INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Indian Institute of Information Technology, Dharwad

Final Report

Query Processing and Optimization of Parallel Database System in Multi Processor Environments

Nitin Singh

17BCS019

Supervised by

Dr. Uma S

30, November, 2019

Introduction

Scope of Topic

In present scenario parallel database systems are being applicable in a broad range of systems, right from database applications (OLTP) server to decision support systems (OLAP) server. These developments involve database processing and querying over parallel systems. A means to the success of parallel database systems, particularly in decision-support applications (Data warehousing), is parallel query optimization. Given a SQL query, parallel query optimization has the goal of finding a parallel plan that delivers the query result in minimal time. Various useful and competent, optimizing solutions to be implemented for the parallel databases. Parallel DBS attempt to develop recently in order to make high performance and high-availability database servers at a much lower price for multiprocessor computer architectures than mainframe computers. The objective of this paper is define a novel approach on how to achieve parallelism for relational database multithreaded query execution use to maximum resource utilization of CPU and memory.

This work deals with query optimization for parallel object-oriented databases using rule-based optimizers. . The implementation approach shows that the parallelism extraction technique combined with a rulebased optimizer generator tool can help to produce a parallel optimizer from a non-parallel one with minimum effort.

The query optimization is a very big field in the context of database management. It has been studied in a great variety of contexts and from many different perspectives, giving rise to several diverse solutions in each case. It focuses on the skewing problem in parallel database architectures with proper survey of literature.

Literature Study

Over the last two decades parallel query processing in database systems was the topic of extensive research. Its outcome is now undoubtedly in daily use as part of major commercial DBMS. Most of the research was concentrated on shared nothing (SN) architectures, e.g. the research prototypes Gamma [4] and Bubba [5]. It is well-designed for modern grid and cluster computing and the focal point of other approaches on symmetric multiprocessing architectures (SMP), such as XPRS and Volcano [5, 6, and 7]. While many other scholars has researched algorithm for database optimization based on CMP. But the mainly of their work were focused on the optimization of join operation considering the L2-cache and the parallel buffers of the shared main memory [8, 9, and 10].

This work is influenced by the ideas presented by Hasan in [7] regarding optimization of SQL queries for parallel machines, and the work of Hong on two-phase parallel query optimization [9]. We follow the same approach adopted by Hasan, which divides parallel query optimization into two phases comprised of join ordering and query rewriting (JOQR) and parallelization. Given this division, we address the issue of implementing parallelism extraction in the parallelization sub-phase, for the OQL language.

Many areas of modern biology are concerned with the management, storage and visualization of data in databases [8] and there exists a few appropriate query languages, like Pathway Query Language (PQL) [2] for such complex data structure In a parallel database system (PDBS), as in a biological database, the effect of skewing is considered one of the most costly goals to achieve.

New proposed Solutions - Research papers

Paper 1: [Query Processing and Optimization of Parallel Database System in Multi Processor Environments, 2012 Sixth Asia Modelling Symposium]

Parallel database systems combine data management and parallel processing techniques to provide high performance, high- availability and scalability for data intensive applications. An ideal approach on a parallel machine would scale easily at a low-cost, and would demonstrate linear speedup and scale up for number of processors. Mainly two architectures have emerged from the quest for an ideal parallel machine: they are shared-memory (SM) and shared-nothing (SN) architecture.

