CS524-Project Proposal

BACKGROUND INFORMATION

Quickstep is a relational database engine designed to efficiently leverage contemporary hardware aspects such as large main memory, multi-core, and multi-socket server settings. Query execution in quickstep is controlled by a cost-based query optimizer which produces the query execution plan. Figure 1 shows a SQL query and the corresponding query plan produced in Quickstep.

The optimized query plan is a Directed Acyclic Graph (DAG), where each node in the DAG corresponds to a relational operator primitive (e.g. Build Hash, Probe Hash, Print Output , Drop Output). Each operator takes some blocks of memory as input. The memory input to an operator/node (M) is the product of the number of data tuples input to the operator and the maximum size of the tuples.The operators then produce output memory blocks.

These output blocks are then consumed by the designated consumer operators in the DAG. The whole process repeats until all the operators in the DAG have executed and produced the final result and the final Drop Output operator is executed. An operator can be dependent on multiple operators to complete before it can be scheduled.These output blocks are then consumed by the designated consumer operators in the DAG. The whole process repeats until all the operators in the DAG have executed and produced the final result and the final Drop Output operator is executed. An operator can be dependent on multiple operators to complete before it can be scheduled.

The edges in the DAG represent the dependencies between the operators. The edge between Probe Hash and Drop hash operators is pipelining because the Drop Hash operator cannot be scheduled before all the work orders of the Probe Hash operator are complete. Similarly, the edge between Probe Hash and Print Output is non-pipeline breaking because the work orders of Print Output can be scheduled in parallel with the work orders from Probe Hash. At any time T, the operators in the DAG whose dependencies have been meet are considered available for scheduling. The scheduler schedules work orders from the available operators.

For a given query, each operator has a selectivity factor and a memory requirement associated with it. The selectivity factor is the fraction of input tuples that appear in the output of the operator.The selectivity of an operator can be computed from the data. The memory requirement for each operator can be calculated using the formula: selectivity factor * M, where M is the memory input for a given block. The memory requirement of an operator is uniformly distributed across it's work orders.Thus at an any time t, M_t units of memory are consumed for query execution. The overall memory footprint (MF) for a query is the sum of memory units consumed for query execution, over the entire query execution time, i.e. $MF = \sum_t M_t$

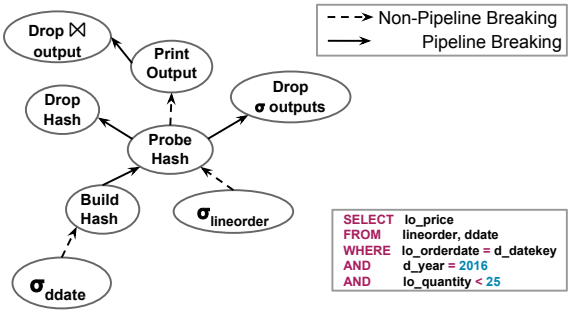

Figure 1. Example of a query plan produced in Quickstep

Once an operator completes, the memory consumed by the operator is made available for use by the other operators.

PROBLEM STATEMENT

Given a query plan DAG in quickstep, find an optimal ordering of scheduling work orders, such that the memory footprint is minimized over the entire query execution time.

SIGNIFICANCE

Identifying the optimal ordering of work orders/operators for queries is of significance to database researchers, as it helps in understanding the tradeoffs between query execution times and the available memory for query execution.

DATA COLLECTION

I plan to collect data regarding the execution times of the different operators, their associated work orders and their memory requirements by executing queries over a quickstep database instance. The queries which I will use to collect data are from the Star Schema Benchmark(SSB) [1]. There are 12 queries in the SSB benchmark that can be placed into 4 buckets based on the query plans they generate. I plan to use one query from each bucket to collect data on memory requirements and execution times.

The execution times of the work orders/operators will be measured on a quickstep instance utilizing only a single CPU core. This measurement would give me the upper bound on the running time of a query for multi-core systems, i.e. a query running in a multi-core ssystem must be atleast as fast as running on a single-core system.

REFERENCES

1. Patrick E ONeil, Elizabeth J ONeil, and Xuedong Chen. 2007. The star schema benchmark (SSB). *Pat* (2007).