A Clarification of Link-Based Global Scheduling

Björn B. Brandenburg and James H. Anderson

The analysis of link-based global scheduling presented in [1, 2] rests on the assumptions that **(i)** schedulability is established using a global schedulability test assuming *fully preemptive, independent* tasks,[1] and that **(ii)** any additional delays incurred due to non-preemptive sections are accounted for by *inflating* the worst-case execution parameter of *each* task prior to applying the test. We clarify these requirements, which were implicitly assumed (and met) in [1, 2], but not explicitly stated, in the following.

Inflation-based analysis. Suppose we are given a global schedulability test GST_P for a global scheduling policy P that assumes fully preemptive, independent tasks and an *original* set of n sporadic tasks $\tau = \{T_1, T_2, \ldots, T_n\}$, where each task T_i has a worst-case execution time of e_i, a period p_i, and a maximum non-preemptive section length cs_i. Deadlines may be arbitrary. For simplicity, we initially assume that tasks do not self-suspend.

If $cs_i \neq 0$ for any T_i, then GST_P obviously cannot be directly applied to τ. To indirectly determine the schedulability of τ using GST, a per-task *inflation term* b_i is determined to construct an *inflated* task set $\tau' = \{T_1', T_2', \ldots, T_n'\}$, where $e_i' = e_i + b_i$ for each T_i' and all tasks are *fully preemptive*. The periods of the inflated tasks remain unchanged ($p_i' = p_i$).

The inflation term b_i is usually called a "blocking term" or said to bound "pi-blocking" [1, 2]. We choose a different term in this document to stress that—even though b_i serves to account for "delays due to non-preemptive sections"—the inflation term is *not* identical to (and does not bound) the maximum cumulative duration of "priority inversion" incurred by any job of T_i, at least not under all definitions of "priority inversion." For the purpose of the following argument, a *priority inversion* exists if a higher-priority job J_h is pending but not scheduled while a lower-priority job J_l is scheduled, thereby violating the normal preemption policy. We revisit the question of how to define "priority inversion" on page 3.

We require each per-task inflation term b_i to be chosen such that the following property holds:

(I) *if there exists a job arrival sequence such that a task in τ misses a deadline under scheduling policy P* ***in the presence of non-preemptive sections****, then there also exists a job arrival sequence such that a task in τ' misses a deadline if scheduled under P* ***even if all tasks are fully preemptive****.*

If Property (I) holds, and if τ' is schedulable according to GST_P when scheduled with policy P (on m processors), then τ is also schedulable under P (on m processors). This follows from the fact that, if τ is *not* schedulable (*i.e.*, if some $T_i \in \tau$ can miss a deadline in some schedule), then by Property (I) there also exists a schedule in which some $T_j' \in \tau'$ misses a deadline, which implies that τ' must fail GST_P.

The inflation approach thus allows us to reuse existing schedulability tests that assume fully preemptive tasks to derive a schedulability test that is sufficient in the presence of non-preemptive sections. The key question, then, is how to find (non-trivial) inflation terms that ensure Property (I)?

Link-based scheduling. Let $lp_i \subset \tau$ denote the set of tasks that may be preempted by T_i under policy P (*i.e.*, lower-priority tasks under fixed-priority scheduling and tasks with longer relative deadlines under EDF). Under link-based scheduling, Property (I) holds if $b_i \triangleq \max\{cs_l \mid T_l \in lp_i\}$ for each task T_i.

To support this claim, we show how to transform a link-based schedule S of τ in which an arbitrary job J_d misses its deadline at time t_d into a corresponding schedule S' of τ' so that the response time of the corresponding inflated job J_d' in S' is no less than that of J_d in S.

[1] Since at the time no other schedulability tests were available for G-EDF, the global scheduling policy considered in [1, 2].

Initially, let S' be a copy of S. Jobs released after t_d obviously have no impact on J'_d; they can thus be safely removed from S'. Next, we remove all priority inversions involving any of the remaining jobs in S'.

Under link-based scheduling, at any point in time, the m highest-priority jobs are linked (if that many exist). From this invariant we immediately obtain that, if a job J_i incurs a priority inversion at time t, then either **(a)** J_i is linked (but not scheduled) or **(b)** there exist m higher-priority jobs.

We remove case-(a) priority inversions from S' by repeatedly applying the following transformation.

Case-(a) priority inversions. Consider the latest point in time t (if any) at which any job J'_i is subject to a case-(a) priority inversion in S': at time t, J'_i is linked but not scheduled because a lower-priority job J'_l is executing non-preemptively on the processor to which J'_i is linked. There are two cases to consider, depending on whether J'_l happens to be J'_d.