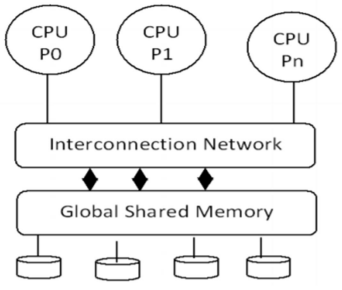


Figure 1 Shared memory machine

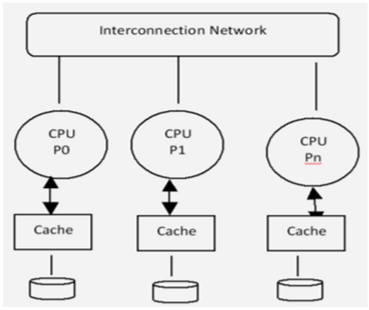


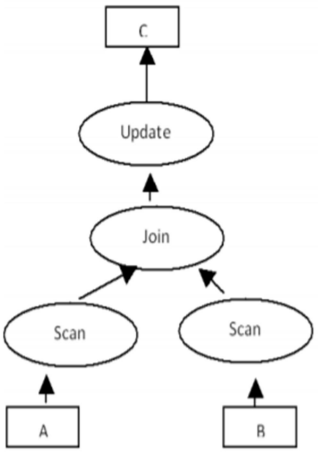
Figure 2 Shared nothing machine

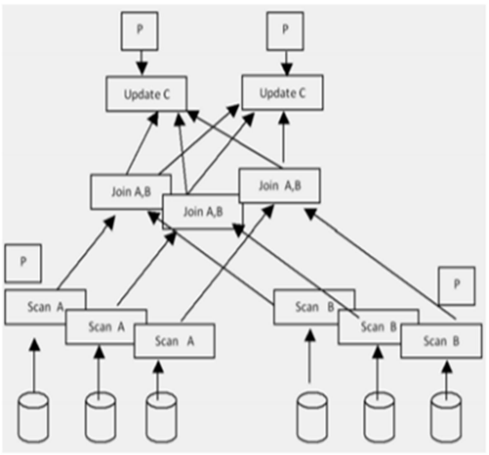
In the shared memory architecture shown in fig. 1, each processor has access to all disks in the system and to a global shared memory. Shared-nothing architecture machine works on the principle of ownership, that each processor in a cluster has individual ownership of the data on that node. In parallel database system PQO allows one to break-up a given SQL statement so that its parts can run simultaneously on different processors in a multi-processor machine. Typical operations that can run in parallel are: full table scans, sorts, sub-queries, etc. The execution strategy that is proposed here as follows: First, the total amount of work that has to be done to evaluate the query

is minimized. Then, try to distribute that minimal amount of work equally over the available processors.

First approach when memory module not shared. Separate the all relations along with it predicates those are use in users query. Assign the separate relation to separate processor along with its memory module. Transfer the intermediates result set with predicates to other CPU to evaluate result

Second approach when memory module shared distributing the tuples of a relation over several disks / memory modules which will help to allow parallel databases to exploit the I/O bandwidth of multiple memory module. In case of select statements apply the block level locking mechanism with read lock. So that one CPU access the data it should be lock for other CPU till the complete execution of SQL statement. If the SQL statement belongs in update statement category then apply the record level locking mechanism with read and write lock Once the record will update by CPU, with the help of cache coherence property update the data in all corresponding memory module

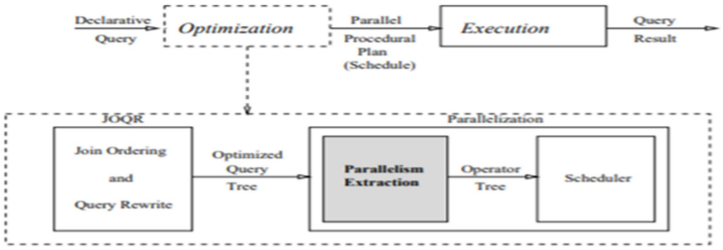




Simple conventional plan But when the same query execute in multiprocessor environment then the query execute in parallel manner shown in Fig and response time will be minimize and system throughput increase dramatically.

Paper 2: [Rule-Based Parallel Query Optimization for OQL Using a Parallelism Extraction Technique]

Our approach departs from a conventional (non-parallel) rule-based query optimizer and employs a parallelism extraction technique to unveil opportunities for independent parallelism and pipelined parallelism that can be exploited during query optimization for parallel OODBMS. Parallel extraction technique based on analyzing the physical algebra generated by a non-parallel optimizer. The primary goal of applying parallelism extraction is to expose the opportunities for transforming QEPs in a way that takes advantage of an underlying parallel architecture.

