CS524-Project Report

Rogers Jeffrey Leo John
University of Wisconsin-Madison
Madison,WI
rl@cs.wisc.edu

INTRODUCTION

The problem of scheduling has been extensively studied in the context of a job shop. Job shop scheduling tries to find an order in which the jobs have to be scheduled on different machines in a plant, taking into account the precedence relationships between jobs and the capacity constraints and tries to minimize the overall time to complete all the jobs. In this paper we examine scheduling in the context of a main memory database system. In particular, we investigate Quickstep, an open source relational database engine designed to efficiently leverage contemporary hardware aspects such as large main memory.

The work for executing a query in quickstep, can be broken down into of a set of logical units called operators. Each operators is divided into a set of work orders which are the actual units of query execution. We attempt to formulate a model that minimizes the memory utilization of the Quickstep database system during query execution, by finding an optimal execution order of the work orders. We limit our study to scheduling work orders on a single core.

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). Operators can be considered as logical units of work. The operators are mapped into work-orders which are the physical units of work. We present the entire discussion on background information in the context of operators which can be easily extended to work orders.

Each operator takes some blocks of memory as input. 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. 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 pipeline breaking or blocking 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. The scheduling of work orders is *non-premptive* For a given query, each operator has a *selectivity*

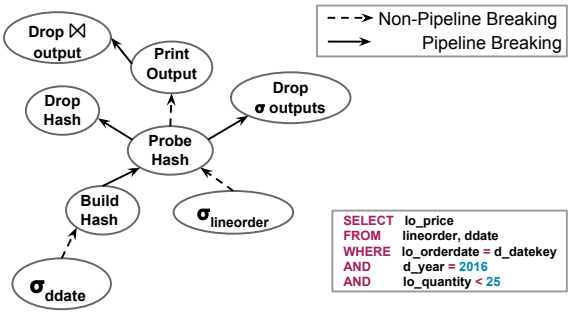

Figure 1. Example of a query plan produced in Quickstep

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. $MF = \sum_t M_t$ 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

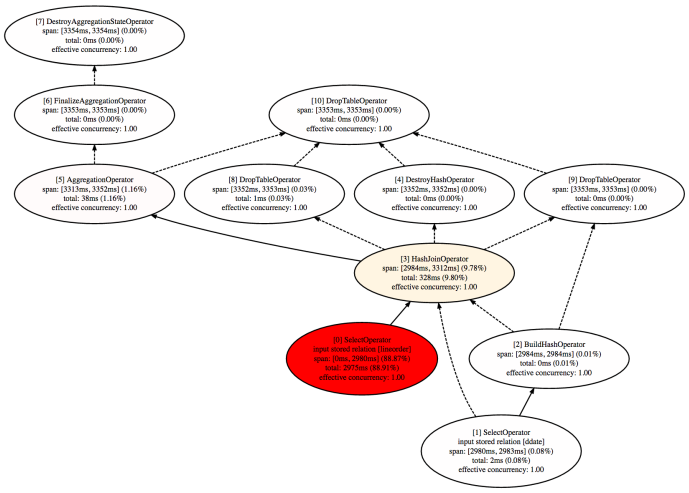

Figure 2. Execution plan for SSB Q-1.1

in understanding the tradeoffs between query execution times and the available memory for query execution.

DATA COLLECTION

We collected 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. Particularly, we collected data for query 1.1 from the Star Schema Benchmark(SSB) [5]. The execution times of the work orders/operators were measured on a quickstep instance utilizing only a single CPU core.

RELATED WORK

The problem of scheduling has been extensively studied in the context of a job shop. Most of the problem formulations attempt to optimize for time. [2] consider a machine scheduling problem where the objective function is to minimize the makespan. [6] describe a rank based model for machine scheduling. Similarly, [3, 1, 4] propose an integer programming formulation to model non-preemptive scheduling of independent jobs for minimizing the weighted tardiness of the job. Our approach is similar to in that we present a discrete time integer program formulation for the problem. However, our objective involves optimizing for memory(resources) as well, instead of just the time.