If $J'_l \neq J'_d$, then we transform the schedule by

1. shortening J'_l's non-preemptive section by one time unit (*i.e.*, allowing J'_l to be preempted by J'_i one time unit earlier),
2. subtracting one time unit from J'_l's execution requirement, and
3. adding one time unit to J'_i's execution requirement, thereby "filling up" the allocation freed by reducing the demand of J'_l.

This has the effect of shifting the preemption of J'_l to an earlier instant by one time unit, and as a result the priority inversion between J'_i and J'_l at time t is removed. The transformation does not reduce the response time of J'_i, and hence also not the response time of any job other than J'_l. (As a corner case, after the transformation, J'_l's response time in S' may be less than $J'_l s$ response time in S if J_l completes immediately at the end of its non-preemptive section. Since $J'_d \neq J'_l$, this corner case is irrelevant.)

If $J'_l = J'_d$, then we simply skip step (2), that is, we leave J'_d's execution requirement unchanged, which ensures that J'_d's response time is not lessened. (Any possible perturbation of the schedule past t_d is irrelevant to the proof.)

We repeat this transformation until no instance of case-(a) priority inversion is left in S'. Next, consider case-(b) priority inversions in S.

Case-(b) priority inversions. In fact, the removal of case-(a) priority inversions from S' has implicitly also removed all instances of case-(b) priority inversion, which can be observed from the following argument.

Suppose there still exists a case-(b) priority inversion in S' even after the removal of all case-(a) priority inversions. We show that this implies the existence of a case-(a) priority inversion, thereby contradicting the initial assumption that all case-(a) priority inversions have already been removed.

If there exists a case-(b) priority inversion at a time t in S' with respect to a pending (but not scheduled) job J_i, then by the definition of case (b) there also exist m higher-priority, linked jobs at time t. That is, all processors are linked to higher-priority tasks.

At the same time, for J_i to incur a priority inversion at time t, on at least one processor Π_k a lower-priority job must be executing at time t. (Otherwise m higher-priority jobs are scheduled and J_i does not incur a priority inversion at time t.)

Let J_h denote the higher-priority job linked to Π_k at time t. Since J_h has an even higher priority than J_i, J_h necessarily also incurs a priority inversion at time t.

We observe that J_h is linked but not scheduled at time t, and that J_h incurs a priority inversion at time t—this, however, constitutes a case-(a) priority inversion.

Therefore, after case-(a) priority inversion elimination, S' is priority-inversion-free, which is equivalent to assuming fully preemptive jobs.

T_i	e_i	cs_i	b_i
T_1	3	0	10
T_2	7	0	10
T_3	3	0	10
T_4	7	0	10

T_i	e_i	cs_i	b_i
T_5	3	0	10
T_6	7	0	10
T_7	20	0	10
T_8	30	10	0

Table 1: Parameters of the example task set depicted in Fig. 1. Periods are irrelevant and have been omitted.

Example. A complete example of the transformation is given in Fig. 1, which shows two example schedules S and S' of a task set $\tau = \{T_1, \ldots, T_8\}$ due to Davis *et al*. with parameters as given in Table 1.

Fig. 1(a) depicts a possible schedule of τ (assuming link-based fixed-priority scheduling on $m = 2$ processors, where T_1 has the highest priority) in which T_7 incurs repeated case-(b) priority inversions.

Fig. 1(b) shows the transformed schedule S' of the inflated task set τ' after all case-(a) priority inversions have been removed as described above. In particular, in inset (b), the execution costs of T_1', T_3', and T_5' have been inflated to account for T_8's non-preemptive sections (which have been removed from T_8'). The transformation has explicitly removed case-(a) priority inversions (incurred by T_1, T_3, and T_5 in S), and thereby implicitly also all case-(b) priority inversions (incurred by T_7 in S).

In the resulting schedule S', all tasks are fully preemptive and there exist no more priority inversions. Furthermore, the response time of each task in Fig. 1(b)—with the exception of T_8'—is no less than the response time of the corresponding task in Fig. 1(a). If the response time of T_8' is relevant, that is, if it must not be shortened for the sake of establishing Property (I) for T_8, then the schedule shown in Fig. 1(c) results (which is truncated at time 50).

The example in Fig. 1 demonstrates that a sound schedulability test cannot claim τ' to be schedulable if the scenario depicted in Fig. 1(a) causes a task in τ to miss a deadline.

