Adaptive parallelization in multi-core systems

Mrunal Gawade
CWI, Amsterdam
mrunal.gawade@cwi.nl

Motivation: With the presence of multi-core CPUs even in desktop computers the opportunities for query parallelism exploitation in database systems are manifold. Exploiting these opportunities calls for a re-look at the optimal parallel plan generation problem. An optimal parallel plan generation in a multi-core CPU setting however is very hard.

Hypothesis: The conventional parallel plan generation methods use cost based or heuristic based optimizations. We envision an alternative approach for parallel plan generation using the actual query execution time as the parallelization decision metric. During successive query invocations an old query plan is morphed into a new query plan by parallelizing the most expensive operator. The parallelization step introduces two new operators which operate on two equi-range partitions. We term this approach the *adaptive parallelization.* The explosion of parallel plan search space is thus controlled by selective parallelization of the most expensive operator. Adaptive parallelization is evaluated by an implementation in MonetDB, the open source columnar database system. We show an optimal parallel plan generation from a large parallel plan search space within bounded time constraints.

Research challenge: A multi-core CPU setting increases the query optimization problem complexity by expanding the already large query plan space. To illustrate one such simple case consider a relational algebra plan.

Join(Select(A), Select(B)).

The plan needs to be executed in an n-core CPU setting. In the case of intra-operator parallelism of the Select operator, the query optimizer could range partition the column from table A and B, and assign one Select operator per range. The number of partitions could vary based on different possibilities such as data type, data range, data distribution, query selectivity, presence of access optimization structures such as indices, size of data being merged after partitions, and position in the plan where the merging occurs.

Assume a case where the upper bound on the maximum number of partitions for both Select operands in an n-core CPU is n. Consider the most simple case of equi-range disjoint partitions over both Select operands. Let $m1$ and $m2$ represent the number of partitions of each operand. The minimum equi-range disjoint partitions are *m=2 (m1=1 and m2=1)*, and the maximum equi-range disjoint partitions are *m=n*, where $m = m1 + m2$. Possible options for m1

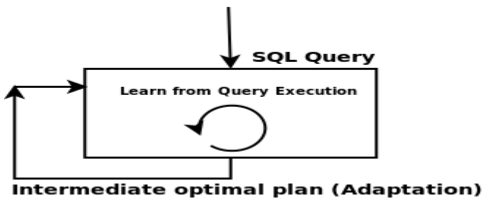
SQL Query
Learn from Query Execution
Intermediate optimal plan (Adaptation)

Figure 1: Adaptive parallel plan generation as a black box view.

and m2 when $m <= n$ are $m1 = 1, m2 = 1, 2...(n-1)$, and $m1 = 2, m2 = 1, 2...(n-2)$ continuing upto $m1 = n-1, m2 = 1$. Hence, the total number of partitioning options are $\frac{(n)(n-1)}{2}$. Each partitioning option represents one parallel plan. Hence, the total parallel plan space size is $\frac{(n)(n-1)}{2}$.

One of the challenges is to enumerate the parallel plan space in the most efficient manner in order to search for an optimal parallel plan. Enumeration of the plan space should be coupled with a learning mechanism, so that a future search could be improved based on the feedback from the already explored plan space. A good learning system should be able to handle issues such as a slow rate of learning and a long time for convergence. The ability to detect the global minimum execution time from many local minima influences the rate of learning. The rate of learning influences the ability of the system to converge as the longer the learning time, the longer is the time for convergence to find an optimal parallel plan.

Progress: We have developed a basic prototype infrastructure for *adaptive parallelization* in MonetDB. Each successive query invocation generates a new parallel plan. The system maintains a history of the execution plans. We have also developed a convergence algorithm that guarantees convergence in minimal runs while detecting a global minimum execution. The convergence algorithm guarantees convergence in different scenarios such as variations in the operator's execution time due to the operating system noise. We use microbenchmarks to study the feedback based behaviour of the individual operators such as select and join. We also experiment with simple queries from TPC-H benchmark with plans having select and join operators as prominent expensive operators.