set by simulating using the 3 schedulers, namely, EDF, RM and LL if appriopriate; and
- Create a user-friendly graphical frontend that presents all the features of tool to the user.

3.2 Sub-Project 1

- For a set of preemptable tasks, for uniprocessor case, analyze which of the Schedulability tests 1 to 7 are applicable.

3.3 Sub-Project 2

- The case of non-preempt able sporadic tasks for uniprocessor, check whether schedulability test 8 is applicable.

3.4 Sub-Project 3

- In the case of multiprocessors scheduling for non-preemptable tasks, analyze which of the Schedulability tests 9 and 10 are applicable.

3.5 Sub-Project 4

- Consider a set of precedence constraints for a uniprocessor, apply algorithm B (Dr Mok '84) to get the EDF-scheduling for the task set.
- Extend Algorithm B to include start times beyond 0 and periodic taks.

4 Tools Used

4.1 Hardware:

- CPU: 2.0 G Hz
- Memory: 1 GB
- OS: WinXP, Red Hat Linux

4.2 Software:

Programming Language Package

- Java Version
- Java(TM)
- Java Hotspot

4.3 Integrated Development Environment

(IDE)

- net beans IDE 5.5.1

5 Experiments

5.1 Algorithm B

Single instance non-premptable tasks may be scheduled by an algorithm known as Algorithm B, where:

1) Sort the tasks according to their deadlines in non-decreasing order and label the tasks such that $d_1 \leq d_2 \leq ... \leq d_n$.
2) Schedule task T_n in the time interval $[d_n - c_n, d_n]$.
3) While there is a task to be scheduled do:
   - Suppose S is the set of all unscheduled tasks whose successors have been scheduled.
   - Schedule as late as possible the task with the latest deadline S.
4) Shift the tasks toward time 0 while maintaining the execution order indicated in step 3.

6 Results

The tool delivers with a high level on confidence accurate results.

7 Conclusion

8 Future Work

The tool is able to handle EDF, LL and RM schedulers. It can be extended to be a universal scheduler implementing many more techniques and algorithms. If the verification model is also integrated in the tool, the tool will be unbeatable.

References

[1] M.K. Cheng, "Real-Time Systems Scheduling, Analysis and Verification", John Wiley & Sons, New Jersey. 2002.

[2] C.L. Liu and J. Layland, "Sqcheduling Algorithms for Multi-programming in a Hard-Real-Time Environment", Journal of the ACM, Vol. 20, no. 1, Jan. 1973, 46-61.

[3] C.M. Krishna and K.G. Shin, "Real-Time Systems", McGraw-Hill, New York, 1997.

[4] A.K. Mok, "The Design of Real-Time Programming Systems Based On Process Models",*In Proceedings of IEEE-CS Real-Time Systems Symposium* , 1984.

[5] B. Koch, "The Theory of Task Scheduling in Real-Time Systems: Compilation and Systematization of the Main Results", Universitat de llles Balears, Dec. 1999.

[6] K. Jeffay, D.F. Stanat , C.U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", *In Proceedings of twelfth IEEE Real-Time Systems Symposium*, San Antonio, Texas, Dec. 1991, 129-139.

[7] M.L. Dertouzos, A.K. Mok., “Mulitprocessor On-Line Scheduling of Hard-Real-Time Tasks”, *IEEE Transactions On Software Engineering*, Vol. 15, no. 12, Dec. 1989
[8] P. Brucker, “Scheduling Algorithms”, Fifth edition, Springer, Heidelberg, 2007.
[9] M.R. Garey, D.S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W.H. Freeman and Company, San Francisco, 1979.
[10] H. Kopetz, “Real-Time Systems Design Principles for Distributed Embedded Applications”, Vol. 395, Springer, 1997.
[11] J. Liu, “Real-Time Systems”, Prentice Hall, 2000.