Query Load Balancing in Parallel Database Systems

Luc Bouganim

Query Load Balancing in Parallel Database Systems

Luc Bouganim
INRIA Saclay Île de France & UVSQ Versailles, France

Luc Bouganim
Email: Luc.Bouganim@inria.fr

Synonyms

Resource scheduling

Definition

The goal of parallel query execution is minimizing query response time using inter- and intraoperator parallelism. Interoperator parallelism assigns different operators of a query execution plan to distinct (sets of) processors, while intraoperator parallelism uses several processors for the execution of a single operator, thanks to data partitioning. Conceptually, parallelizing a query amounts to divide the query work in small pieces or tasks assigned to different processors. The response time of a set of parallel tasks being that of the longest one, the main difficulty is to produce and execute these tasks such that the query load is evenly balanced within the processors. This is made more complex by the existence of dependencies between tasks (e.g., pipeline parallelism) and synchronizations points. Query load balancing relates to static and/or dynamic techniques and algorithms to balance the query load within the processors so that the response time is minimized.

Historical Background

Parallel database processing appeared very early in the context of database machines in the 1970s. Parallel algorithms (e.g., hash joins) were later proposed in the early 1980s, where tuples are uniformly distributed at every stage of the query execution. However several works (e.g., [*8*]) gave considerable evidence that data skew, i.e., nonuniform distribution of tuples, exists, and its negative impact on query execution was shown in, e.g., [*7*]. This motivated numerous studies [*10*] on intra- and interoperator load balancing in the 1990s.

Foundations

Load-balancing problems can appear with intraoperator parallelism (variation in partition size, namely, *data skew*) and interoperator parallelism (variation in the complexity of operators, synchronization problems). Intra- and interoperator load-balancing problems are first detailed on a simplified query execution plan example considering a static allocation of processors to the query operators. The main load-balancing techniques proposed to address these problems are described next.

Load-Balancing Problems

Figure 1a shows a simplified query execution plan for the following query: "Select T.b from R, S, T where R.Rid = S.Rid and S.Sid = T.Sid and R.a = value" (the scan and project operators were omitted to simplify the drawing). The following assumptions are made: (i) the degree of parallelism (i.e., number of processors allocated) for the selection on R (called σR), the join with S (called ⍰⍰ *S*⍰⍰S) and the join with T (called ⍰⍰⍰⍰T), has been statically fixed, using a cost model, to, respectively, 2, 3, and 2, and (ii) these operators are processed in pipeline, thus leading to a total degree of parallelism of 7.

