Стратегии выполнения операции соединения в XQuery

© Борис Хвостиченко

Санкт-Петербургский Государственный Университет
boriskhv@acm.org

Аннотация

В работе рассматриваются альтернативные стратегии выполнения соединения в XQuery-запросах. Предложен алгоритм выполнения соединения в основной памяти и приведены теоретическое и экспериментальное его сравнения с алгоритмом вложенных циклов.

1 Введение

Формат данных XML уже стал достаточно распространен в различных приложениях. XML описывает структуру данных самими данными, что позволяет не придерживаться заданной заранее жесткой структуры. Для выполнения операций над такими данными требуется специальный язык запросов, таким языком стал XQuery [6].

Так как объемы данных, представленных в XML формате быстро растут, то актуальным стал вопрос быстродействия XQuery-запросов. Эта работа рассматривает выполнение операции соединения XML-элементов, принимая во внимание упорядоченность элементов в XML.

В работе принимаются следующие предположения: в ходе выполнения запроса операции производятся над (возможно упорядоченным) множеством XML-элементов; сложность выполнения запроса – это количество элементарных операций сравнения над элементами в ходе выполнения запроса.

2 Другие подходы

Вопрос выполнения операции соединения в XQuery не подвергался особо детальному исследованию. В работе [2] изложены принципы выбора последовательности выполнения множественных операций соединения, но не предложены никакие оценки стоимости выполнения. Работа [3] посвящена выбору оптимального порядка выполнения соединения при условии сохранения порядка каждым соединением.

Индустриальные реляционные системы разбивают XML-документы по отношениям, специально хранят порядок элемента в документе [4, 1] и заботятся о его восстановлении в ходе выполнения запроса. При этом используется уже существующий оптимизатор, который добивается оптимальности выполнения запроса.

3 Выполнение упорядоченного соединения в основной памяти

Требование языка XQuery [6] – сохранение естественного порядка, присущего XML-данным. Порядок элементов в получившемся списке определяется порядком элементов в более внешнем множестве. Поэтому операция соединения в общем случае не является коммутативной, т.е. $a \otimes b \neq b \otimes a$. В общем случае соединение выполняется методом вложенных циклов (Nested Loop Join), который гарантирует корректный порядок, но имеет стоимость, пропорциональную произведению размеров внешнего и внутреннего множеств.

Для снятия ограничения некоммутативности операции соединения можно искусственно запоминать, а потом восстанавливать порядок элементов. Иначе говоря, Op = RestoreOrder ∘ Op ∘ SaveOrder, где $Op \neq SaveOrder$ или $RestoreOrder$. Можно считать, что порядок элементов уже хранится, так как это требование модели данных XQuery и XPath [5], поэтому операция *SaveOrder* не требуется.

При сортировке результата или явной разупорядоченности запроса с помощью *Unorder*, восстановление порядка не требуется. Операцию восстановления порядка нужно выполнять только один раз – в конце выполнения запроса.

Исходя из вышесказанного, можно предложить несколько правил эквивалентности:

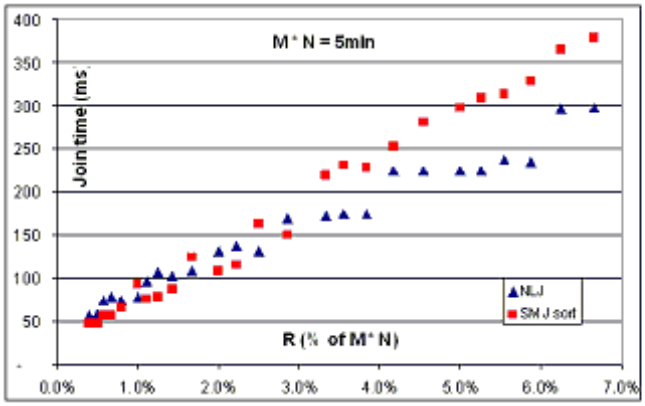
1. $RestoreOrder \circ Op = Op \circ RestoreOrder$
2. $Sort = Sort \circ Unorder$
3. $Unorder \circ RestoreOrder = Unorder$
4. $Unorder \circ Op = Op \circ Unorder$

