УДК 004.021

ОПТИМАЛЬНЫЕ МНОГОПОТОЧНЫЕ СТРУКТУРЫ ДАННЫХ: ПОСТРОЕНИЕ АДАПТИВНЫХ РЕАЛИЗАЦИЙ МНОЖЕСТВ, МАССИВОВ И СПИСКОВ

Цуциев А.С. (Университет ИТМО)
Научный руководитель – PhD, доцент Аксенов В.Е.
(Университет ИТМО)

Создание многопоточной структуры данных сопряжено с рядом проблем, одной из которых является использование блокировок в отдельно взятой реализации, а именно их число и расположение. С одной стороны, большое число блокировок позволяет уменьшить число ожиданий взятия блокировки при выполнении операций, с другой стороны, так как операции с блокировками являются сами по себе затратными по времени, время исполнения существенно увеличивается. Представляемая работа описывает попытку установления экспериментальным путем зависимости между количеством и расположением блокировок в различных реализациях многопоточных структур данных и их производительностью. Серия экспериментов с различными реализациями динамических массивов и множеств привела автора исследования к созданию адаптивной структуры данных, которая может быть использована как основа для реализации многопоточных хэш-таблиц и множеств.

Введение. Первоначально, нашей целью являлось получение практических результатов, позволяющих расширить концепцию concurrency-optimal (Optimal Concurrency for List-Based Sets – Vitaly Aksenov, Vincent Gramoli, Petr Kuznetsov, Di Shang, Srivatsan Ravi), а именно: способность структуры данных «принимать» все корректные многопоточные «расписания», то есть корректно работать при всех корректных сценариях многопоточного исполнения.
Далее, помимо возможности корректной работы, мы обратились к вопросу пропускной способности структуры данных, а именно, к количеству исполняемых операций за единицу времени. По итогам сравнения различных версий динамических массивов и множеств: без блокировок, lock-free, с различными конфигурациями блокировок как по типу, так и количеству, была разработана адаптивная многопоточная структура данных на основе консистентного хэширования, опережающая по скорости работы все рассмотренные неадаптивные версии.

Основная часть. В рамках экспериментов нами были рассмотрены динамически расширяемые массивы и множества со следующими операциями:

1) Массивы:
    a. Чтение: получение элемента по индексу, поиск элемента по значению, запрос размера структуры;
    b. Запись: добавление в конец, добавление на произвольную позицию, удаление по значению, удаление по позиции, удаление произвольного набора элементов, очистка всей структуры;
2) Множества:
    a. Чтение: поиск элемента по значению, запрос размера структуры;
    b. Запись: добавление, удаление по значению, удаление произвольного набора элементов, очистка всей структуры;

Для различных вариантов их реализации проводились замеры пропускной способности, оцениваемой как число успешно выполненных операций за единицу времени для различного числа потоков, максимального числа элементов, а также отношения числа операций чтения к количеству операций записи. Каждый замер представлял из себя 5 независимых запусков тестирующей программы, предварительное заполнение структуры элементами не производилось. Код программы написан на языке Java с использованием только стандартной библиотеки.

Первым рассмотренным вариантом реализации, как и в упомянутой во введении работе, стала последовательная, не предназначенная по своей сути для исполнения в многопоточной среде из-за того, что она не удовлетворяет гарантиями корректности, например, ввиду отсутствия линеаризации исполняемого кода. Модель показала сравнительно хорошую производительность, растущую с увеличением числа потоков.

Второй была изучена lock-free реализация, показавшая худшие результаты без ярко выраженной динамики и с наибольшей долей неуспешных операций среди рассмотренных моделей. Такие результаты обусловлены дополнительными ресурсозатратными усложнениями модели, необходимыми для ее корректной работы без применения блокировок.

После этого были проанализированы две версии с блокировками: обычной и read-write. Производительность обеих версий оказалась ниже последовательной, но существенно выше lock-free и падала с ростом числа потоков, однако, скорость падения также уменьшилась – функция практически не убывала на тестах с большими значениями числа потоков.