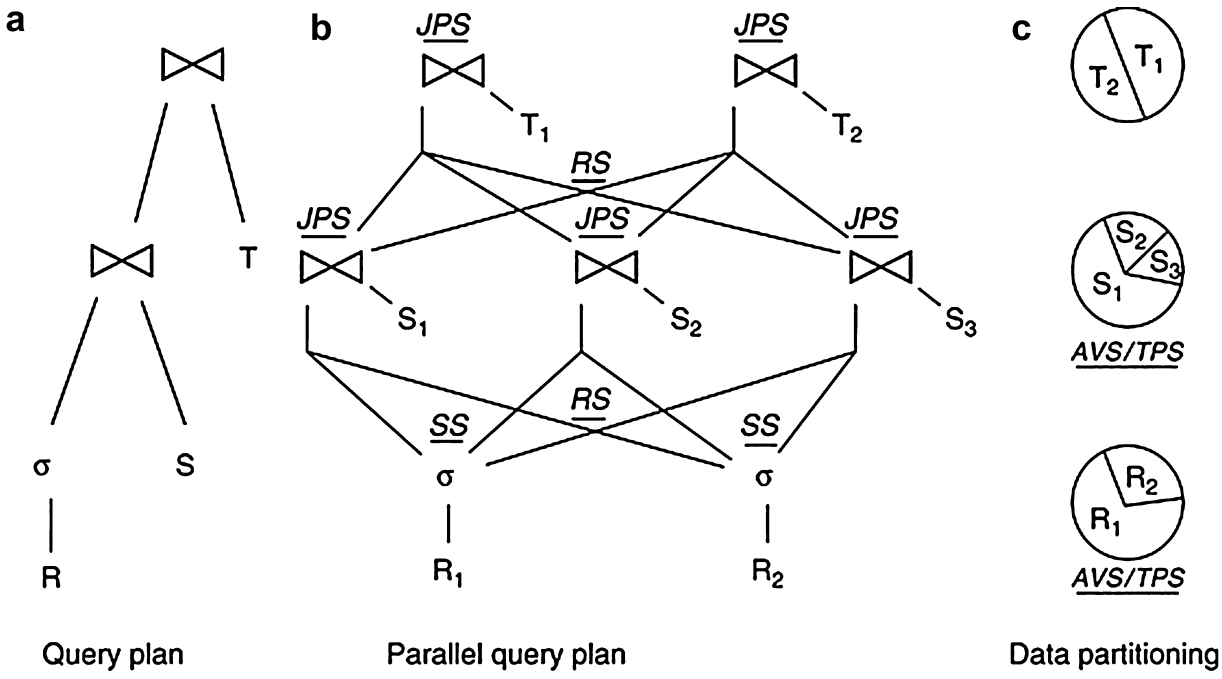

Fig. 1 Intra- and interoperator load-balancing problems on a simple example

Intraoperator load-balancing issues are first illustrated using the classification proposed in [*13*]. As shown in Fig. 1c, R and S are poorly partitioned because of *attribute value skew (AVS)* inherent in the data set and/or *tuple placement skew (TPS)*. The processing time of the two instances σR1 and σR2 are thus not equal. The case of ⍰⍰⍰⍰S is likely to be worse (see Fig. 1b). First, the number of tuples received can be different from one instance to another because of poor redistribution of the partitions of R (*redistribution skew*, *RS*) or variable selectivity according to the partition of R processed (*selectivity skew*, *SS*). Finally, the uneven size of S partitions (*AVS/TPS*) yields different processing times for tuples sent by the σR operator, and the result size is different from one partition to the other due to join selectivity (*join product skew*, *JPS*). The skew effects are therefore propagated toward the query tree, and even with a perfect partitioning of T, the processing time of ⍰⍰⍰⍰T1 and ⍰⍰⍰⍰T2 can be highly different (uneven size of their left input resulting from ⍰⍰⍰⍰S). Intraoperator load balancing is thus difficult to achieve statically, given the combined effects of different types of data skew.

In order to obtain good load balancing at the interoperator level, it is necessary to choose how many and which processors to assign to the execution of each operator. This should be done while taking into account pipeline parallelism, which requires interoperator communication and introduces precedence constraints between operators (i.e., an operator must be terminated before the next one begins). In [*15*], three main problems are described:

(i) The degree of parallelism and the allocation of processors to operators, when decided in the parallel optimization phase, are based on a possibly inaccurate cost model. Indeed, it is difficult, if not impossible, to take into account highly dynamic parameters like interference between processors, memory contentions, and, obviously, the impacts of data skew.

(ii) The choice of the degree of parallelism is subject to errors because both processors and operators are discrete entities. For instance, considering Fig. 1b, the number of processors for σR, ⋈S, and ⋈T may have been computed by the cost model as, respectively, 1.5, 3.8, and 2.4 and have been rounded to 2, 3, and 2 processors. But the good distribution, taking into account data skew on S partitions, should have been 1, 4, and 2.

(iii) The processors associated with the latest operators in a pipeline chain may remain idle a significant time. This is called the pipeline delay problem. For instance, while tuples do not match the selection on R or the join with S, processors assigned to ⋈T remain idle.

In a shared-nothing architecture, the interoperator load-balancing problem is even more complex, since the degree of parallelism and the set of processors assigned for some operators are constrained by the physical placement of the manipulated data. For instance, if R is partitioned on two nodes, σR must be executed on these nodes.

This simple example thus shows that static allocation of processors to operators is usually far from optimal, thus advocating for more dynamic strategies. In the following section, existing proposals at the intra- and interoperator level are detailed.

Intraoperator Load Balancing

Good intraoperator load balancing depends on the degree of parallelism and on the allocation of processors for the operator. For some algorithms, e.g., the parallel hash join algorithm, these parameters are not constrained by the placement of the data. Thus, the home of the operator (the set of processor where it is executed) must be carefully decided. The skew problem makes it hard for a parallel query optimizer to make this decision statically (at compile-time) as it would require a very accurate and detailed cost model. Therefore, the main solutions rely on adaptive techniques or specialized algorithms which can be incorporated in the query optimizer/processor. These techniques are described below in the context of parallel joins, which has received much attention. For simplicity, each operator is given a home either statically or just before execution, as decided by the query optimizer/processor.

