Lock-free Concurrent Reading While Writing

Принципиальным элементом большинства lock-free алгоритмов является допущение, что элемент данных может быть атомарно прочитан или изменён. На практике это приводит к тому, что большинство реализаций для нетривиальных данных используют размещение в отдельных блоках памяти (куча, Java-объекты) и манипулируют указателями. Такой подход вносит дополнительные расходы в виде дополнительных косвенных обращений к памяти и управления кучей, снижает эффективность использования кэшей.

Представляет интерес разработка практически эффективных алгоритмов атомарного обновления многословных объектов с учётом применения как элементов различных lock-free структур данных (Hash-таблицы, очереди).

Отдельное потенциально инетерсное направление - решение задачи на основе HTM-механизмов и оценка эффективности с учётом различных профилей нагрузки.

Описание

Работы в области CRWW ведутся достаточно давно - первые статьи датированы 80-ми годами. Есть общие алгоритмы, которые обеспечивают lock-free mw/mr crww. На практике они довольно громоздкие и требуют значительных накладных расходов как по памяти, так и по скорости (**? ну жен пруфлинк**, на пальцах - это сильно избыточно, требует матрицы ячеек).

В нашей конкретной задаче писатель всегда один, и достоверно известно, что пишет не с такой частотой, чтобы читатель постоянно натыкался на нечетные версии.

Для одной ячейки есть алгоритм seqlock(stampedlock), базирующийся на версионировании - writer, начав писать, увеличивает версию на 1; закончив - еще на 1. Читатель, увидев нечетную версию понимает, что писатель в процессе и уходит в цикл. К сожалению, он не lock-free. Если writer встанет, читатель зациклится.

Задача: защитить multi-word регистр seqlock-ом и сделать алгоритм lock-free.

Предлагаемое решение:

- каждому писателю выделим отдельный буфер, в нем писатель готовит значение;
- читатель, видя нечетную версию, идет в этот буфер;
- если и в буфере нечетная, писатель гарантировано продвинулся.

Задача-минимум: реализовать, формально доказать в терминах состояний.

Дальнейшие задачи:

- работать над доказательством совместно с человеком, который будет доказывать CHM (ConcurrentHashMap), потому что впоследствии мы из док-ва CHM извлечем требования к MW-регистру
- доказательство на языке состояний требует общего порядка на операциях перехода между состояниями (**есть работы со сложными доказательствами в терминах состояний ?**). С теоретической точки зрения интересно получить доказательство на языке инвариантов и в терминах модели памяти, где набор возможных fences шире, чем полный барьер памяти.

Идеальный результат:

- реализован lock-free sw/rw mw-crww алгоритм
- приведено строгое доказательство в терминах состояний
- англоязычная публикация алгоритма и доказательства
- построено доказательство в тех же терминах, что и доказательство алгоритма chm - предположительно, в терминах инвариантов
- выяснены необходимые для использования в chm требования к mw-регистру

Related papers:

1. **Concurrent Reading and Writing - original Lamport's paper (1977), only read/write registers assumed as primitives**
2. Concurrent Reading and Writing of Clocks - interesting paper by Lamport (1990) about the special case of monotonic clocks that provide for much simpler solutions
3. Concurrent Reading While Writing - Original Peterson's algorithm (1983), proves n+2 buffers lower bound for wait-free algo
4. Concurrent reading while writing II: The multi-writer case - Peterson's extension to mutli-writer case (1987), only IEEE Xplore ...
   PDF REQUIRED
5. A Fully Asynchronous Reader/Writer Mechanism for Multiprocessor Real-Time Systems (1997) - Single Writer, Multiple Readers, using n+2 buffers