An Asymptotically Optimal Real-Time Locking Protocol for Clustered Scheduling under Suspension-Aware Analysis

Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)
bbb@mpi-sws.org

1. OPTIMAL LOCKING PROTOCOLS

The purpose of real-time locking protocols is to limit *priority inversions* [5], which, intuitively, occur when a high-priority task is delayed by a lower-priority task. Such locking-related delay, also called *priority inversion blocking* (*pi-blocking*), is problematic in real-time systems because it can result in deadline misses. However, some pi-blocking is unavoidable when using locks and thus must be bounded and accounted for during schedulability analysis.

Clearly, an "optimal" locking protocol should minimize pi-blocking to the extent possible. Formally, a locking protocol is asymptotically optimal if it ensures that, for *any* task set, maximum pi-blocking is bounded within a constant factor of the minimal pi-blocking unavoidable in *some* task set [3]. Interestingly, there exist two classes of schedulability analysis that yield *different* lower bounds: under *suspension-oblivious* (*s-oblivious*) analysis, $\Omega(m)$ pi-blocking is fundamental, whereas under *suspension-aware* (*s-aware*) analysis, $\Omega(n)$ pi-blocking is unavoidable in the general case [2, 3], where m and n denote the number of processors and tasks, respectively. As the names imply, the key difference is that suspensions are accounted for explicitly under s-aware analysis, whereas they are (pessimistically) modeled as execution in the s-oblivious case.

For the simpler s-oblivious case, asymptotically optimal locking protocols have been designed for partitioned, global, and clustered *job-level fixed-priority*[1] (JLFP) scheduling [4]. The s-aware case, however, is much less understood: only two asymptotically optimal protocols for partitioned JLFP scheduling are known so far [2, 3].

In contrast, the problem of optimal s-aware locking under global and clustered JLFP scheduling has remained open to date. While it was initially assumed [3] that Block *et al.*'s *Flexible Multiprocessor Locking Protocol* (FMLP) [1]—which is based on $O(n)$ FIFO queues—is asymptotically optimal under global scheduling, it was later observed [2] that this holds only under some, but not all global JLFP schedulers. In fact, it was shown that both *priority inheritance* [5] and (unconditional) *priority boosting* [5], one of which is used in each previously proposed s-aware protocol to expedite the completion of critical sections by temporarily raising the effective priority of lock-holding jobs, can give rise to non-optimal $\Omega(\Phi)$ pi-blocking [2], where Φ is the ratio of the longest and the shortest period (and unbounded in general). Finally, to the best of our knowledge, no asymptotically optimal s-aware locking protocol for the general case of clustered JLFP scheduling has been proposed in prior work.

[1] The class of job-level fixed-priority schedulers includes both classic fixed-priority and EDF scheduling. Clustered scheduling is a generalization of both partitioned and global scheduling under which disjoint clusters of processors are scheduled globally.

2. THE GENERALIZED FMLP+

We have solved the problem of asymptotically optimal s-aware locking under clustered JLFP scheduling by devising a new progress mechanism that circumvents the $\Omega(\Phi)$ bound mentioned above.

Priority boosting/inheritance is susceptible to $\Omega(\Phi)$ pi-blocking because a high-priority job J_h can be repeatedly preempted by critical sections that were started *after* J_h was already scheduled [2]. This is avoided by the following *restricted boosting* mechanism. Let $t_r(J_i)$ denote the latest point in time that a job J_i either **(i)** was *released*, **(ii)** *resumed* from a locking-unrelated self-suspension, or **(iii)** *requested* (*i.e.*, tried to lock) a resource. A priority-boosted, lower-priority job J_l may preempt a higher-priority, un-boosted job J_h only if $t_r(J_l) < t_r(J_h)$. This implies that J_h is preempted only by critical sections that were in progress when J_h became available for scheduling, of which there are at most $n - 1 = O(n)$ (*i.e.*, one per task, assuming tasks are sequential). Further, it can be shown that lock-holder progress is guaranteed in the sense that at least one lock-holder is always scheduled (if any exist). By scheduling priority-boosted jobs in order of increasing t_r timestamps (*i.e.*, FIFO w.r.t. lock request time), $O(n)$ pi-blocking per request is achieved.

Restricted boosting generalizes the idea underlying the partitioned *FIFO Multiprocessor Locking Protocol* (FMLP$^+$) [2], namely to order lock-holding jobs by request time. Combined with $O(n)$ FIFO queues, we obtain a locking protocol that is asymptotically optimal under clustered (and hence also under global) JLFP scheduling.

3. OUTLOOK

We believe the proposed protocol offers improved schedulability, in particular if Φ is large, and are in the process of deriving fine-grained (*i.e.*, non-asymptotic) pi-blocking bounds. We plan to implement and evaluate the protocol in LITMUS$^{\mathrm{RT}}$ and expect overheads to be relatively low due to the simplicity of FIFO queuing and timestamp-based preemption checks. Further, we will explore the potential of an analogously designed "restricted inheritance" mechanism.

References

[1] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In *Proc. RTCSA*, 2007.
[2] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2011.
[3] B. Brandenburg and J. Anderson. Optimality results for multiprocessor real-time locking. In *Proc. RTSS*, 2010.
[4] B. Brandenburg and J. Anderson. The OMLP family of optimal multiprocessor real-time locking protocols. *Design Automation for Embedded Systems*, to appear, 2012.
[5] R. Rajkumar. *Synchronization In Real-Time Systems—A Priority Inheritance Approach.* Kluwer Academic Publishers, 1991.