Adaptive techniques: The main idea is to statically decide on an initial allocation of the processors to the operator (using a cost model) and, at execution time, adapt this decision to skew using load reallocation. A simple approach to load reallocation is detecting the oversized partitions and partition them again onto several processors (among those already allocated to the operator) to increase parallelism [6]. This approach is generalized in [2] to allow for more dynamic adjustment of the degree of parallelism. It uses specific *control operators* in the execution plan to detect whether the static estimates for intermediate result sizes differ from the runtime values. During execution, if the difference between estimate and real value is high enough, the control operator performs data redistribution in order to prevent join product skew and redistribution skew. Adaptive techniques are useful to improve intraoperator load balancing in all kinds of parallel architectures. However, most of the work has been done in the context of shared-nothing where the effects of load unbalance are more severe on performance.

Specialized algorithms: Parallel join algorithms can be specialized to deal with skew. The approach proposed in [3] is to use multiple join algorithms, each specialized for a different degree of skew, and

to determine the best at execution time. It relies on two main techniques: range partitioning and sampling. Range partitioning is used instead of hash partitioning (in the parallel hash join algorithm) to minimize redistribution skew of the building relation. Thus, processors can get partitions of equal number of tuples, corresponding to different ranges of join attribute values. To determine the values that delineate the range values, sampling of the building relation is used to produce a histogram of the join attribute values, i.e., the numbers of tuples for each attribute value. Sampling is also useful in determining which algorithm to use and which relation to use for building or probing. The parallel hash join algorithm can then be adapted to deal with skew as follows: (i) Sample the building relation to determine the partitioning ranges. (ii) Redistribute the building relation to the processors using the ranges. Each processor builds a hash table containing the incoming tuples. (iii) Redistribute the probing relation using the same ranges to the processors. For each tuple received, each processor probes the hash table to perform the join. This algorithm can be further improved to deal with high skew using additional techniques and different processor allocation strategies [3]. A similar approach is to modify the join algorithms by inserting a scheduling step which is in charge of redistributing the load at runtime [14].

Interoperator Load Balancing

The interoperator load-balancing problem was extensively addressed during the 1990s. Since then many processor allocation algorithms have been proposed for different target parallel architectures and considering CPU, I/Os, or other resources, such as available memory.

The main approach in shared-nothing is to determine dynamically (just before the execution) the degree of parallelism and the localization of the processors for each operator. For instance, the rate match algorithm [9] uses a cost model in order to define the degree of parallelism of operators having a producer-consumer dependency such that the producing rate matches the consuming rate. It is the basis for choosing the set of processors which will be used for query execution (based on available memory, CPU, and disk utilization). Many other algorithms are possible for the choice of the number and localization of processors, for instance, by a dynamic monitoring and adjustment of the use of several resources (e.g., CPU, memory, and disks) [11].

Shared-disk and shared-memory architectures provide more flexibility since all processors have equal access to the disks. Hence there is no need for physical relation partitioning and any processor can be allocated to any operator [12].

Considering the shared-disk architecture, Hsiao et al. [5] propose to assign processors recursively from the root up to the leaves of a so-called allocation tree. This tree is derived from the query tree, each pipeline chain (i.e., set of operators having pipeline dependencies) being represented as a node. The edges of the allocation tree represent precedence constraints. All available processors are assigned to the root node of the allocation tree (the last pipeline chain to be executed). Then, a cost model is used to divide the CPU power between each child of the root in order to ensure that all the data necessary for the execution of the root pipeline chain will be produced synchronously.

The approach proposed in [4] for shared memory allows the parallel execution of independent pipeline chains called tasks. The main idea is combining IO-bound and CPU-bound tasks to increase system resource utilization. Before execution, a task is classified as IO-bound or CPU-bound using cost model information. CPU-bound and IO-bound tasks can then be run in parallel at their optimal IO-CPU balance, by dynamically adjusting the degree of intraoperator parallelism of the tasks.