MODEL FORMULATION

We present a discrete time integer programming formulation to find the optimal ordering of scheduling the work orders in the quickstep database system. We first define the following variables:

o set of all operators

wo set of all work orders

T Total time for executing the query (Time horizon)

M represents the mapping between work orders and the operator

$PRED$ precedence relationships, indicating that work order i has to complete before work order j

$PIPE$ pipelining relationships, indicating that work order i has to complete before work order j

s_i start time of the work order i

e_i end time of the work order i

d_i duration of the work order i

Mem_i memory consumed by the work order i

$\alpha_{i,k}$ binary variable equal to one if work order i was scheduled in time k

$\omega_{i,k}$ binary variable equal to one if work order i ended at time k

$a_{i,k}$ binary variable equal to one if work order i was active at time k

$m_{i,k}$ binary variable equal to one if the memory needed by work order i was in use at time k

We assume that the the work orders are being scheduled on a single core, i.e only one work order can be scheduled at a time. This is expressed by the equation:

$$\sum_{i \in wo} \alpha_{i,k} = 1 \quad \forall k = 1 \dots T \tag{1}$$

A work order has to be active over the entire duration of its execution. Therefore, the following equation holds by definition:

$$a_{i,k} = a_{i,k-1} + \alpha_{i,k} - \omega_{i,k} \quad \forall i \in wo, k = 1 \dots T \tag{2}$$

Further, since this is a single core system, new work orders cannot be scheduled/processed as long as the core is busy with an existing work order. The following equation expresses this.

$$\sum_{i \in wo} a_{i,k} = 1 \quad \forall k = 0 \dots T \tag{3}$$

A job, once completed cannot be scheduled again. This constraint is expressed by the following equation.

$$\sum_{k=1}^{T} \alpha_{w,k} = 1 \quad \forall w \in wo \tag{4}$$

As described in the background information, precedence relationships exist among operators and these precedence relationships can be blocking or non-blocking(pipelining). If an operator B is blocking on operator A, then all the work orders of operator A should be scheduled before the work orders of operator B. We express the blocking precedence relationships between the operators through the following constraint:

$$s_{w_B} > s_{w_A} + d_{w_A} \quad \forall (w_A, w_B) \in PRED \tag{5}$$

We also describe a second type of precedence called pipelining, wherein two operators A,B who are pipelined can have their work orders scheduled in an interleaved fashion. Let us consider two operators A and B whose work orders are w_{A1}, w_{A2}, w_{A3} and w_{B1}, w_{B2} respectively. A valid schedule could be