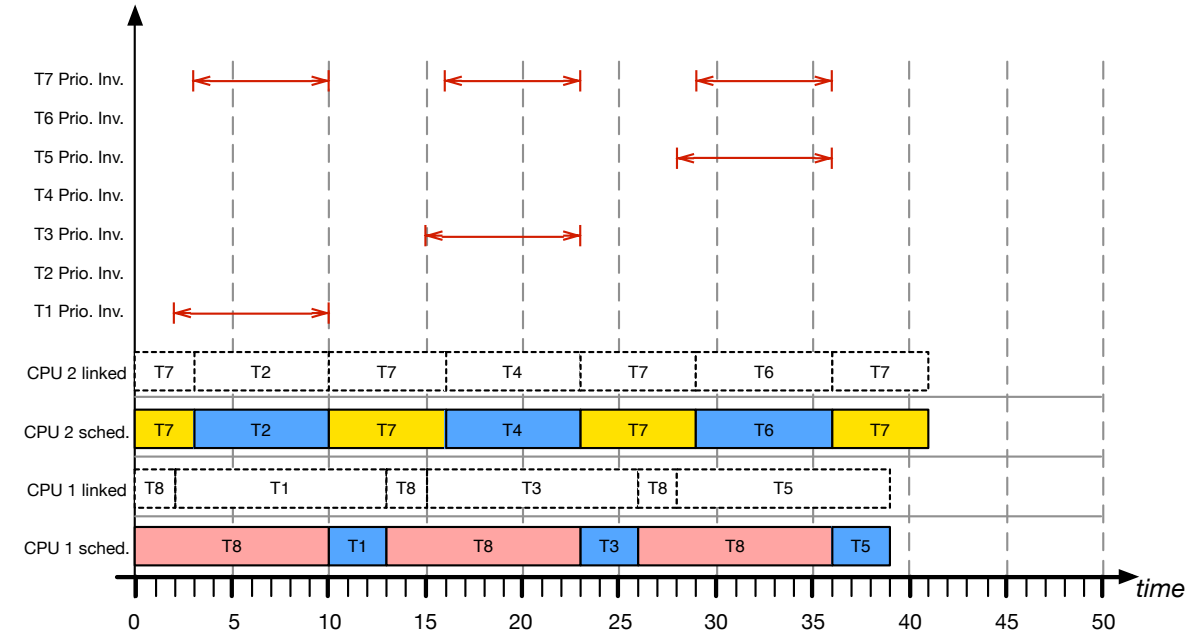
Maximum inflation. From the link-based scheduling algorithm [2], we observe that, with respect to each job (and in the absence of self-suspensions), an interval of case-(a) priority inversion occurs at most once and only immediately upon release, and its duration is bounded by the length of one lower-priority non-preemptive critical section. Therefore, case-(a) elimination increases each job J_i''s execution requirement by at most $b_i = \max\{cs_l \mid T_l \in lp_i\}$ time units.

The resulting schedule S' is thus a valid, fully preemptive schedule of τ' in which J_d''s response time is no shorter than that of J_d in S. This establishes Property (I).

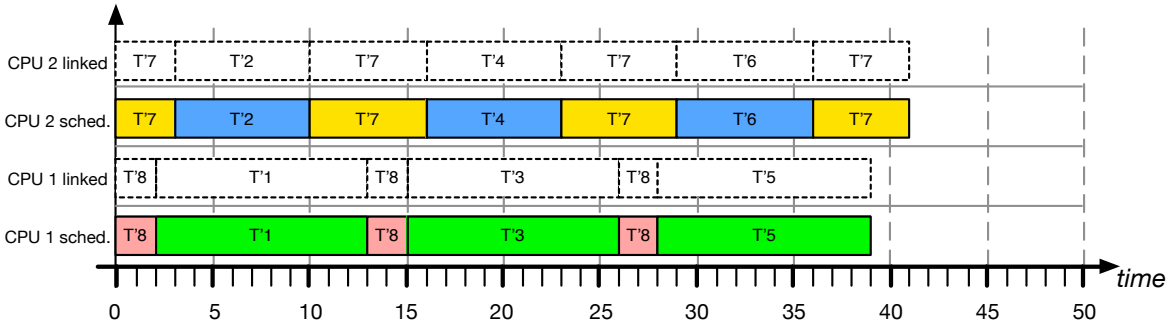
Self-suspensions. Allowing self-suspensions (due to either I/O or suspension-based locking-protocols) does not change the structure of the argument. A job that resumes from a self-suspension is vulnerable to case-(a) priority inversion similar to a newly released job. Thus, if η_i bounds the maximum number of self-suspensions of T_i, then setting $b_i \triangleq (1 + \eta_i) \cdot \max\{cs_l \mid T_l \in lp_i\}$ is sufficient for the argument to hold.[2]

Relation to suspension-oblivious pi-blocking. The previous descriptions of link-based schedulability analysis [1, 2] refer to the per-task inflation charge b_i as "pi-blocking," and claim that link-based scheduling ensures $O(1)$ pi-blocking. In fact, this is accurate under the *suspension-oblivious* definition of pi-blocking, but it is not true under the *suspension-aware* definition of pi-blocking [2–4].