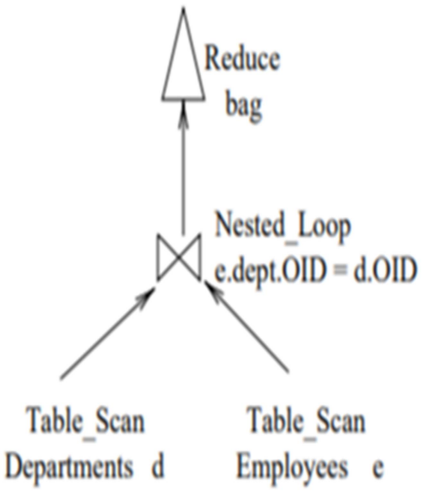


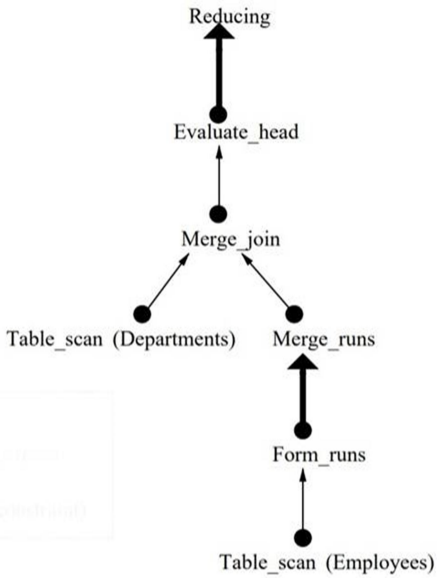
This work addresses the issues related to incorporating parallelism extraction techniques into the optimization step of a non-parallel query optimizer to implement modules of a parallel optimizer for OQL. We have adopted a solution based on the use of a rule-based optimizer generator tool and its associated rule language OPTL. Although we provided a solution based on the use of OPTL as the query rewriting rule language, the ideas behind parallelism extraction can also be applied to build parallel query optimizers, regardless of the data model or query language supported. The use of a rule based optimizer and the analysis of an existing implementation for OQL using the parallelism extraction technique provides a simple and fast way to prototype a parallel optimizer for OQL. Using the parallelism extraction technique, we have obtained a target parallel physical algebra that abstracts the atomic operations (pieces of code)

supported by a scheduler for a parallel OODBMS that can be easily realized using current parallel architectures. The resulting plans, besides unveiling parallelism opportunities, are easily adaptable within the context of different parallel architectures.

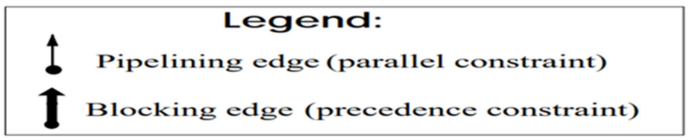
Example

select struct(E: e.name, D: e.dept.name) **from** e in Employees;





QEP before parallel extraction Physical Plan after parallel extraction



The timing constraints between the physical operators, identified by the analysis of the operators' behaviour and relationships in a procedural plan

op-op	table scan	merge join	form runs	merge runs	eval. head	red.	group	nest.
table scan		par.c.	par.c.		par.c.		par.c.	
merge join		par.c.	par.c.		par.c.		par.c.	
form runs				pre.c.				
merge runs		par.c.	par.c.		par.c.		par.c.	
eval. head						pre.c.		
red.		par.c.	par.c.		par.c.		par.c.	
group								pre.c.
nest.					par.c.			

Table 1. Timing constraints

Paper 3:[On the performances in simulation of parallel databases: an overview on the most recent techniques for query optimization , 2009 International Workshop on High Performance Computational Systems Biology]

In this paper, we simulated a parallel database and developed a partition join where both skew effects and load balancing techniques were adopted. Then we compared various techniques to achieve a conceptual framework for the assessment of existing processing approaches and for the development of new algorithms. We found that highly dynamic techniques have great advantages with respect to their own complexity as well as to the expected success of load balancing because they rely on observed execution times rather than inaccurate cost estimates. We particularly favour on-demand scheduling methods treating execution skew and strongly recommend to pursue this approach further. In addition, we noted an unexplored potential for skew treatment in the design of database schemata and materialized views. We consider this an interesting line of research for efficient parallel query processing. Our solution tries to combine the advantages of hash partitioning and symmetric join which is described in Fragment and Replicate join technique. This technique splits the pairs to be tested over several processors. Each processor computes part of the join, and then the results are merged. Then we analysed this procedure by comparative studies with other join techniques. We present a classification of skew effects and also load balancing approaches.

PARTITIONED JOIN

This technique works in this way: the system partitions the relations r and s each into n partitions, denoted r0, r1,. . . , rn-1 and s0, s1,. . . , sn--1. The system sends partitions ri and si to processor Pi , where their join is computed locally. The partitioned join works correctly only if the join is an equijoin and if we partition r and s by the same

partitioning function on their join attributes. In a partitioned join, there are two different ways of partitioning r and s: • Range partitioning on the join attributes • Hash partitioning on the join attributes ,In either case, the same partitioning function must be used for both relations. For range partitioning, the same partition vector must be used for both relations. For hash partition, the same hash function must be used on both relations. Once the relations are partitioned, we can use any join technique like hash-join, merge-join, or nested-loop join, locally at each processor Pi to compute the join of ri and si .