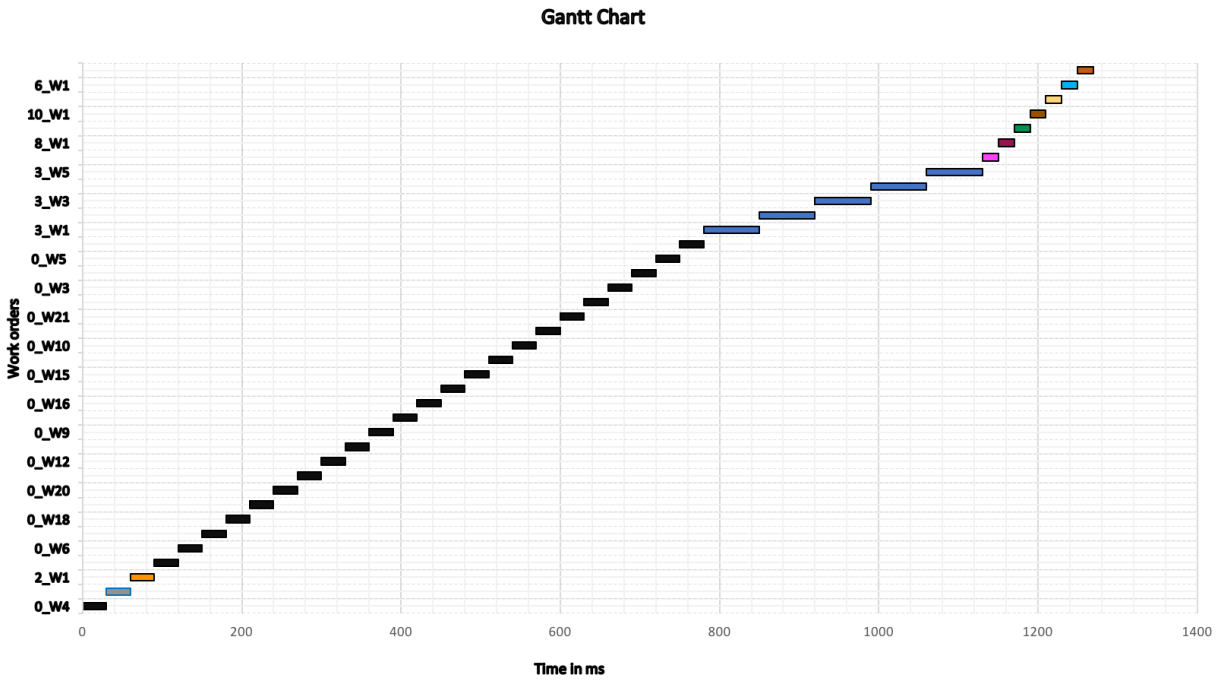

Figure 3. Gantt chart showing the execution order if the operators. Duration of each work order has been rounded off to the nearest milli second

w_{A1}, w_{B1}, w_{A2}, w_{A3} , w_{B2}. The following equation expresses this pipelining relationship:

$$\sum_{i,k} \omega_{i,k} \mid i, A \in M, k <= T_t$$
$$<= \sum_{i,k} \omega_{j,k} \mid j, B \in M, k <= T_t$$
$$\forall (A,B) \in PIPE \quad , T_k = 1 \ldots T \quad A \in o, B \in o \quad (6)$$

The equation enforces that if operator A is pipelined to B, then the number of work orders of operator A that are complete should be greater than or equal to the number of work orders completed to operator B. However, this does not eliminate invalid schedules like w_{A1}, w_{B1}, w_{B2}, w_{A2} , w_{A3}, where there are no outputs produced to consume the data produced by work orders w_{A2} and w_{A3}. To avoid this we introduce a new constraint,

$$\sum_{i,k} k * \omega_{i,k} \mid i, A \in M, <= \sum_{j,k} k * \omega_{j,k} \mid j, b \in M, k = 1 \ldots T \quad (7)$$

This constraint enforces that, the end time of the operator B (which is at the end of the pipeline), should be greater than the end time of the operator A(which is at the beginning of the pipeline).

Finally, we describe our objective equation. The objective equation consist of two parts.

$$\sum_{i \in wo} Mem_i * (T - s_i) + \lambda \sum_{i \in wo} s_i \quad (8)$$

The first part of the objective equation calculates the total memory occupied by every work order during the query equation. Recall that, though a work order completes it's memory is not released until a DropTable work order is invoked. Thus the memory stays active until it is destroyed. But our model does not have a notion of destroy work orders built into it. We handle such destroy work orders by specifying in the data that work orders belonging to an operator of type destroy have negative memory.

Therefore, to calculate the memory consumption, we assume that the work order occupies memory through the entire duration query execution. Thus, for the work orders who have positive memory requirements and negative memory requirements , they end up occupying memory right from the instant they were scheduled until the entire query execution completes. The net effect of this formulation is that, a work order's memory is counted until the time a corresponding destroy work order gets scheduled.

The second part of the of the objective equation minimizes the total start time over all the work orders. The parameter lambda can be used to control the tradeoff between minimizing for time and memory.

