IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

The Utilization Bound of Uniprocessor Preemptive Slack-Monotonic Scheduling is 50%

Björn Andersson

HURRAY-TR-071005

Version: 0

Date: 10-22-2007

The Utilization Bound of Uniprocessor Preemptive Slack-Monotonic Scheduling is 50%

Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: bandersson@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

Consider the problem of scheduling a set of sporadically arriving implicit-deadline tasks to meet deadlines on a uniprocessor. Static-priority scheduling is considered using the slack-monotonic priority-assignment scheme. We prove that its utilization bound is 50%.

The Utilization Bound of Uniprocessor Preemptive Slack-Monotonic Scheduling is 50%

Björn Andersson
IPP Hurray Research Group ISEP/IPP
Rua Dr. Antonio Bernardino de Almeida 431
4200-072 Porto, Portugal
bandersson@dei.isep.ipp.pt

ABSTRACT

Consider the problem of scheduling a set of sporadically arriving implicit-deadline tasks to meet deadlines on a uniprocessor. Static-priority scheduling is considered using the slack-monotonic priority-assignment scheme. We prove that its utilization bound is 50%.

Categories and Subject Descriptors

D.4 [**Operating Systems**]: Process Management

General Terms

Algorithms, Performance

Keywords

Real-time and embedded systems

1. INTRODUCTION

Consider the problem of preemptive scheduling of n sporadically arriving tasks on a uniprocessor. A task τ_i is given a unique index in the range $1..n$. A task τ_i generates a (potentially infinite) sequence of jobs. The time when these jobs arrive cannot be controlled by the scheduling algorithm and the time of a job arrival is unknown to the scheduling algorithm before the job arrives. It is assumed that the time between two consecutive jobs from the same task τ_i is at least T_i. Every job released from task τ_i requests to finish the execution of C_i time units at most T_i time units after its arrival. It is assumed that $0 \leq C_i \leq T_i$ and T_i and C_i are real numbers. The utilization is defined as $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. The utilization bound UB_A of an algorithm A is the maximum number such that if $U \leq UB_A$ then all tasks meet their deadlines when scheduled by A.

Static-priority scheduling solves this problem by assigning a priority to each task and every job is given the priority of the task that released the job. At run-time, a dispatcher selects, among jobs that have not yet finished execution, the job with the highest priority and this selected job executes on the processor. Typically, priorities to tasks are assigned using the Rate-monotonic (RM) [6] priority-assignment scheme; it assigns a task τ_j a higher priority than task τ_i if $T_j < T_i$. RM is well-explored. In particular, it is known that RM is an optimal static-priority scheduling algorithm, meaning that if there is any priority-assignment that causes all deadlines to be met then the assignment performed by RM will cause all deadlines to be met as well. It is also known that the utilization bound of RM is 69%.

Slack-monotonic (SM) is an alternative priority-assignment scheme which assigns the highest priority to the task with the least slack, that is, the deadline minus the execution time. Intuitively, a task with small slack must not be delayed too much in order to meet its deadline and hence it is given a high priority. SM is known to perform well in simulation studies of multiprocessor scheduling [7, 1] but unfortunately no formally proven results are available and it has not been analyzed for uniprocessor scheduling. In particular, no utilization bound of SM is known.

In this paper, we analyze SM on a uniprocessor. In our system model with sporadic scheduling, SM assigns task τ_j a higher priority than task τ_i if T_j-C_j < T_i-C_i. We do not assume any specific tie-breaking rules between tasks with the same slack. With this behavior, we prove the utilization bound of SM; it is 50%.

The remainder of this paper is organized as follows. Section 2 presents results within static-priority scheduling that we will use. Section 3 gives the understanding of the performance of SM in order to understand why the utilization bound of SM is 50% and in particular why it performs differently from RM. Section 4 presents a formal proof of the utilization bound of SM. Section 5 gives conclusions and open questions.

2. RESULTS WE WILL USE

Previous research [6] has explored a so-called *critical instant*. A critical instant of a task τ_i is an instant in time such that the response-time of task τ_i is maximized. It was found [6] that:

THEOREM 1. *A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.*

The result of Theorem 1 was originally used for RM but it is valid also for other priority-assignment schemes. This can be seen easily in [6] by the fact that Theorem 1 was

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

introduced before RM was introduced. By using Theorem 1, a schedulability condition, Theorem 2, has been derived [4]:

THEOREM 2. *The response time of task τ_i, denoted R_i, can be calculated as:*

$$R_i = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil \cdot C_j$$

where hp(i) is the set of indices of the tasks with a higher priority than τ_i.

