Brief Announcement: Fast and Scalable Group Mutual Exclusion

Shreyas Gokhale
The University of Texas at Dallas
Richardson, TX 75080, USA
shreyas.gokhale@utdallas.edu
https://orcid.org/0000-0002-7589-6927

Neeraj Mittal
The University of Texas at Dallas
Richardson, TX 75080, USA
neerajm@utdallas.edu
https://orcid.org/0000-0002-8734-1400

Abstract

The *group mutual exclusion (GME)* problem is a generalization of the classical mutual exclusion problem in which every critical section is associated with a *type* or *session*. Critical sections belonging to the same session can execute concurrently, whereas critical sections belonging to different sessions must be executed serially. The well-known read-write mutual exclusion problem is a special case of the group mutual exclusion problem.

In a shared memory system, locks based on traditional mutual exclusion or its variants are commonly used to manage contention among processes. In concurrent algorithms based on *fine-grained* synchronization, a single lock is used to protect access to a *small* number of shared objects (e.g., a lock for every tree node) so as to minimize contention window. Evidently, a large number of shared objects in the system would translate into a large number of locks. Also, when fine-grained synchronization is used, most lock accesses are expected to be uncontended in practice.

Most existing algorithms for the solving the GME problem have high space-complexity per lock. Further, all algorithms except for one have high step-complexity in the uncontented case. This makes them unsuitable for use in concurrent algorithms based on fine-grained synchronization. In this work, we present a novel GME algorithm for an asynchronous shared-memory system that has $O(1)$ space-complexity per GME lock when the system contains a large number of GME locks *as well as* $O(1)$ step-complexity when the system contains no conflicting requests.

2012 ACM Subject Classification Theory of computation → Concurrent algorithms

Keywords and phrases Group Mutual Exclusion, Fine-Grained Synchronization, Space Complexity, Contention-Free Step Complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.49

Funding This work was supported, in part, by the National Science Foundation (NSF) under grants numbered CNS-1115733 and CNS-1619197.

1 Introduction

The *group mutual exclusion (GME)* problem is a generalization of the classical mutual exclusion (ME) problem in which every critical section is associated with a *type* or *session* [7]. Critical sections belonging to the same session can execute concurrently, whereas critical sections belonging to different sessions must be executed serially. The GME problem models situations in which a resource may be accessed at the same time by processes of the same group, but not by processes of different groups. As an example, suppose data is stored on multiple discs in a shared CD-jukebox. When a disc is loaded into the player, users that need data on that disc can access the disc concurrently, whereas users that need data on a different disc have to wait until the current disc is unloaded [7]. The well-known readers/writers problem is a special case of the group mutual exclusion problem.

In a shared memory system, locks based on traditional mutual exclusion or its variants are commonly used to manage contention among processes. A lock grants a process exclusive access to a shared object, preventing any other process from modifying the object. This makes it easier to design, analyze, implement and debug lock-based concurrent algorithms.

With the wide availability and use of multicore systems, developing concurrent data structures that scale well with the number of cores has gained increasing importance. Many of the best performing concurrent algorithms for data structures use fine-grained synchronization in which a single lock is used to protect access to a small number of shared objects so as to minimize contention window [5]. In order to perform an operation, a process typically needs to lock only a small number of objects, thereby allowing multiple processes whose operations do not conflict with each other to manipulate parts of the data structure at the same time. Typically, in such a system, most lock accesses are expected to be uncontended in practice [5]. Recently, GME-based locks have been used to improve the performance of a concurrent skip list using the notion of unrolling in which multiple key-value pairs are stored in a single node [8].

All existing algorithms for solving the GME problem have either high space-complexity of $\Omega(n)$ per lock or high step-complexity of $\Omega(n)$ in the uncontented case or both, where n denotes the number of processes in the system [7, 6, 2, 1, 4]. This makes them unsuitable for use in lock-based concurrent data structures that employ fine-grained synchronization to manage contention. In this work, we present a novel GME algorithm for an asynchronous shared-memory system that has $O(1)$ space-complexity per GME lock when the system contains a large number of GME locks *as well as* $O(1)$ step-complexity when the system contains no conflicting requests.