Для улучшения результатов было принято решение оптимизировать работу структуры данных с учетом специфики конкретной задачи ее применения. С этой целью был рассмотрен более узкий набор операций: для массива были выбраны только поиск по значению, добавление в конец, получение размера и удаление по значению. Такой подход дает возможность скорректировать стратегию работы структуры: при определенных сценариях исполнения отпадает необходимость взятия блокировок без потери линеаризации, что заметно увеличивает производительность модели по сравнению с версией без таких модификаций.
Далее мы решили использовать блокировки не на всю структуру данных, а на часть элементов (с учетом описанных ранее стратегий): объект разбивается на блоки, каждый из которых имеет собственную read-write блокировку. Такая модификация позволяет получить большой выигрыш в производительности по сравнению с предыдущими версиями, однако, результаты в значительной степени зависят от подбора параметров структуры: числа и размера блоков.

Одной из проблем описанных выше моделей является сравнительно высокое время работы операций поиска и удаления за счет их линейной асимптотики. Для достижения лучшей производительности было принято решение использовать хэш-функцию, которая вместе с делением данных на блоки, позволила бы довести асимптотику до логарифмической.
Структура данных была основана на принципе консистентного хэширования, используемого для шардирования в распределенных системах, и представляет из себя массив юнитов, каждый из которых определяется значением хэша. В юните хранятся элементы, значение хэш-функции для которых попадают в отрезок, соответствующий данному юниту. Каждая операция с элементом начинается с вычисления его хэша и определения его юнита с помощью бинарного поиска, что и обеспечивает время работы, равное логарифму длины массива юнитов плюс размер юнита. Такой подход позволил существенно улучшить производительность: лучшие результаты – в 3 раза больше, чем у последовательной реализации – были получены при среднем размере юнитов по 100-300 элементов.

Однако, главной проблемой предыдущего подхода остаётся распределение значений хэш-функции элементов, что вынуждает использовать либо большое число юнитов маленького размера, что приводит к неиспользованию большинства из них, либо к излишней нагрузке только на некоторые.
Для решения этой проблемы нами был использован подход, схожий с описанным в бакалаврской работе «Разработка конкурентных структур данных, дружелюбных к памяти» Смирнова Р.А., где блоки, достигающие размера большего, чем допустимый, разделяются пополам: на элементы, большие добавляемого и меньшие.

В нашем случае, к модели была добавлена возможность разбивать слишком нагруженные и «склеивать» недостаточно нагруженные юниты, причем сравнение в процессе разбиения проводится не для самих элементов, а для значений их хэшей, что позволяет избежать обращения к компараторам пользовательских типов, которые могут работать долго. Данные корректировки производятся автоматически, с учетом упомянутых выше оптимальных размерах юнитов (100-300 элементов). Такие значения позволяют сбалансировать как нагрузку на сами юниты, так и не допустить превалирование операций по изменению их конфигурации над операциями с самой структурой.
Вышеописанная модель может быть использована как основа для построения адаптивных многопоточных реализаций множеств, мультимножеств, а также хэш-таблиц.
Аналогичный подход позволяет построить адаптивную версию многопоточного списка, где блоки, на которые он разделяется, изменяют свой размер по аналогичной стратегии в соответствии с изменением нагрузки.

Выводы. После проведения вышеперечисленных экспериментов нами были сделаны следующие выводы:

1) Использование блокировок эффективнее, чем lock-free подход;
2) Применение read-write блокировок предпочтительно (с учетом специфики их реализации в используемой библиотеке);
3) Блокировки необходимо по возможности применять к блокам элементов, а не ко всей структуре сразу;
4) Слишком малый размер блоков, а, как следствие, слишком большое количество блокировок, ведет к падению производительности;
5) При разработке многопоточной структуры данных необходимо учитывать специфику задачи (набор используемых операций, потенциальную нагрузку и т. п.), в соответствии с которой модифицировать универсальную модель;
6) Адаптивный подход при разбиении данных на блоки позволяет достичь существенного увеличения производительности.

Цуциев А.С. (автор)

Аксенов В.Е. (научный руководитель)