Using Lock Servers to Scale Real-Time Locking Protocols: Chasing Ever-Increasing Core Counts (Artifact)*

Catherine E. Nemitz
The University of North Carolina at Chapel Hill, USA
nemitz@cs.unc.edu

Tanya Amert
The University of North Carolina at Chapel Hill, USA
tamert@cs.unc.edu

James H. Anderson
The University of North Carolina at Chapel Hill, USA
anderson@cs.unc.edu

Abstract

During the past decade, parallelism-related issues have been at the forefront of real-time systems research due to the advent of multicore technologies. In the coming years, such issues will loom ever larger due to increasing core counts. Having more cores means a greater potential exists for platform capacity loss when the available parallelism cannot be fully exploited. In this work, such capacity loss is considered in the context of real-time locking protocols. In this context, lock nesting becomes a key concern as it can result in transitive blocking chains that force tasks to execute sequentially unnecessarily. Such chains can be quite long on a larger machine. Contention-sensitive real-time locking protocols have been proposed as a means of "breaking" transitive blocking chains, but such protocols tend to have high overhead due to more complicated lock/unlock logic. To ease such overhead, the usage of lock servers is considered herein. In particular, four specific lock-server paradigms are proposed and many nuances concerning their deployment are explored. Experiments are presented that show that, by executing cache hot, lock servers can enable reductions in lock/unlock overhead of up to 86%. Such reductions make contention-sensitive protocols a viable approach in practice. This artifact contains the implementation of two contention-sensitive locking protocol variants implemented with four proposed lock-server paradigms, as well as the experiments with which they were evaluated.

2012 ACM Subject Classification Computer systems organization → Real-time systems, Computer systems organization → Embedded and cyber-physical systems, Software and its engineering → Mutual exclusion, Software and its engineering → Real-time systems software, Software and its engineering → Synchronization, Software and its engineering → Process synchronization

Keywords and phrases multiprocess locking protocols, nested locks, priority-inversion blocking, reader-/writer locks, real-time locking protocols

Digital Object Identifier 10.4230/DARTS.4.2.2

Related Article Catherine E. Nemitz, Tanya Amert, and James H. Anderson, "Using Lock Servers to Scale Real-Time Locking Protocols: Chasing Ever-Increasing Core Counts", in Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS 2018), LIPIcs, Vol. 106, pp. 25:1–25:23, 2018.
http://dx.doi.org/10.4230/LIPIcs.ECRTS.2018.25

Related Conference 30th Euromicro Conference on Real-Time Systems (ECRTS 2018), July 3–6, 2018, Barcelona, Spain

* Work supported by NSF grants CNS 1409175, CPS 1446631, CNS 1563845, and CNS 1717589, ARO grant W911NF-17-1-0294, and funding from General Motors. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGS-1650116. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Scope

This artifact was used in the evaluation portion of the conference paper [2] to generate figures and form the numbered observations.

More specifically, the artifact showed a significant difference in blocking and overhead between MCS locks [1] and the C-RNLP (Contention-sensitive Real-Time Nested Locking Protocol) variants. Employing a lock server or multiple lock servers significantly lowers overhead on our platform. The artifact also explores the performance of each lock server variant with different numbers of cores and sockets in use.

2 Content

The artifact package includes:

- Readme guide: `readme`
- Makefile: `Makefile`
- Source code: `src/`
- Include files: `include/`
- Scripts: `scripts/`
- Example plots: `plots/artifact_evaluation_examples/`
- Paper: `paper.pdf`

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://cs.unc.edu/~anderson/papers/ecrts18b_code.tgz`.

4 Tested platforms

The artifact was originally tested on a dual-socket, 18-cores-per-socket Intel Xeon E5-2699. Each core of this platform has a 32KB L1 data cache and a 32KB L1 instruction cache. Pairs of cores share a unified 256KB L2 cache, and all cores on a socket share a unified 45MB L3 cache. The experiments conducted on this platform led to the numbered observations in the paper [2].

The artifact was additionally tested on a four-socket, 6-cores-per-socket Intel Xeon L7455. On this machine, each core has a 32KB L1 data cache and a 32KB L1 instruction cache. Additionally, there is a 3MB L2 cache, and all cores on a socket share a 12MB L3 cache. The results of this evaluation matched the overarching trends originally observed [3].

In general, to see similar trends as those presented in the paper, a multicore machine with multiple sockets is required. Given such a platform, similar trends are expected, though exact measurements will vary.

5 License

6 MD5 sum of the artifact

27f58adbc5448551931207e0eedbc0aa

7 Size of the artifact

1.45 MB

Acknowledgements. The authors thank the reviewers for their evaluation of and helpful feedback about this artifact.

References

1 J. Mellor-Crummey and M. Scott. Algorithms for scalable synchronization of shared-memory multiprocessors. *Transactions on Computer Systems*, 9(1), 1991.

2 C. Nemitz, T. Amert, and J. Anderson. Using lock servers to scale real-time locking protocols: Chasing ever-increasing core counts. In *ECRTS 2108*.

3 C. Nemitz, T. Amert, and J. Anderson. Using lock servers to scale real-time locking protocols: Chasing ever-increasing core counts (extended version), 2018. URL: `http://www.cs.unc.edu/~anderson/papers.html`.