The results in Theorem 2 is a cornerstone in the real-time scheduling theory for static-priority scheduling. It has been extended to cope with various practical concerns such as non-ideal effects of real-time kernels [3] and an alternative formulation has been proposed as well [5].

In our discussion later in the paper we will need Lemma 1, a slight variation of Theorem 2.

LEMMA 1. *If task τ_i misses its deadline then it holds that*

$$T_i < C_i + \sum_{j \in hp(i)} \lceil \frac{T_i}{T_j} \rceil \cdot C_j$$

PROOF. Follows from Theorem 2. □

3. UNDERSTANDING THE PERFORMANCE OF SM

Consider two tasks τ_1 and τ_2 with T_1=1,C_1=0.5+ϵ and T_2=0.5+ϵ and C_2=ϵ, where ϵ=0.01. With RM, τ_2 is assigned the highest priority and both tasks meet their deadlines (this can be easily seen because their utilization does not exceed 0.69.)

Let us now use SM on this task set. τ_1 is assigned the highest priority and τ_2 is assigned the lowest priority. Figure 1 illustrates the arrival times and the schedule when both tasks arrive simultaneously. It can be seen that τ_2 misses a deadline. It is clear that SM performs worse than RM.

In order to get the intuition of why the utilization bound of SM is 50%, consider the same task set but let ϵ be a positive number which does not exceed 0.5[1]. With SM we obtain that τ_1 has higher priority than τ_2. The utilization of the task set is $\frac{0.5+\epsilon}{1} + \frac{\epsilon}{0.5+\epsilon}$. τ_2 misses a deadline for every choice of ϵ with $0 \leq \epsilon < 0.5$. By letting $\epsilon \to 0$ we obtain a task set that misses a deadline and the utilization is marginally higher than 0.5. Hence, slack-monotonic has a utilization bound no greater than 50%. In the next section we will show that the utilization bound of SM is, in fact, 50%.

4. PROVING THE UTILIZATION BOUND OF SM

THEOREM 3. *If it holds for a task set τ that $\sum_{j=1}^{n} \frac{C_j}{T_j} \leq 0.5$ and τ is scheduled by SM on a uniprocessor then all deadlines are met.*

PROOF. The proof is by contradiction. Suppose that the theorem was false. Then there must be a task set τ such that $\sum_{j=1}^{n} \frac{C_j}{T_j} \leq 0.5$ and τ is scheduled by SM on a uniprocessor and a deadline was missed. Of all tasks that missed a deadline, let i denote the index of the task with the highest priority. We can now delete all tasks with a lower priority than i and still the theorem is false. Since τ_i missed a deadline it follows that an exact schedulability test for that task must be false. For this reason, we have (from Lemma 1) the following:

$$T_i < C_i + \sum_{j \in hp(i)} \lceil \frac{T_i}{T_j} \rceil \cdot C_j \tag{1}$$

where $hp(i)$ is the set of indices of tasks with higher priority than task τ_i. Let us define $hplong(i)$ and $hpshort(i)$ as:

$$hplong(i) = \{\tau_j \in hp(i) : T_j > T_i\}$$

and

$$hpshort(i) = \{\tau_j \in hp(i) : T_j \leq T_i\}$$

It is clear that the set $hp(i)$ can be partitioned into $hplong(i)$ and $hpshort(i)$. Applying this knowledge on Inequality 1 gives us:

$$T_i < C_i + \sum_{j \in hplong(i)} \lceil \frac{T_i}{T_j} \rceil \cdot C_j + \sum_{j \in hpshort(i)} \lceil \frac{T_i}{T_j} \rceil \cdot C_j$$

Taking an upper bound on one of the ceiling yields:

$$T_i < C_i + \sum_{j \in hplong(i)} \lceil \frac{T_i}{T_j} \rceil \cdot C_j + \sum_{j \in hpshort(i)} 2 \cdot \frac{T_i}{T_j} \cdot C_j$$

Dividing by T_i and rewriting yields:

$$1 < \frac{C_i}{T_i} + \sum_{j \in hplong(i)} \lceil \frac{T_i}{T_j} \rceil \cdot \frac{1}{\frac{T_i}{T_j}} \cdot \frac{C_j}{T_j} + \sum_{j \in hpshort(i)} 2 \cdot \frac{C_j}{T_j}$$

