Erratum to
"Tardiness Bounds under Global EDF Scheduling on a Multiprocessor"

UmaMaheswari C. Devi
IBM Research – India
Bangalore, KA, India

James H. Anderson
Department of Computer Science
The University of North Carolina
Chapel Hill, NC, U.S.A.

Abstract

In [2] and [3], we derived closed-form expressions for upper bounds on tardiness that can be incurred by tasks of a sporadic task system under the preemptive and non-preemptive global *earliest-deadline-first* (EDF) scheduling algorithms on a multiprocessor. In those papers, we also outlined an iterative procedure for improving the bounds provided by the closed-form expressions. It has come to our attention that the iterative improvement outlined by us suffers from an error. In this note, we point out the error and offer a correction.

1 Error

In [2], we proved that under global preemptive EDF, on an m-processor system, each task T_k of a sporadic task system $\tau = \{T_1, T_2, \cdots, T_n\}$ incurs a tardiness not exceeding

$$\frac{(\sum_{T_i \in \mathcal{E}_{\max}(\Lambda-1)} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(\Lambda-1)} u_i} + e_k, \tag{1}$$

where $\Lambda = \lceil U_{sum} \rceil$ and $\mathcal{E}_{\max}(\ell)$ denotes a subset of ℓ tasks of τ with the highest execution costs (with ties resolved arbitrarily).[1] ($\mathcal{U}_{\max}(\ell)$ is defined analogously with respect to task utilizations.)[2] In [2], we also claimed that an optimized bound of

$$\frac{(\sum_{T_i \in \mathcal{E}_{\max}(\Lambda-1)} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(\Lambda-2)} u_i} + e_k, \tag{2}$$

can be shown to hold for T_k, a proof of which was provided later in the expanded version [3]. (Note that the number of task utilizations that (2) is dependent upon is lowered by one to $\Lambda - 2$ from $\Lambda - 1$ in (4).)

In both [2] (pp. 336, second paragraph) and [3] (pp. 164, second paragraph), we outlined a procedure for further optimizing the bound in (2). This optimization is motivated by the fact that while $|\mathcal{E}_{\max}(\Lambda-1) \cup \mathcal{U}_{\max}(\Lambda-2)|$ could be up to $2\Lambda - 3$, by the tardiness bound derivation, only exactly $\Lambda - 1$ tasks contribute to both the execution-cost summation involving $\Lambda - 1$ tasks and the utilization summation involving $\Lambda - 2$ tasks, in the numerator and denominator, respectively,

[1] In this note, we assume that all ties in task selections are resolved using task indices.

[2] [2] actually assumed a fully-utilized system and derived bounds with $m-1$ tasks (as opposed to $\Lambda - 1$ tasks) in both the summations. The assumption was relaxed later in [3], leading to an improvement for systems that are not fully utilized.

of (2). In other words, the summations in (2) over the highest execution costs as well as the highest utilizations are upper bounds, specified so for convenience, in order to arrive at a closed-form expression. However, by the tardiness bound derivation, it is sufficient to consider exactly $\Lambda - 1$ tasks with the highest possible cumulative *lag* (see [3, Sec. 3.1]) at the end of a non-busy interval of interest for contribution to the two summations (regardless of whether the execution costs and utilizations of those tasks are among the highest). It is known that at most $\Lambda - 1$ tasks can be executing and have a positive lag at the end of a non-busy interval, and at least one of those $\Lambda - 1$ tasks is not tardy ([3, Lemma 4]). Further, it is known that if Θ_k denotes a tardiness bound of T_k, then $\mathsf{lag}(T_k, t)$ is at most $(\Theta_k - e_k) \cdot u_k + e_k$, at all relevant t, if T_k is tardy, and at most e_k, otherwise ([3, Lemma 5]). Finally, by the tardiness bound derivation, the tardy tasks contribute to both the utilization and execution-cost sums, while the non-tardy task to just the execution-cost sum.