EXPERIMENTAL RESULTS

We simulated a simple program by which we generated random texts written in text files described above which were saved in the disk and acted as the database. Then we computed the time required to execute a join query in one of the nodes. By varying different parameters, we simulated the different join techniques in parallel environment. After getting these values of transfer time and assemble time, we got the plots as follows

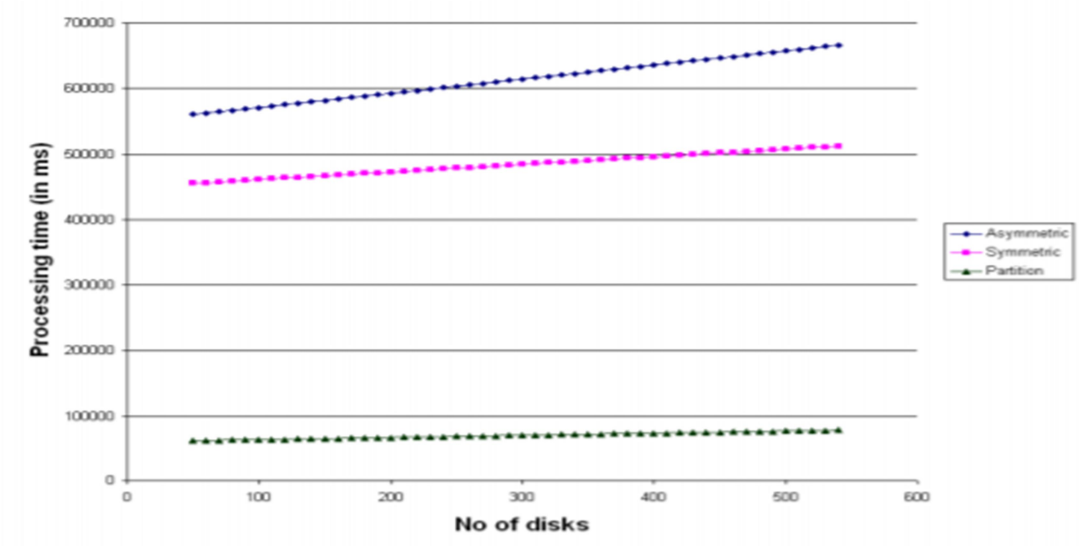


Figure 2. In this figure the no. of disks is varied so that the disk transfer time changes. In the X-axis, we have No. of disks that are varied and in the Y-axis, we have the observed processing time in millisecond

In figure 2., Here, we found that the partition join requires less processing time than both of asymmetric and symmetric join and

remains almost unchanged. In comparison between symmetric and asymmetric join, it has been found that symmetric join is better with respect to the processing time for asymmetric join.

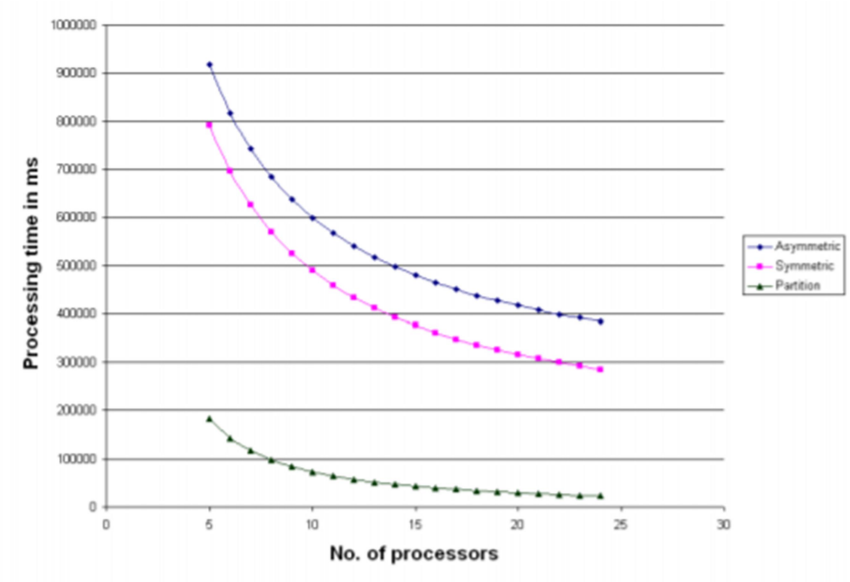


Figure 3. In the X-axis, we have variable no. of processors and in the Y-axis, we have the observed processing time in millisecond

When we varied the number of processors, we observed that in the beginning the partition join required more time and it decreases if we increase the number of processors. In all the cases it takes less processing time than both of asymmetric and symmetric join. It is evident from the Figure 3 that symmetric join performs better than asymmetric join.