RESULTS

We present our results of the model for Query 1.1 of the SSB Benchmark:

```
select sum(lo_extendedprice*lo_discount)
    as revenue
from lineorder, ddate
```

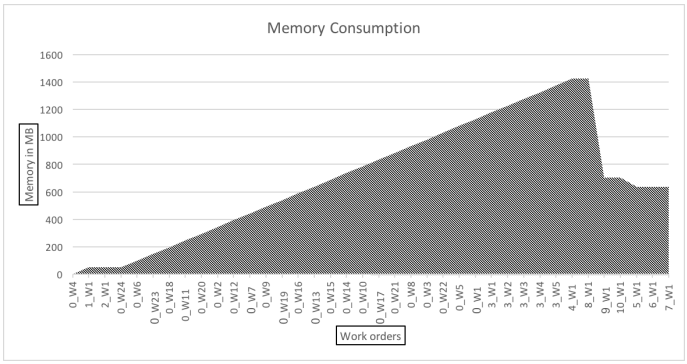

Figure 4. Memory Consumption over the query execution time.

```
where lo_orderdate = d_datekey
                and d_year = 1993
                and lo_discount between
                    1 and 3
                and lo_quantity < 25;
```

The query involves a join between the fact table lineorder and the dimension table ddate followed by an aggregation on the product of the discount and the price attributes of the lineorder table. The query also involves filtering of the ddate table based in the d_year attribute and the lineorder table based on the lo_discount and lo_quantity attributes. Figure 2 shows the execution DAG for Query 1.1. There are 10 operators in the DAG corresponding to the selection filtering , aggregation operations. There are also destroy operators free the memory occupied by the various operators over the query execution. The arrows between the operators indicate the precedence relationships between the operators.

Figure 7 presents the ordering of the work orders obtained as an output during the solve of the model. It can be seen that work orders from 0, 1 and 2 as interleaved as they do not have precedence relationships. However, since 1 and 2 have a pipelining relationship associated with them (as seen from Figure 2), they are scheduled in sequence. Also, since 3 is blocked on 1, all work orders of 3 are scheduled only after all work orders of 1 are scheduled.

Similarly the schedule respects the precedence for the other operators. Work orders of operators 4, 9 and 2 which are *DropTable* operators , are scheduled immediately after work orders of 3 are scheduled. These operators correspond to releasing the memory used by operators 0, 1 and 3 respectively. Although, operators 5 and 6 are available for scheduling, the schedule gives priorities to the destroy operators that free memory, there by reducing the amount of memory that is in use. We can therefore see that the scheduler presented by the model optimizes for the memory usage.

Figure 4 shows the memory consumed by the work orders during query execution. The figure has been simplified to show the memory in use only when a work order is scheduled. The drop in memory corresponds to the DropTable work orders being scheduled.

LIMITATIONS

Representing time as a discrete unit has a limitation of causing the model to blow up for really huge data sets. This is mainly because increasing the size of the data, increases the number of work orders produced thereby increasing the overall processing time of the query.

FUTURE WORK

An important aspect of this work, is to extend the model to multi-core settings. The current design of the model lends itself to be easily extended to a multi core setting. We would also like to address the limitations of the current model by extending and tuning it to scale for query execution on several Gigabytes of data. It would also be interesting to compare the results with the current execution ordering in the Quickstep main memory database system.

REFERENCES

1. Debra J Hoitomt, Peter B Luh, Eric Max, and Krishna R Pattipati. 1990. Scheduling jobs with simple precedence constraints on parallel machines. *IEEE Control Systems Magazine* 10, 2 (1990), 34–40.
2. Jan Karel Lenstra and AHG Rinnooy Kan. 1978. Complexity of scheduling under precedence constraints. *Operations Research* 26, 1 (1978), 22–35.
3. Peter B Luh, Debra J Hoitomt, Eric Max, and Krishna R Pattipati. 1988. Parallel machine scheduling using Lagrangian relaxation. In *Computer Integrated Manufacturing, 1988., International Conference on*. IEEE, 244–248.
4. Peter B Luh, Debra J Hoitomt, Eric Max, and Krishna R Pattipati. 1990. Schedule generation and reconfiguration for parallel machines. *IEEE Transactions on Robotics and Automation* 6, 6 (1990), 687–696.
5. Patrick E ONeil, Elizabeth J ONeil, and Xuedong Chen. 2007. The star schema benchmark (SSB). *Pat* (2007).
6. Harvey M Wagner. 1959. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly* 6, 2 (1959), 131–140.