2 The Group Mutual Exclusion Algorithm

Our GME algorithm is inspired by Herlihy's universal construction for deriving a wait-free linearizable implementation of a concurrent object from its sequential specification using consensus objects [5]. Roughly speaking, the universal construction works as follows. The state of the concurrent object is represented using (i) its initial state and (ii) the sequence of operations that have applied to the object so far. The two are maintained using a singly linked list in which the first node represents the initial state and the remaining nodes represent the operations. To perform an operation, a process first creates a new node and initializes it with all the relevant details of the operation (type, input arguments, etc.). It then tries to append the node at the end of the list. To manage conflicts in case multiple processes are trying to append their own node to the list, a consensus object is used to determine which of the several nodes is chosen to be appended to the list. Specifically, every node stores a consensus object and the consensus object of the current last node is used to decide its successor (i.e., the next operation to be applied to the object). A process whose node is not selected simply tries again. A helping mechanism is used to guarantee that every process trying to perform an operation eventually succeeds in appending its node to the list.

We modify the aforementioned universal construction to derive a GME algorithm that satisfies several desirable properties. Intuitively, an operation in the universal construction corresponds to a critical section request in our GME algorithm. Appending a new node to the list thus corresponds to establishing a new session. However, unlike in the universal construction, a single session in our GME algorithm can be used to satisfy multiple critical section requests. This basically means that every critical section request does not cause a new node to be appended to the list. This requires some careful bookkeeping so that no "empty" sessions are established. Further, a simple consensus algorithm, implemented using LL/SC instructions, is used to determine the next session to be established. Helping is used to ensure that every request is eventually satisfied.

If the node for a critical section request is appended to the list, then the process that owns the node is said to enter the session as a leader; otherwise it is said to enter the session as a follower. A leader process, on leaving its critical section, relinquishes the ownership of its current node and claims the ownership of the node for the previous session instead. This enables us to bound the length of the linked list and make our algorithm space-efficient.

More details of our GME algorithm are available in [3], where we also describe a way to bound the values of all the variables used in our GME algorithm.

▶ **Theorem 1** (multi-lock space complexity)**.** *The space complexity of our GME algorithm is $O(m + n^2 + n\ell)$ space, where n denotes the number of processes, m denotes the number of GME objects and ℓ denotes the maximum number of locks a process needs to hold at the same time.*

▶ **Theorem 2** (concurrent entering step complexity)**.** *The maximum number of steps a process has to execute in its entry and exit sections provided no other process in the system has an outstanding* conflicting *request during that period is $O(1)$.*

Note that the above theorem also implies $O(1)$ step complexity in contention-free case.

References

1 V. Bhatt and C. C. Huang. Group Mutual Exclusion in $O(\log n)$ RMR. In *Proceedings of the 29th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 45–54, JUL 2010.

2 R. Danek and V. Hadzilacos. Local-Spin Group Mutual Exclusion Algorithms. In *Proceedings of the 18th Symposium on Distributed Computing (DISC)*, pages 71–85, OCT 2004.

3 S. Gokhale and N. Mittal. Fast and Scalable Group Mutual Exclusion. Available at http://arxiv.org/abs/1805.04819.

4 Y. He, K. Gopalakrishnan, and E. Gafni. Group Mutual Exclusion in Linear Time and Space. In *Proceedings of the 17th International Conference on Distributed Computing And Networking (ICDCN)*, JAN 2016.

5 M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Morgan Kaufmann, 2012.

6 P. Jayanti, S. Petrovic, and K. Tan. Fair Group Mutual Exclusion. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 275–284, JUL 2003.

7 Y.-J. Joung. Asynchronous Group Mutual Exclusion. *Distributed Computing (DC)*, 13(4):189–206, 2000.

8 K. Platz. *Saturation in Lock-Based Concurrent Data Structures*. PhD thesis, Department of Computer Science, The University of Texas at Dallas, 2017.