Intra-query Load Balancing

Intra-query load balancing combines intra- and interoperator parallelism. To some extent, given a parallel architecture, the load-balancing techniques presented above can be extended or combined. For instance, the control operators used a priori for intraoperator load balancing can modify the degree of parallelism of an operator, thus impacting interoperator load balancing [2].

A general load-balancing solution in the context of hierarchical parallel architectures (a shared-nothing system whose nodes are shared-memory multiprocessors) is the execution model called dynamic processing (DP) [1]. In such systems, the load-balancing problem is exacerbated because it must be addressed both locally (among the processors of each shared-memory node) and globally (among all nodes). The basic idea of DP is decomposing the query into self-contained units of sequential processing, each of which can be carried out by any processor. Intuitively, a processor can migrate horizontally (intraoperator parallelism) and vertically (interoperator parallelism) along the query operators. This minimizes the communication overhead of internode load balancing by maximizing intra and interoperator load balancing within shared-memory nodes.

Key Applications

Load-balancing techniques are essential in applications dealing with very large databases and complex queries, e.g., data warehousing, data mining, business intelligence, and more generally all OLAP (online analytical processing) applications.

Data Sets

DBGen, a synthetic data generator, can be used to generate biased data distribution, for studying intraoperator load-balancing issues. It allows generating data with nonuniform distribution (Zipfian, Poisson, Gaussian, etc.). See http://research.microsoft.com/~Gray/DBGen/

Recommended Reading

1. Bouganim L, Florescu D, Valduriez P. Dynamic load balancing in hierarchical parallel database systems. In: Proceedings of the 22th International Conference on Very Large Data Bases; 1996. p. 436–47.

2. Brunie L, Kosch H. Control strategies for complex relational query processing in shared nothing systems. ACM SIGMOD Rec. 1996;25(3):34–9.
3. De Witt DJ, Naughton JF, Schneider DA, Seshadri S. Practical skew handling in parallel joins. In: Proceedings of the18th International Conference on Very Large Data Bases; 1992. p. 27–40.

4. Hong W. Exploiting inter-operation parallelism in XPRS. In Proceedings of the ACM SIGMOD International Conference on Management of Data; 1992. p. 19–28.

5. Hsiao H, Chen MS, Yu PS. On parallel execution of multiple pipelined hash joins. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, New York; 1994. p. 185–96.

6. Kitsuregawa M, Ogawa Y. Bucket spreading parallel hash: a new, robust, parallel hash join method for data skew in the super database computer. In: Proceedings of the 16th International Conference on Very Large Data Bases; 1990. p. 210–21.

7. Lakshmi MS, Yu PS. Effect of skew on join performance in parallel architectures. In: International Symposium on Databases in Parallel and Distributed Systems, Austin; 1988. p. 107–20.

8. Lynch C. Selectivity estimation and query optimization in large databases with highly skewed distributions of column values. In: Proceedings of the 14th International Conference on Very Large Data Bases; 1988. p. 240–51.

9. Metha M, De Witt D. Managing intra-operator parallelism in parallel database systems. In: Proceedings of the 21th International Conference on Very Large Data Bases, Morgan Kaufmann, San Francisco, 1995. p. 382–94.

10. Özsu T, Valduriez P. Principles of Distributed Database Systems (2nd edn.). Prentice Hall; 1999 (3rd edn., forthcoming).

11. Rahm E, Marek R. Dynamic multi-resource load balancing in parallel database systems. In: Proceedings of the 21th International Conference on Very Large Data Bases; 1995.

12. Shekita EJ, Young HC. Multi-join optimization for symmetric multiprocessor. In Proceedings of the 19th International Conference on Very Large Data Bases; 1993. p. 479–92.

13. Walton CB, Dale AG, Jenevin RM. A taxonomy and performance model of data skew effects in parallel joins. In: Proceedings of the 17th International Conference on Very Large Data Bases; 1991. p. 537–48.

14. Wolf JL, Dias DM, Yu PS, Turek J. New Algorithms for parallelizing relational database joins in the presence of data skew. IEEE Trans Knowl Data Eng. 1994;6(6):990–7.

15. Wilshut N, Flokstra J, Apers PG. Parallel evaluation of multi-join queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data; 1995. p. 115–26.