Selected Issues in Parallel Query Optimization
(Proposal for a Working Group)

Waqar Hasan
Stanford University
and
Hewlett-Packard Laboratories, USA
hasan@hpl.hp.com

Patrick Valduriez
Projet Rodin, INRIA
Rocquencourt, France
Patrick.Valduriez@inria.fr

1 Introduction

A key technology that contributed to the commercial success of relational databases is *query optimization*, the compilation of non-procedural queries into optimal plans for accessing the data. This problem now poses fresh challenges in the context of parallel database systems[DG92, Val93, vB89].

We propose a working group to discuss selected issues in parallel query optimization. Section 2 defines the problem and Section 3 briefly introduces the issues that we propose for discussion.

2 Problem Statement

2.1 Components of a Query Optimizer

A query optimizer compiles a non-procedural *query* into a procedural *execution plan*. In doing so, the optimizer searches some space of plans and chooses the plan with the least cost. The design of a query optimizer may be characterized in terms of the following three components[KZ88, GHK92]:

- **Execution Space:** This is the space of execution plans considered by the optimizer. When increasing the size of the execution space, a designer must balance the following tradeoffs *(a)* improved chances of finding good plans *(b)* increased time or space requirements of existing search algorithms *(c)* need for better search algorithms.
- **Cost Model:** The cost model provides a way of estimating the cost of an execution plan. It provides the total order, $<_{cost}$, for comparing two execution plans. The choice of the cost metric is critically related to the complexity of the search algorithm.
- **Search Algorithm:** This is the algorithm used to search the execution space for a plan with optimal cost. More specifically, it solves the following optimization problem:

Given a relational query q, an execution space E and a total order $<_{cost}$ over elements of E, find $p \in E$ such that
(1) p computes q
(2) There does not exist $p' \in E$ such that p' also computes q and $p' <_{cost} p$.

2.2 Challenge of Parallel Query Optimization

The parallel query optimization problem poses a new challenge since:

1. The execution space must be designed to model parallel execution.
2. The cost model must be designed to include some measure of *response time* since one goal of a Parallel DBMS is to *speedup* individual queries.
3. New search algorithms must be developed due to changes (1) and (2) above.

3 Interesting Questions

In this Section, we summarize the issues selected for discussion.

3.1 (Cost Model): What measure of cost should be used?

The development of efficient algorithms may be eased by the choice of simple cost measures. On the other hand, practical considerations about how much machine throughput may be sacrificed to deliver reduced response time demand more complicated cost measures.

Should the cost measure be response time or should it be modulated by some limits on the extra work generated by parallelism. Should response time be the elapsed time for the first tuple, the last tuple, or the kth tuple. XPRS[HS91] and DBS3[ZZBS93] use linear combinations of work (i.e. resource usage) and last-tuple response time; [GHK92] proposes measures that place limits on throughput degradation.

3.2 (Execution Space): How should parallel execution plans be represented?

What style of representation makes it easy to model the sources and deterrents of parallelism?

Join trees (also called processing trees) have been traditionally used to represent query execution plans. A join tree is a tree whose leaves represent base relations and interior nodes represent join operations. Details of execution such as the choice of join and access method are represented as *annotations*.

One approach towards representing parallel execution is to add extra annotations for parallelism (such as *degree of partitioning* to each join). Another approach is to use operator trees which are a macro-expansion of a join tree so as to identify the atomic units of execution. A third approach is the SVP model[SP92] which provides a formal framework for expressing divide-and-conquer mappings and allows parallel data processing to be expressed as series-parallel graphs.

The choice of representation has some important consequences. For example the operator tree representation allows XPRS[HS91, Hon92] to use different degrees of partitioning for the *build* and *probe* operators of the same join. What is the right balance between generality and simplicity?

3.3 (Execution Space): What join orders should be considered?

It is often assumed that bushy join trees are the right space to be searched. This choice has a price. It substantially increase the space and time requirements of optimization algorithms. Are the advantages of bushy trees worth the additional cost of search?

Bushy trees provide the opportunity to execute two sibling joins independently in parallel. However, assuming that the number of joins in a query is much smaller than the number of processors, this is not a scalable form of parallelism. Do data partitioning and pipelining generate sufficient parallelism that bushy trees may be ignored?

It is possible to show bushy trees that represent the "same execution" as a left-deep tree. Consider for example $(R_1 \bowtie (R_2 \bowtie R_3))$ and $((R_1 \bowtie R_2) \bowtie R_3)$ in which each join is a nested-loops join.

What do bushy trees really add? Are there "good enough" classes of shapes (right deep[SD90], zig-zag[ZZBS93], segmented)?

3.4 (Execution Space): Can/Should the execution spaces for shared-memory and shared-nothing systems be combined?