Based on the above facts, we proposed an iterative improvement to the bound in (2), with the objective of choosing $\Lambda - 1$ tasks with the highest cumulative lag bounds such that $\Lambda - 2$ of those are tardy, while one task is not. In what follows, we let τ_{tardy} denote a set of $\Lambda - 2$ tasks that contribute to both the utilization and execution-cost sums, and τ_{ntrdy}, a singleton set containing a single task, contributing to just the execution-cost sum. Our iterative algorithm is as follows: Letting x denote the first term, $\frac{(\sum_{T_i \in \mathcal{E}_{\max}(\Lambda-1)} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max}(\Lambda-2)} u_i}$, of (2), choose $\Lambda - 2$ tasks with the highest value for $x \cdot u_k + e_k$ for inclusion in τ_{tardy}[3] for contribution to both the utilization and execution-cost sums and an additional task with the highest execution cost of the remaining $n - \Lambda + 2$ tasks for inclusion in τ_{ntrdy} to contribute additionally to the execution-cost sum, and use the resultant sums in (2) to improve x. Repeat the procedure successively until two consecutive iterations yield the same set of tasks.

While the above algorithm may look correct superficially, its task-selection method may not always pick the highest $\Lambda - 1$ task lag bounds, subject to the constraints specified above, and hence, the correct tasks, to contribute to the two sums. This is illustrated by the following example.

Example. Consider a task set composed of the following tasks with $U_{sum} = 4.0$ scheduled under EDF on four processors: $T_1(15, 150), T_2(15, 150)$, $T_3(9, 18), \ldots, T_8(9, 18)$, $T_9(1, 10), \ldots, T_{16}(1, 10)$. For this task set, to start with, $x = \frac{15+15+9-1}{4-2*0.5} = 12.67$, $x \cdot u_k + e_k$ for T_1 and T_2 are 16.267, while it is $15.3\bar{3}$ for T_3 to T_8, and 2.267 for T_9 to T_{16}. The algorithm described above will therefore add T_1 and T_2 to τ_{tardy} to contribute to both the execution-cost as well as the utilization sums. Since T_3 to T_8 have the highest execution costs of the remaining tasks, the algorithm chooses T_3 as the additional task for the execution-cost sum, for a revised $x = \frac{15+15+9-1}{4-(0.1+0.1)} = 10.0$. The set of tasks contributing to the two sums is not altered in the next iteration, and so the algorithm terminates with tardiness bounds of $x + e_1 (= e_2) = 10.0 + 15.0 = 25.0$ for T_1 and T_2, 19.0 for $T_3, \ldots, T_8$, and 11.0 for $T_9, \ldots, T_{16}$.

In the above example, note that though $x \cdot u_3 + e_3 < x \cdot u_2 + e_2$, we have $x \cdot u_3 + e_3 + e_2$, *i.e.*, $15.3\bar{3} + 15 = 30.3\bar{3}$ larger than $x \cdot u_2 + e_2 + e_3$, which is 25.267. Therefore, assuming T_1 and T_2 to be the tardy tasks, and T_3 as the non-tardy task does not yield the highest $\Lambda - 1$ task lag bounds, and hence, does not correctly bound x. On the other hand, a correct upper bound is obtained by letting T_1 and T_3 contribute to both the utilization and execution cost sums, and T_2 to just the execution-cost sum, for an x of 11.176. Each task's correct tardiness bound is thus larger by 1.176 time units.

2 Correction

The problem with the above algorithm is that it ignores the possibility that though $x \cdot u_i + e_i$ is smaller than $x \cdot u_j + e_j$, $x \cdot u_i + e_i + e_j$ could be larger than $x \cdot u_j + e_j + e_i$. In other words, by deciding on the membership to τ_{tardy} based on the highest task lags under tardy conditions, not all possible combinations of task lags are considered in determining the maximum possible lag due to any subset of $\Lambda - 1$ tasks (such that one of them is not tardy) and hence, the membership to the two task sets. In the above example, by including T_2 in τ_{tardy} based on task lag bounds under tardy conditions alone, the possibility of including T_3 to τ_{tardy} and T_2 to τ_{ntrdy}, which has a higher cumulative task lag is precluded. This error can be corrected by choosing the tasks not in two steps but in a single step in each iteration by modifying the task-selection