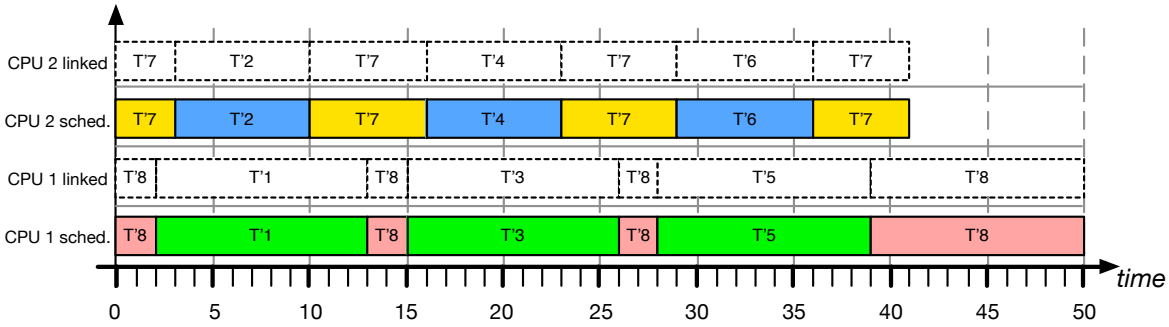
[2]It is possible to derive more accurate bounds that take the actual frequency of long non-preemptive sections into account.



(a) The original schedule S. Task T_8 is assumed to execute non-preemptively during $[0, 10)$, $[13, 23)$, and $[26, 36)$.



(b) The schedule S' after case-(a) priority inversions have been removed.



(c) The schedule S' after case-(a) priority inversions have been removed if the response time of T'_8 must not be shortened.

Figure 1: An example schedule (due to Davis *et al.*) before and after case-(a) priority inversion removal.

In short, a job J_i incurs suspension-oblivious priority inversion at time t if and only if J_i is pending but not scheduled and fewer than m higher-priority jobs are *pending*. That is, to rule out this kind of priority inversion, m higher-priority jobs merely need to exist, but they do not necessarily need to be scheduled.

In contrast, a job J_i incurs a suspension-aware priority inversion at time t if and only if J_i is pending but not scheduled and fewer than m higher-priority jobs are *scheduled*, which matches the definition of priority inversion adopted on page 1.

It may be surprising to talk about "suspensions" in the context of non-preemptive sections (where tasks don't actually suspend). However, suspension-oblivious schedulability analysis relies on the inflation of execution time parameters exactly as described herein. Therefore, in hindsight, a perhaps more fitting description of suspension-oblivious schedulability analysis would have been "inflation-based schedulability analysis."

This relationship also illustrates why it is useful to distinguish between suspension-oblivious (*i.e.*, "inflation-based") and suspension-aware (*i.e.*, "accurate-processor-demand-based") schedulability analysis: the existence of higher-priority delayed jobs is "useful knowledge" under suspension-oblivious analysis precisely because their inflated counterparts can be arranged to take up a processor. In other words, their mere existence guarantees that it is possible to establish Property (I) by converting any delays and inversions into additional execution of fully preemptive *higher-priority* tasks, which allows us to charge these delays as regular interference.

Summary. Link-based scheduling is inherently tied to inflation-based schedulability analysis, which is analogous to suspension-oblivious schedulability analysis [2–4]. The "trick" underlying link-based scheduling is not to avoid priority inversion altogether (which is impossible in general), but to arrange untimely preemptions (the root cause of priority inversion) such that most delays arise only when m higher-priority tasks are present, which allows any such delays to be conveniently attributed to inflated demand. In short, link-based scheduling ensures analysis conditions that are favorable for inflation-based analysis.

The bound on suspension-oblivious pi-blocking characterizes the magnitude of the inflation term required in the worst case to construct a schedule that establishes Property (I). As argued in this document, a charge of $b_i = O(1)$ per-task maximum suspension-oblivious pi-blocking is sufficient for Property (I) to hold. With eager preemptions, that is, without link-based scheduling, much more pessimistic bounds must be assumed, as described in [2].

References

[1] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In *RTCSA'07*, 2007.

[2] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, UNC Chapel Hill, 2011.

[3] B. Brandenburg and J. Anderson. Optimality results for multiprocessor real-time locking. In *RTSS'10*, 2010.

[4] B. Brandenburg and J. Anderson. The OMLP family of optimal multiprocessor real-time locking protocols. *Design Automation for Embedded Sys*, 17(2), 2013.