From the definition of $hplong(i)$, it follows that for those terms with index $j \in hplong(i)$, it holds that $\lceil T_i/T_j \rceil$=1. Using that yields:

$$1 < \frac{C_i}{T_i} + \sum_{j \in hplong(i)} \frac{1}{\frac{T_i}{T_j}} \cdot \frac{C_j}{T_j} + \sum_{j \in hpshort(i)} 2 \cdot \frac{C_j}{T_j}$$

Since $\sum_{j=1}^{n} \frac{C_j}{T_j} \leq 0.5$ and $i \leq n$ it follows that $\sum_{j=1}^{i} \frac{C_j}{T_j} \leq 0.5$. Let us now summarize what we have. We have that:

$$1 < \frac{C_i}{T_i} + \sum_{j \in hplong(i)} \frac{1}{\frac{T_i}{T_j}} \cdot \frac{C_j}{T_j} + \sum_{j \in hpshort(i)} 2 \cdot \frac{C_j}{T_j} \tag{2}$$

and

$$\sum_{j=1}^{i} \frac{C_j}{T_j} \leq 0.5 \tag{3}$$

and

$$\forall j \in hp(i) : T_j - C_j \leq T_i - C_i \tag{4}$$

and

$$\forall j \in \{1, 2, \ldots, n\} : 0 < C_j \leq T_j \tag{5}$$

Let us reason about Inequality 4. Since C_i is non-negative (follows from Inequality 5) we have:

$$\forall j \in hp(i) : T_j - C_j \leq T_i \tag{6}$$

rewriting it yields:

$$\forall j \in hp(i) : 1 - \frac{C_j}{T_j} \leq \frac{T_i}{T_j} \tag{7}$$

[1]This example was found by Marko Bertogna in July 2007.

Figure 1: A task set where SM misses a deadline and the utilization is 50%. An arrow pointing upwards indicate the arrival of a task. An arrow pointing downwards indicate the deadline of a task.

Applying Inequality 7 on Inequality 2 yields:

$$1 < \frac{C_i}{T_i} + \sum_{j \in hplong(i)} \frac{1}{1 - \frac{C_j}{T_j}} \cdot \frac{C_j}{T_j} + \sum_{j \in hpshort(i)} 2 \cdot \frac{C_j}{T_j}$$

From Inequality 3 and Inequality 5 we have that $\forall j \in hplong(i)$ $C_j/T_j \leq 0.5$. Applying this yields:

$$1 < \frac{C_i}{T_i} + \sum_{j \in hplong(i)} 2 \cdot \frac{C_j}{T_j} + \sum_{j \in hpshort(i)} 2 \cdot \frac{C_j}{T_j}$$

From Inequality 5 it follows that $C_i/T_i \leq 2 \cdot C_i/T_i$. Applying this yields:

$$1 < 2 \cdot \frac{C_i}{T_i} + \sum_{j \in hplong(i)} 2 \cdot \frac{C_j}{T_j} + \sum_{j \in hpshort(i)} 2 \cdot \frac{C_j}{T_j} \tag{8}$$

Dividing Inequality 8 by two and simplifying yields:

$$0.5 < \sum_{j=1}^{i} \frac{C_j}{T_j} \tag{9}$$

But Inequality 9 contradicts Inequality 3. Hence the theorem is correct. □

5. CONCLUSIONS AND FUTURE WORK

Very little is known about SM-scheduling on a uniprocessor. Nonetheless, we have proven the utilization bound of SM for scheduling tasks on a uniprocessor; the utilization bound is 50%.

It has been a long-standing open question in the real-time scheduling literature [2] whether there is an polynomial time-complexity algorithm for exact schedulability testing of RM. And no progress is currently in sight. For this reason, it is interesting to explore whether a polynomial time-complexity algorithm can be designed for non-optimal priority assignment schemes (such as SM).

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para Ciência e Tecnologia - FCT) and the ARTIST2 Network of Excellence on Embedded Systems Design.

6. REFERENCES

[1] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proc. of the International Conference on Real-Time Computing Systems and Applications*, pages 337–346, Cheju Island, South Korea, Dec. 12–14, 2000.

[2] S. Baruah and J. Anderson. Where is real-time and embedded systems research going? *IEEE Distributed Systems Online*, 1, 2000.

[3] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, 21(5):475–480, 1995.

[4] M. Joseph and P. Pandya. Finding response times in real-time systems. *BCS Computer Journal*, 29(5):390–395, 1986.

[5] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proc. of the IEEE Real Time Systems Symposium*, pages 166–171, Santa Monica, CA, USA, Dec. 5–7, 1989.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for the Computing Machinery*, 20:46–61, 1973.

[7] K. Ramamritham, J. A. Stankovic, and P. F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):184–194, 1990.