УДК 004.021

РАЗРАБОТКА И РЕАЛИЗАЦИЯ КОНКУРЕНТНОГО ДЕРЕВА ПОИСКА С ПОДДЕРЖКОЙ ЗАПРОСОВ НА ОТРЕЗКЕ

Кокорин И.В. (Университет ИТМО)
Научный руководитель – PhD Аксёнов В.Е.
(Университет ИТМО)

В этой работе исследуется возможность разработки конкурентного lock-free дерева поиска, предоставляющего возможность исполнять не только запросы по ключу (вставка ключа, удаление ключа и проверка ключа на существование), но и запросы на отрезке (или массовые запросы). В работе исследуется возможность эффективного конкурентного исполнения таких запросов на отрезке, для которых существуют последовательные алгоритмы, вычисляющие их за время, пропорциональное высоте дерева поиска.

Введение. Иерархические структуры данных позволяют нам эффективно не только исполнять запросы над отдельными ключами (примерами таких запросов являются вставка ключа, удаление ключа и поиск ключа), но и запросы на отрезке. Примером такого запроса яаляется Count(MinKey; MaxKey) — поиск количества ключей, лежащих в диапазоне [MinKey; MaxKey]. Используя иерархические структуры данных (например, бинарные деревья поиска) такой запрос можно исполнить не за время O(n), где n это число ключей в структуре данных, а за время O(h), где h — высота дерева поиска. Следовательно, используя сбалансированые бинарные деревья поиска (например, AVL-дерево), можно исполнять такие запросы за время O(log n).

Несмотря на то, что вопрос последовательного исполнения подобных запросов хорошо изучен, в настоящий момент не существует эффективных алгоритмов конкурентного исполнения таких запросов. Все существующие на данный момент алгоритмы конкурентного исполнения запросов на отрезке (например, описанные в [2, 3, 4]) работают за время O(n). В этой работе мы предлагаем универсальный метод, который может быть использован для эффективного исполнения произвольных массовых запросов в иерархических структурах данных и применяем разработанный алгоритм для эффективной реализации запроса Count в двоичном дереве поиска.

Основная часть. Основная сложность эффективного исполнения массовых запросов заключается в необходимости сохранять помимо самой структуры дерева некую вспомогательную информацию (например, для эффективного выполнения запроса Count(MinKey; MaxKey) необходимо для каждой вершины хранить число ключей в её поддереве). Так как у нас конкурентно исполняется сразу несколько запросов, это может првести к тому, что структура дерева и вспомогательная информация меняется различными потоками несогласованно, что приводит к тому, что некоторые запросы начинают исполняться некорректно (то есть к нелинеаризуемым исполнениям).

Будем рассматривать для начала несбалансированные деревья поиска. Балансировка дерева может выполняться путём перестройки поддеревьев, аналогично подходу описанному в [1]. Кроме того, будем рассматривать деревья, которые хранят ключи в листовых вершинах, а во внутренних вершинах хранят только служебную информацию.

Для того, чтобы корректно исполнять конкурентные запросы, необходимо их упорядочить. Для этого будем в каждой вершине V хранить очередь дескрипторов операций, которые должны быть исполнены в поддереве вершины V. Будем считать, что дескриптор операции Op содержит всю информацию, необходимую для исполнения этой операции.

Каждый поток T, который хочет исполнить над структурой анных операцию Op, сначала вставляет дескриптор этой операции в очередь корневого узла. После чего поток T (в дальнейшем мы будем называть его потоком-инициатором) спускается по дереве от корня к листьям, в каждой вершине V, которую он посещает, производя следующие операции:

1) В очереди вершины V помочь исполнить все операции, дескрипторы которых расположены ближе к голове очереди, чем дескриптор операции Op.

2) Достать из очереди дескриптор операции Op

3) Изменить необходимым образом вспомогательную информацию, хранимую в вершине (например, при вставке несуществующего ключа в поддерево вершины V необходимо увеличить на единицу число ключей в поддереве вершины V, а при удалении существующего ключа из поддерева вершины V — уменьшить это число на единицу).