Query processing techniques developed for XPRS and GAMMA[DGS+90] are quite different. For instance in XPRS, producers can simply write their output to a shared data structure without knowing the *number* of consumers. In GAMMA, the producers are explicitly given split tables that specify the number and location of each consumer. Another example is that Asynchronous pipelines are totally avoided in XPRS while they are used as a routine matter in GAMMA.

Is a unified execution space possible/desirable or should the two be considered separate problems?

A unification will be desirable if hybrid architectures consisting of a shared-nothing system in which each node is a shared-memory multiprocessor end up providing the best price-performance.

3.5 (Search Algorithm): What are appropriate tactics to solve the problem?

Should the optimization problem be broken into phases? If so, what are the right stages? How strong is the interaction across stages? For example Hong and Stonebraker use a 2-phase algorithm. First the best join order is found using work as the measure of cost. The sequential plan is then dynamically parallelized using a heuristic to maximize machine utilization. On the other hand, [GHK92, LST91] consider the effect of hot spots on response time while deciding the join order. [HLY93] extends the work of [LST91] to account for skew as well.

Should the execution space be restricted to simplify the problem? If so, what are the right simplifications? For example [HLY93, LST91] restrict the execution space to synchronized join trees which prohibit pipelined parallelism; [ZZBS93] models memory usage but excludes resource contention by using the notion of *logical* homes of relations; [GHK92] removes hot spots but excludes memory usage.

Heuristic algorithms proposed for centralized query optimization failed to get adopted. Are heuristics a promising way to go for parallel query optimization? If so, how can we gain confidence in their performance? [HLY93, LST91, SE93] propose heuristics that interleave join ordering and parallelization.

3.6 (Other): Static vs Dynamic Optimization?

If a compiler had perfect knowledge then all decisions could be taken at compile time. If perfect (infinitely fast) algorithms were available then all decisions could be taken at runtime. In an imperfect world how do we make the cut between compile-time and runtime?

For example XPRS[Hon92] decides the join order at compile time. The degree of partitioning of each unit of execution is determined dynamically at runtime. [LST91] determine the join order and degree of partitioning at compile time; [HLY93] adjust the join order to account for skew at runtime.

To what extent should the impact of the deterrents of parallelism such as limits on available resources, resource contention or skew be handled at compile time. Is it possible for compile time techniques to reduce the severity of problems for the runtime? On the other hand, what aspects of plans should be changeable at runtime?

4 References

[DG92] D. J. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[DGS+90] D.J. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[GHK92] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query Optimization for Parallel Execution. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, June 1992.

[HLY93] K.A. Hua, Y. Lo, and H.C. Young. Including the Load Balancing Issue in The Optimization of Multi-way Join Queries for Shared-Nothing Database Computer. In *Second International Conference on Parallel and Distributed Information Systems*, San Diego, California, January 1993.

[Hon92] W. Hong. Exploiting inter-operation parallelism in xprs. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, June 1992.

[HS91] W. Hong and M. Stonebraker. Optimization of parallel query execution plans in xprs. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, December 1991.

[KZ88] R. Krishnamurthy and C. Zaniolo. Optimization in a logic based language for knowledge and data intensive applications. In *Proceedings of the Extending Database Technology Conference*, 1988.

[LST91] H. Lu, M-C. Shan, and K-L. Tan. Optimization of Multi-Way Join Queries for Parallel Execution. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, Barcelona, Spain, September 1991.

[SD90] D.A. Schneider and D.J. DeWitt. Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines. In *Proceedings of the Sixteenth International Conference on Very Large Data Bases*, Brisbane, Australia, 1990.

[SE93] J. Srivastava and G. Elsesser. Optimizing Multi-Join Queries in Parallel Relational Databases. In *Second International Conference on Parallel and Distributed Information Systems*, San Diego, California, January 1993.

[SP92] P. Valduriez S. Parker, E. Simon. SVP, a Data Model Capturing Sets, Streams and Parallelism. In *Proceedings of the Eightteenth International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada, August 1992.

[Val93] P. Valduriez. Parallel Database Systems: Open Problems and New Issues. *Distributed and Parallel Databases: An International Journal*, 1(2):137–165, April 1993.

[vB89] Gunter von Bulltzingsloewen. Optimizing SQL Queries for Parallel Execution. In G. Graefe, editor, *Workshop on Database Query Optimization*, pages 189–195. Oregon Graduate Center, Portland, Oregon, May 1989. CSE Technical Report 89-005.

[ZZBS93] M. Ziane, M. Zait, and P. Borla-Salamet. Parallel Query Processing in DBS3. In *Second International Conference on Parallel and Distributed Information Systems*, San Diego, California, January 1993.