An Adaptive Query Execution System for Data Integration

Paper Authors: Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy and Daniel Weld

Published: SIGMOD Conference 1999

Presented in CPSC 504 2009 by Noreen Kamal

Outline

- Motivation for creating Tukwila – an adaptive query execution system for data integration
- Architecture of Tukwila
- Interleaving Planning and Execution
- Adaptive Operators
  - Dynamic Collectors
  - Double pipelined join
- Concluding Remarks

Why do we need data integration?

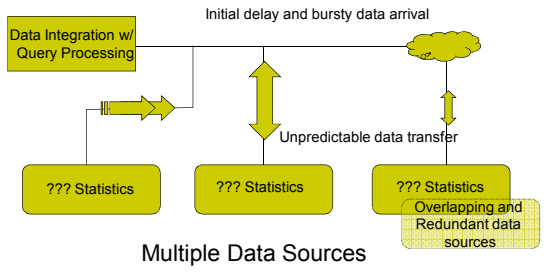
- Goal is to provide a *uniform* query interface to a magnitude of data sources
- The information is distributed over a "collection of internal autonomous sources" and/or over the World Wide Web
  - (intranet or internet)
- Advantages to data integration:
  - Users don't have to locate each data source
  - Interaction with each data source can occur independently
  - Ability to manually combine data from different sources

What are the key challenges to data integration?

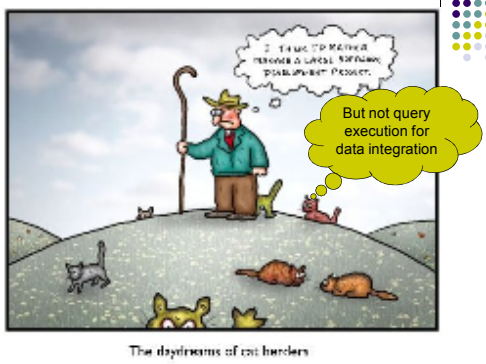
1. Query reformulation — FOCUS OF PREVIOUS RESEARCH
2. Construction of wrapper programs — FOCUS OF PREVIOUS RESEARCH
3. Design of new query processing techniques — HAS HAD LITTLE ATTENTION AND CREATED BOTTLENECKS. FOCUS OF PAPER AND TUKWILA

Most previous research in data integration focused on integration of web-based sources with small amount of data returned from each source

What is the need for adaptive query execution?



Conceptual diagram



The daydreams of cat herders

Discussion 1

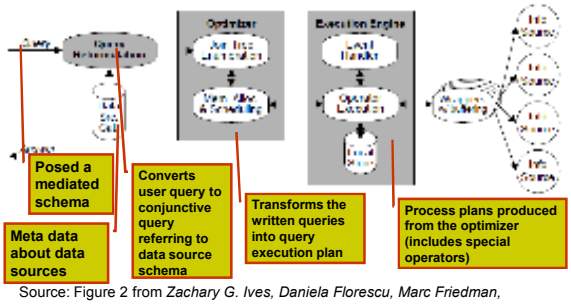
For one of the following motivating situations of Tukwila
1) Absence of statistics
2) Unpredictable data arrival characteristics
3) Overlap and redundancy among sources
4) Optimizing the time to initial answers

- Q1: Do you think the above four motivations matter a lot in the context of Tukwila? Why or Why not?
- Q1: If you are a member of Tukwila team, what rules or policy do you think are more important among the above four?
- Q2: What kind of other features that query processing in data integration may have, when it occurs over network-bound, autonomous data sources.

Tukwila: adaptive query execution

- Allows or accounts for *BETWEEN* optimizer and execution engine through **interleaved planning and execution**
  - Allows a partial plan to be sent from the optimizer to the execution engine (and later decide how to proceed after the partial plan is completed)
  - Or sends a complete plan but the execution engine may check condition that require incremental reoptimization
- *WITHIN* the execution engine with **adaptive operators**
  - Double pipline hash join – Returns initial results quickly
  - Collector operator – manages data from large set of possibly overlapping or redundant data
  - Plan can contain conditional nodes

Tukwila Architecture



Source: Figure 2 from *Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Y. Levy, Daniel S. Weld: An Adaptive Query Execution System for Data Integration. SIGMOD Conference 1999: 299-310*

Interleaving Planning and Execution – Novel aspect of Tukwila optimizer

- Doesn't have to create a complete execution plan if essential statistics are missing (optimizer generates partial plan)
- Produces annotated operator tree and generates appropriate event condition action rules
- Query optimizer conserves state of its search space, so it is able to resume optimization in an incremental fashion