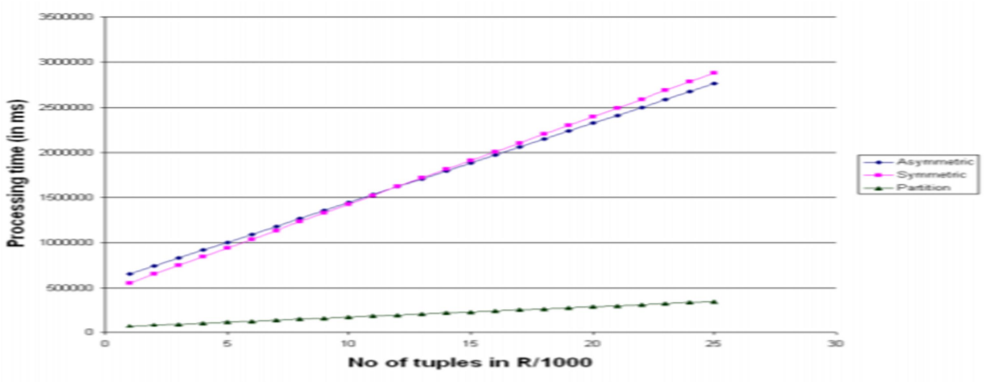


Figure 4. In this figure the no.of tuples that are sent to other nodes through network is varied. In the X-axis, we have No. of tuples in R normalized by the factor 1000 and in the Y-axis, we have the observed processing time in millisecond

In figure 4, we found that the partition join requires less processing time than both of asymmetric and symmetric join and remains almost unchanged. In comparison between symmetric and asymmetric join, it has been found that symmetric join is better with respect to the processing time for asymmetric join

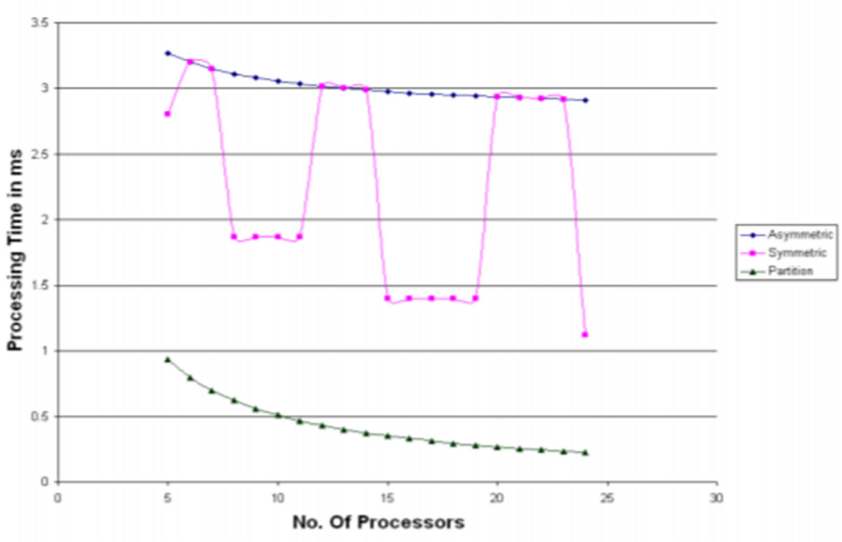


Figure 5. In this figure the no.of processors is varied in case of parallel join. In the X-axis, we have No. of processors that are varied and in the Y-axis, we have the observed processing time in millisecond

In figure 5 we observed some oscillations in case of symmetric join. The reason behind is that when the multiplication of the number of tuples for both relation matches or very close to the number of processors, it takes less time for computation in symmetric case. Otherwise, the symmetric join acts as the asymmetric join. But as a whole the partition join acts better regarding the processing time. As a matter of fact, the processing time is greater when the number of processors is low, but it decreases with the increase of the processing units.

Both in single processor and parallel environment, partitioned join technique showed far better performance in all the variations of the different parameters. We find that highly dynamic scheduling methods based on observed execution times are superior in both complexity and attainable load balance. We also suggest the tuning of database schemata as a new anti-skew measure.

References

- https://ieeexplore.ieee.org/document/6243945/
- https://ieeexplore.ieee.org/document/707485/
- https://ieeexplore.ieee.org/document/5298684/
- https://www.tutorialcup.com/dbms/object-based-data-models.htm
- https://www2.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter8/node3.html