4) Добавить дескриптор операции Op в одного или нескольких детей вершины V и продолжить исполнение операции Op в этих детях (если это необходимо).

Использование очередей для упорядочивания запросов гарантирует следующий инвариант: пусть существуют два запроса R и Q, при этом оба этих запроса должны быть исполнены в поддереве вершины V. Тогда R и Q будут исполнены в этом поддереве в том же порядке, в котором они были добавлены в очередь коневой вершины (так как в каждой вершине мы исполняем запросы в порядке, заданном очередью данной вершины). Следовательно, все запросы исполняются в порядке, в котором они были вставлены в очередь корневой вершины. С другой стороны, если запросы R и Q должны исполяться в разных поддеревьях, то они могут быть исполнены в произвольном порядке, в том числе и параллельно (так как они не мешают друг другу). Именно это и позволяет нам исполнять запросы параллельно.

Заметим, что это позволяет легко проверять структуру данных на линеаризуемость: действительно, мы знаем, в каком порядке запросы были добавлены в очередь корневой вершины. Именно в таком порядке запросы должны были исполняться. Следовательно, мы получили последовательное исполнение, которому должно быть эквивалентно наше конкурентное исполнение. Сверив результаты конкурентного исполнения с результатами параллельного, мы легко проверяем параллельное исполнение на линеаризуемость, причём эту проверку мы может сделать за полиномиальное время.

Кроме того, некоторые запросы в структуре данных могуть быть реализованы с wait-free гарантией прогресса. В нашем примере запрос, проверяющий существование ключа в структуре может быть реализован именно так, так как он не читает и не изменяет вспомогательную информацию в дереве (число ключей в различных поддеревьях). В этой работе нами был предложен алгоритм исполнения таких запросов с wait-free гарантией прогресса.

Так как запросы на вставку и удаение ключей могут разбалансировать дерево, мы используем механизм перестройки поддеревьев для сохранения сбалансированности. А именно: если в поддереве вершины V было исполнено слишком много запросов на изменение, мы полностью перестраиваем это поддерево в идеально сбалансированное двоичное дерево поиска.

Выводы. Разработанный алгоритм исполнения запроса через спуск дескриптора по очередям позволяет применять для конкурентного lock-free исполнения массовых запросов те же алгоритмы, которыми мы пользуемся для эффективного последовательного исполнения. Разработанный алгоритм был применён для эффективной реализации запроса Count(MinKey; MaxKey), что позволило реализовать этот запрос с lock-free гарантиями прогресса и за время O(log n) на один запрос.

Тот факт, что запросы исполняются в том порядке, в котором они были добавлены в корневую очередь, позволил нам протестировать реализованую структуру данных на линеаризуемость за полиномиальное время.

Кроме того, разработанная структура данных позволяет реализовать некоторые запросы с wait-free гарантией прогресса. В работе с такой гарантией прогресса была реализована проверка существования ключа в дереве.

Список литературы

1. Prokopec A., Brown T., Alistarh D. Analysis and Evaluation of Non-Blocking Interpolation Search Trees //arXiv preprint arXiv:2001.00413. – 2020
2. Avni H., Shavit N., Suissa A. Leaplist: lessons learned in designing tm-supported range queries //Proceedings of the 2013 ACM symposium on Principles of distributed computing. – 2013 – С. 299-308.
3. Arbel-Raviv M., Brown T. Harnessing epoch-based reclamation for efficient range queries //ACM SIGPLAN Notices. – 2018 – Т. 53 – №. 1 – С. 14-27.
4. Sagonas K., Winblad K. Efficient support for range queries and range updates using contention adapting search trees //Languages and Compilers for Parallel Computing. – Springer, Cham, 2015 – С. 37-53.

Кокорин И.В. (автор) Подпись

Аксёнов В.Е. (научный руководитель) Подпись