[3]Follows from the lag bound of $(\Theta_k - e_k) \cdot u_k + e_k$ for a task T_k with a tardiness bound of Θ_k, specified above, and the fact that letting the expression in (2), which is a tardiness bound for T_k, denote Θ_k, $\Theta_k - e_k$ is given by the first term of (2).

rule to choose $\Lambda - 1$ tasks that maximize

$$e_i + \sum_{\substack{T_j \in \text{subset of } \Lambda - 2 \\ \text{tasks in } \tau \setminus \{T_i\}}} (x \cdot u_j + e_j), \tag{3}$$

and letting the maximizing set's T_j's ($\Lambda - 2$ such tasks) contribute to both the execution-cost and utilization sums (by including them in τ_{tardy}), and its lone T_i to contribute to just the execution-cost sum (be added to τ_{ntrdy}). As proof of correctness of the above selection rule, we claim the following.

Claim 1 *Let $x + e_i$ denote a tardiness bound of task T_i in τ. Then, at all relevant t, an upper bound to the maximum cumulative lag of any subset of $\Lambda - 1$ tasks of τ such that one of them is non-tardy is given by the maximum value of (3) taken over all $T_i \in \tau$.*

Proof: Let τ_{tardy} and τ_{ntrdy} denote the set of $\Lambda - 2$ T_j's and the lone T_i, respectively, in a set of tasks maximizing (3). Since the tardiness of T_k is at most $x + e_k$, by ([3, Lemma 5]), $\mathsf{lag}(T_k, t)$ is at most $x \cdot u_k + e_k$, at all relevant t, if T_k is tardy, and at most e_k, otherwise. Suppose the claim is not true. Then, there exist task sets τ'_{tardy} and τ'_{ntrdy} such that $|\tau'_{\text{ntrdy}}| = 1$ and $|\tau'_{\text{tardy}}| = \Lambda - 2$ and $\sum_{T_i \in \tau'_{\text{ntrdy}}} e_i + \sum_{T_j \in \tau'_{\text{tardy}}} x \cdot u_j + e_j > \sum_{T_i \in \tau_{\text{ntrdy}}} e_i + \sum_{T_j \in \tau_{\text{tardy}}} x \cdot u_j + e_j$. But then, (3) cannot be maximized with the lone task in τ_{ntrdy} for T_i and the tasks in τ_{tardy} for T_j's, a contradiction. ■

In the above example, starting with $x = 12.67$ given by (2), we have (3) maximized with $\tau_{\text{ntrdy}} = \{T_2\}$ and $\tau_{\text{tardy}} = \{T_1, T_3\}$. x is therefore revised to $\frac{15+15+9-1}{4-(0.1+0.5)} = \frac{38}{3.4} = 11.176$. The two sets are unaltered in the next iteration and the algorithm terminates.

In [2] and [3], we also proved a tardiness bound of

$$\frac{(\sum_{T_i \in \mathcal{E}_{\max(\Lambda)}} e_i) - e_{\min}}{m - \sum_{T_i \in \mathcal{U}_{\max(\Lambda-1)}} u_i} + e_k, \tag{4}$$

for task T_k under non-preemptive global EDF. We also mentioned that an iterative procedure as specified for preemptive EDF can be applied to improve the bounds under non-preemptive EDF also. It should be noted that a corrected version of the iterative algorithm should be applied in the context of non-preemptive EDF also.

Experimental evaluations of the iterative algorithm presented in [2] and [3] are with the incorrect version and hence should be slightly under-estimating the bounds.

The same error has been carried over to and occurs in the description of the iterative improvement in [1] also.

However, a corrected version has been used in the experimental evaluation in [4] that compares the iterative improvement to (2) with another iterative algorithm proposed in [4] to further improve the tardiness bounds under global EDF.

Acknowledgement. We are thankful to Jeremy Erickson and Sanjoy Baruah for bringing this error to our attention.

References

[1] U. Devi. *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, University of North Carolina at Chapel Hill, December 2006. Under Preparation.

[2] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, pages 330–341, December 2005.

[3] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. *Real-Time Systems*, 38(3):133–189, February 2008.

[4] J. Erickson, U. Devi, and S. Baruah. Improved tardiness bounds for global EDF. In Submission, Jan 2010.