Data & Knowledge Engineering 69 (2010) 29–49

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Transaction reordering

Gang Luo [a,*], Jeffrey F. Naughton [b], Curt J. Ellmann [b], Michael W. Watzke [c]

[a] *IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA*
[b] *University of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53706, USA*
[c] *Teradata, 5752 Tokay Blvd Suite 400, Madison, WI 53719, USA*

A R T I C L E   I N F O

Article history:
Available online 31 August 2009

Keywords:
Data warehousing
Workload management
Transaction reordering
Continuous data loading
Synchronized scans

A B S T R A C T

Traditional workload management methods mainly focus on the current system status while information about the interaction between queued and running transactions is largely ignored. This paper proposes using transaction reordering, a workload management method that considers both the current system status and information about the interaction between queued and running transactions, to improve the transaction throughput in an RDBMS. Our main idea is to reorder the transaction sequence submitted to the RDBMS to minimize resource contention and to maximize resource sharing. The advantages of the transaction reordering method are demonstrated through experiments with three commercial RDBMSs.

1. Introduction

Traditional workload management methods mainly focus on the current system status [1,2]. For example, in a typical RDBMS, the load controller only allows a certain number of complex queries to run concurrently. Also, if the system is in the danger of thrashing (i.e., admitting more transactions for execution will lead to excessive overhead and severe performance degradation [1]), the load controller may choose not to run any new transactions.

To support modern applications, users are continually requiring higher performance from RDBMSs. To meet this requirement, it is natural to ask whether or not we can use information about the interaction between queued and running transactions to improve the existing workload management methods. The answer to this question is "yes." In fact, in many instances, it is possible to improve the throughput of an RDBMS through the utilization of such information. More specifically, we can improve the throughput of an RDBMS that is processing a sequence of transactions by reordering these transactions before submitting them for execution. This is due to opportunities for either resource sharing among multiple transactions (e.g., sharing data in the buffer pool, or perhaps even sharing intermediate computations common to several transactions) or lowering resource contention (e.g., avoiding lock conflicts). Information about the interaction between queued and running transactions is essential in capturing such opportunities.

There are two main reasons why transaction reordering might be effective. The first is system independent – for example, it might be that a reordering of a transaction sequence truly eliminates some intrinsic lock conflicts between adjacent transactions and/or makes resource sharing possible. The second is system dependent – for example, a system may have a particular implementation of buffer management or concurrency control that renders one order of transactions superior to another. Even reordering to exploit system dependent opportunities is useful. Commercial RDBMSs are large, complex pieces of code, and changes in functionality can require a very long design-implement-test-release cycle. In many cases it may be

* Corresponding author. Tel.: +1 914 784 6932; fax: +1 914 784 6040.
E-mail addresses: gangluo@cs.wisc.edu, luog@us.ibm.com (G. Luo), naughton@cs.wisc.edu (J.F. Naughton), ellmann@wisc.edu (C.J. Ellmann), michael.watzke@teradata.com (M.W. Watzke).

0169-023X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2009.08.007

far simpler to do some reordering of transactions outside of the RDBMS before submitting them to the RDBMS for execution than it would be to change, say, the concurrency control subsystem of the RDBMS. This is especially true for database application developers who are unable to change the database engine.

This paper presents a general transaction reordering framework, which utilizes both the current system status and information about the interaction between queued and running transactions. The basic concept is simple and shown in Fig. 1. In an RDBMS, generally, at any time there are M_1 transactions waiting in a FIFO transaction admission queue Q to be admitted to the system for execution, while another M_2 transactions forming a set S_r are currently running in the system. Such a transaction admission queue Q is commonly used for load control purpose [1,2].

Those transactions in the transaction admission queue Q are the candidates for reordering. That is, the reorderer reorders the transactions waiting in Q so that the expected throughput of the reordered transaction sequence exceeds that of the original transaction sequence. In its reordering decisions, the reorderer exploits properties it deduces about the blocked transactions in Q and the properties it knows about the active transactions in S_r. The improvement in overall system throughput is a function of (a) the number of factors considered for reordering transactions, and (b) the quality of the original transaction sequence. The more factors considered, the better quality the reordered transaction sequence has. However, the time spent on reordering cannot be unlimited, as we need to ensure that the reordering overhead is smaller than the benefit we gain in throughput. Also, we need to ensure acceptable transaction response time in the sense that no transaction is subject to starvation.

There are a wide range of reordering algorithms that could be used. At the extremes, we could:

(1) Do no analysis. Run all the transactions in the order that they arrive at the RDBMS.
(2) Take a snapshot of the system. Analyze every possible order of the transactions and record the corresponding throughput. Pick the optimal order to run all the transactions.

The first extreme may be undesirable if some amount of reordering can improve the throughput. The second extreme is obviously unrealistic due to the exponential analysis overhead. Our goal is to find a good compromise between these two extremes. That is, under the constraint of acceptable transaction response time, we want to maximize the difference between the gain in throughput and the reordering overhead.

Reordering transactions requires CPU cycles. However, the increasing disparity between CPU and disk performance renders trading CPU cycles for disk I/Os more attractive as a way of improving DBMS performance [3]. As shown in detail in Section 4 below, some forms of transaction reordering can be regarded as a way to trade CPU cycles for disk I/Os. Also, our experiments in three commercial RDBMSs show that with minor overhead, our transaction reordering method greatly improves the throughput of a targeted class of transactions while it has only a minor impact on the throughput of other classes of transactions.

There are many resource allocation factors that can be considered for transaction reordering. In this paper, due to space constraints, we only consider two factors: lock conflicts (with an application to materialized view maintenance [4]) and buffer pool performance (with an application to exploiting synchronized scans [5]).

Transaction reordering can be implemented in two places: (1) inside the RDBMS, or (2) outside the RDBMS as an add-on module. These two choices are shown in Fig. 2, where the dotted rectangle denotes the RDBMS. The inside-RDBMS choice affords more opportunities for reordering, as the reorderer is tightly integrated with the RDBMS and can use detailed information about the current state of the system. Also, certain reordering policy (such as the one described in Section 4 for exploiting synchronized scans) can only be implemented using the inside-RDBMS choice if it requires support of other modules in the RDBMS. The outside-RDBMS choice has the advantage of not needing to change the database engine and is especially suitable

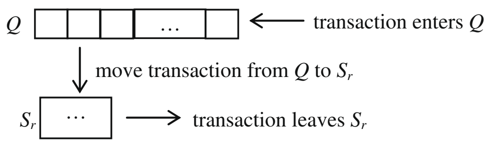

Fig. 1. The general transaction reordering framework.

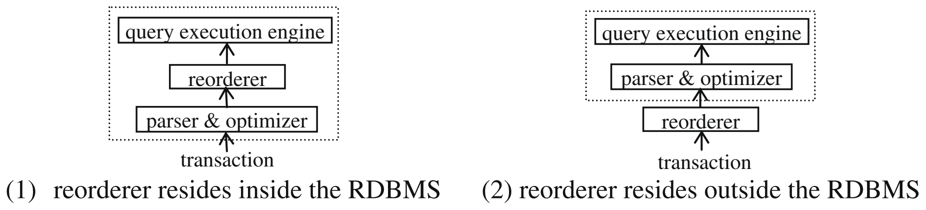

Fig. 2. Transaction reordering architecture.