Interleaving Planning and Execution – Query Plans

- Operators are organized as *fragments* (pipeline units):
  - At end of each, results are materialized and rest of plan can be re-optimized or rescheduled
  - Plan consists of **partially ordered set of fragments** and **set of global rules**
- Fragments consist of fully pipelined tree of physical operators and a set of local rules (each node is a physical operator)

Interleaving Planning and Execution – Query Plans (con't)

- **RULES** are how several kinds of adaptive behavior is implemented in Tukwila
  - **Re-optimization** (at end of fragment),
  - **Contingent planning** (check properties of result to choose next fragment),
  - **Adaptive operators** (policy for memory overflow resolution in double pipeline join and also for collectors),
  - **Rescheduling**

Interleaving Planning and Execution – Query Plans (con't)

- **Structure:** *when* ***event*** *if* ***condition*** *then* ***action*** but includes: **owner** and **action flag**
  - Events can include: open, close (fragments/operators); error, timeout, out of memory etc
  - Dynamic qualities: state(), Card(), Time(), Memory()
  - Action examples: alter memory or re-optimize, return an error etc.

Discussion 2

- One key aspect of Tukwila is that it can produce partial plans if there are not enough statistics. We know that traditional systems have trouble finding statistics as well.
- Would using a partial plan and then re-evaluating during execution be a good idea in traditional systems? If yes, why do you think traditional systems don't do this? If no, why isn't it a good idea?

Interleaving Planning and Execution – Query Execution

- Executes a query plan (as expected)
- Gathers statistics about each operation
- Handles exception conditions or re-invokes the optimizer

Interleaving Planning and Execution – Event Handling

- Interprets rules attached to the query execution plan
- Execution engine may generate events
  - Which go into an event queue
- For each event the event handler uses a hash table to find all matching rules in the active set
- The active rule, it evaluates the condition and if it is satisfied, the rule is executed and moves to the next event

Adaptive Query Operators – Dynamic Collectors

- Handles query over large overlapping data such as *movie reviews, product information, bibliographic information, sites that are deliberately mirrored*
- Differs from Unions as Collector Operators can contact only some sources
- The optimizer has estimates of overlap relations between data sources
- The collector can provide guidance on which data source should be access through programmed *policy*
  - Implemented using normal rule-execution

Adaptive Query Operators - Double pipelined join

- Issues with traditional Hash Joins with data integration systems
  - Optimizer may not know the relative size of each relation (as typically the smaller relation is the inner)
  - Important is the time to 1st tuple, so it maybe advantageous to use the larger relation as the inner if it sends data faster
  - Time to 1st tuple is extended due the hash join's non-pipelined behavior

Adaptive Query Operators - Double pipelined hash join

- Double pipelined hash join is symmetric and incremental
- Produces tuples as quickly as possible AND masks slow data source transmission
- But it must hold BOTH relations in memory
- Double pipelined join is data driven: each join relation sends tuple through the join operator as fast as possible

Adaptive Query Operators - Double pipelined hash join

- 2 problems exist:
  - Follow data driven, bottom-up execution and Tukwila is top down, iterator bases
    - So it uses multi-threading: join consists of separate threads for output, left child and right child
    - And employs a small tuple transfer queue
  - Enough memory is required to hold both the relations

Adaptive Query Operators - Double pipelined hash join

- Handling memory overflows in Tukwila:
  - Incremental left flush: switch to strategy of reading tuples from right side relations and as necessary flush a bucket from the left side relation's hash table → gradually degrade to a Hybrid Hash (and so on…)
  - Incremental symmetric flush: pick a bucket to flush from disk and flush the bucket from both sources
- Incremental left flush will perform fewer disk I/Os
- Incremental symmetric flush may have reduced latency as both relations are processed in parallel

Discussion 3

When this paper was written (10 years ago), perhaps it was okay to claim that 'the sizes of most data integration queries are expected to be only moderately large'. But does this hold today, especially with the coming era of 'cloud computing'? Specifically:

1. Do double pipelined hash joins seem effient enough for today's data?
2. Would you use double pipelined hash joins in non-data integration applications?
3. does it seem like it would work better for the double pipelined hash join for when there is a continuous and steady flow of data? However, networked data can have sudden bursts of large about of data - do you think that the double-pipelined hash join is suited for this bursty environment?

Concluding Remarks

- Data Integration poses many challenges through lack of knowledge about the data source and transmission rates
- Dynamic query execution is successful in handling these challenges
- Interleaving planning and optimization is possible through fragments and rules
- Adaptive Query operators in the execution engine was provided through the dynamic collectors and double pipelined hash join