Optimization of Multi-Join Queries in Shared-Nothing Systems

Kian-Lee Tan

Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore 0511

Received April 9, 1994.

Abstract

This paper proposes a semi-greedy framework for optimizing multi-join queries in shared-nothing systems. The plan generated by the framework comprises several pipelines, each performing several joins. The framework determines the “optimal” number of joins to be performed in each pipeline. The decisions are made based on the cost estimation of the entire processing plan. Two existing optimization algorithms are extended under the framework. An analytical model is presented and used to compare the quality of plans produced by each optimization algorithm. Our study shows that the new algorithms outperform their counterparts that are not extended.

Keywords: Multi-join optimization, shared-nothing systems, pipelining, hash join, segmented right-deep tree.

1 Introduction

In a shared-nothing system, a set of processing nodes are interconnected through a communication network. Each node has its own local memory and disk drives. A processor at one node has no direct access to memory or disks of other nodes. Data sharing among processors is realized by some message-passing mechanisms. To achieve high degree of parallelism, relations are usually partitioned into disjoint fragments and distributed across some number of nodes. In this way, each node can process the portion of database on its disk independently.

In this paper, we address the multi-join optimization problem for shared-nothing systems. In the literature [1,6], the basic technique that has been adopted for pipelining multi-join queries is to employ an iterative approach. In each iteration, several relations are selected such that all except one can fit in the total memory of the whole system. The joins of these selected relations are then processed in a pipelined fashion. This process is repeated until all the joins in the query are performed. Traditionally, optimization algorithms that are based on this technique greedily process as many joins as possible in each pipeline, thereby minimizing the number of pipelines[1,6]. However, these algorithms may fail to utilize the disk and I/O bandwidth effectively, and hence lead to high I/O cost and/or communication bottleneck. We propose a semi-greedy approach that determines the “optimal” number
of joins to be performed in each pipeline. The decisions are made based on the cost estimation of the entire processing plan. Two optimization algorithms that use the semi-greedy approach are studied and compared with their counterparts that use the traditional method.

We develop an analytical model to compare the quality of plans produced by each optimization algorithm on a variety of large, multi-join queries. The results of our study show that the optimization algorithms that are based on the semi-greedy approach outperform their counterparts that are based on the traditional method.

Throughout this paper, execution plans for multi-join queries will be depicted as trees, with each internal node corresponding to a join and each leaf node corresponding to a base relation. Only hash-based join methods will be considered, since they generally offer superior performance. As in [6], each join in an execution tree will have its building relation to the left and its probing relation to the right. Hash tables are built on building relations and probed by probing relations. If all the internal nodes of an execution tree have at least one leaf (that is, base relation) as a child, then the tree is called deep. Otherwise it is called bushy. A left-deep tree is a deep tree whose probing relations are restricted to base relations. Conversely, a right-deep tree is a deep tree whose building relations are restricted to base relations. We also assume that each relation is fully declustered across all the nodes. It is also expected that the total memory in the system is large enough to contain the entire of several relations. In this way, several join operations can be pipelined each time. This is not unreasonable since each node is expected to have a large memory and/or the system has a large number of nodes.

The remainder of the paper is organized as follows. Section 2 looks at how multi-join queries are processed in shared-nothing systems, and the related issues. In Section 3, we describe the proposed semi-greedy approach and the optimization algorithms that are studied. The cost model used by each optimization algorithm to compute the elapsed time is presented in Section 4. Section 5 presents the results of our performance study, and finally, conclusions are drawn in Section 6.

2 Processing Multi-Join Queries in Shared-Nothing Systems

2.1 The Execution and Process Models

All the optimization algorithms described in this paper are based on a segmented execution model, which is similar to the one described in [1,6]. In a segmented execution model, a query plan is broken down into a collection of memory-resident, right-deep segments that are executed one at a time. By memory-resident, we mean that the hash tables for all the building relations in a segment fit in memory. As in [1], we assume that the full result of each right-deep segment is always written to disk. Pipelining along the right-deep segment is used to achieve inter-operator