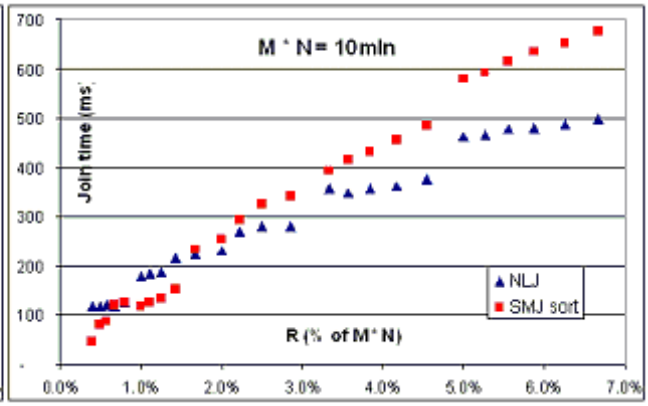
Правило 1 описано выше. Второе правило говорит о том, что перед сортировкой можно разупорядочить множество. Третье убирает восстановление порядка, если запрос явно разупорядочен. Четвертое перемещает разупорядочивание как можно ниже по дереву выполнения запроса.

3.1 Оценка сложности алгоритма

Предположим, что количество элементов во внешнем множестве равно M, во внутреннем – N.

Труды 7ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2005, Ярославль, Россия, 2005.





Тогда сложность метода вложенных циклов равна Cost(NLJ) = *M*N*.

При применении соединения Sort-Merge Join (SMJ) с восстановлением порядка, стоимость складывается из начальной сортировки, стоимости соединения и стоимости восстановления порядка (т.е. сортировки результата по порядку элементов). Можно считать, что сложность сортировки множества величиной *M* равна $M \log_2 M$. Соответственно, общая стоимость всей операции равна:

$Cost(SMJ) = M \log_2 M + N \log_2 N + M + N + R \log_2 R$,

где *R* – количество элементов в результате.

Так как *R* может принимать любое значение в диапазоне от 0 до *M*N*, то в общем случае SMJ проигрывает по производительности NLJ. Однако если размер результата невелик относительно *M*N*, то SMJ выполняется быстрее.

3.3 Экспериментальные результаты

Экспериментально были протестированы алгоритмы вложенных циклов и соединения слиянием с последующей сортировкой. Тесты проводились на множествах размером *M=1000*, *N*∈{1000, 5000, 10000}. Множества случайным образом наполнялись равномерно распределенными ключами-натуральными числами. Каждый тест был проведен пять раз, полученные результаты усреднялись. Эксперименты показали, что при *R*<0.015**M*N*, SMJ работает быстрее, чем NLJ (зависимость времени выполнения соединения от размера результата приведена на Рис. 1). Это проявляется при достаточно больших размерах *M*N*, при *M*N<1mln* измерения выходят за границы точности. Эмпирически полученная константа 0.015 зависит от конкретной машины и реализации алгоритмов: те же самые тесты на другом компьютере показали аналогичное поведение, но численно граница изменилась.

При соединении нескольких множеств, сложность алгоритма вложенных циклов прямо пропорциональна произведению размеров всех множеств, поэтому преимущество будет на стороне SMJ. Получающийся порядок можно использовать так, как это делают реляционные оптимизаторы [7].

4 Заключение и будущая работа

В работе предложен способ выполнения соединения упорядоченных множеств в XML. Показано преимущество предложенного метода при выполнении алгоритма в основной памяти и малых размерах результата. В дальнейшем работа будет расширена оценкой различных стратегий соединения на диске.

Литература

[1] Z. H. Liu, M. Krishnaprasad, and V. Arora. Native XQuery processing in Oracle XMLDB. In *SIGMOD'05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 828-833, New York, NY, USA, 2005. ACM Press.

[2] N. May, S. Helmer, C. Kanne, and G. Moerkotte. XQuery Processing in Natix with an Emphasis on Join Ordering. *First International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P 2004)*, June 2004.

[3] G. Moerkotte. Constructing optimal bushy trees possibly containing cross products for order preserving joins is in P. Technical report, University of Mannheim, August 2003.

[4] M. Nicola and B. van der Linden. Native XML support in DB2 universal database. In *VLDB'05: Proceedings of the 31st international conference on Very large data bases*, pages 1164-1174. VLDB Endowment, 2005.

[5] XQuery 1.0 and XPath 2.0 Data Model, 2003. http://www.w3.org/TR/xpath-datamodel/.

[6] XQuery 1.0: an XML Query Language, 2005. http://www.w3.org/TR/xquery/.

[7] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD '79: Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23-34, New York, NY, USA, 1979. ACM Press.

Join execution strategies in XQuery

Boris Khvostichenko

This paper considers alternative join execution strategy in XQuery. Main memory join algorithm is presented for joining ordered sets of XML elements, together with theoretical and experimental comparison versus Nested Loop Join algorithm.