An Adaptive Query Execution Engine for Data Integration

Zachary Ives, Daniela Florescu, Marc Friedman, Alon Levy, Daniel S. Weld

University of Washington

Slides by Peng Li, Modified by Rachel Pottinger

Presentation: Rachel Pottinger

Discussion: Jason Hall

Outline

- Tukwila Architecture
- Interleaving of planning and execution
- Adaptive Query Operators
  Collector & Double Pipelined Join
- Performance

The main challenges of the design of DISs:

- Query Reformulation
- The construction of wrapper programs
- Query optimizers and efficient query execution engines

Motivations:

- Little information for cost estimates
- Unpredictable data transfer rates
- Unreliable, overlapping sources
- Want initial results quickly
- Network bandwidth generally constrains the data sources to be “small”

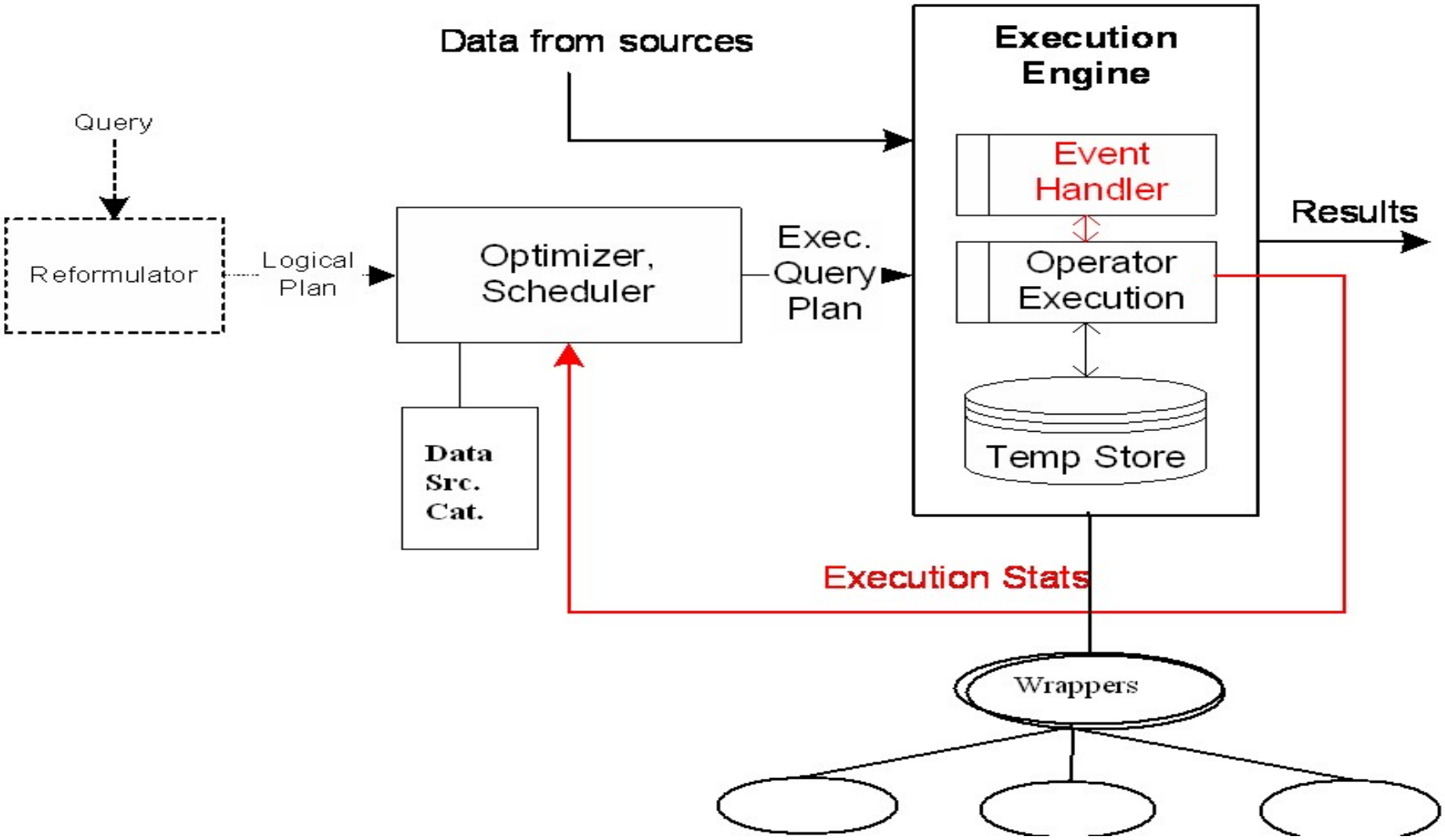
System needs to be adaptive

Discussion in Pairs

Tukwila and its double-pipelined hash join emphasize early data return over faster and complete data return. Why do we “want initial results quickly?”

- Why would this be important with data integration
- Where else could it be beneficial?

Tukwila Architecture



Novel Features of Tukwila

- Interleaving of planning and execution
  - Compensates for lack of information
- Handle event-condition-action *rules*
  - When and how to modify the implementation of certain operators at runtime if needed.
  - Detect opportunities for re-optimization.
- Manages overlapping data sources (*collectors*)
- Tolerant of latency (*double-pipelined join*)
  - Returns initial results quickly

Discussion in Pairs

Credit to Carol & Nalin

Tukwila manages overlapping data sources, without really explaining why it matters.

- What problems could overlapping data introduce?
- What could be some potential ways to handle it?

Interleaving of planning and execution

Novel characteristics of Tukwila:

- The optimizer can create a partial plan if essential statistics are missing or uncertain
- The optimizer generates both operator trees *and* appropriate event-condition-action rules.
- Optimizer conserves the state of its search space when it calls the execution engine.

Overview of the query plan structure

- A plan includes a partially-ordered set of fragments and a set of global rules
- A fragment consists of a fully pipelined tree of physical operators and a set of local rules.
- The fragment is the key mechanism for implementing the adaptive property: at the end of each fragment, the rest of the plan can be re-optimized or rescheduled

Rules

- Re-optimization

  The optimizer's cardinality estimate for the fragment's result is significantly different from the actual size → re-invoke optimizer

- Contingent planning

  The execution engine checks properties of the result to select the next fragment

- Rescheduling

  Reschedule if a source times out

- Adaptive operators

Rule format

When *event* if *condition* then *actions*

When *closed(frag1)*

if *card(join1)>2*est_card(join1)*

then *replan*

An event triggers a rule, coursing it to check its condition. If the condition is true, the rule fires, executing the action(s).

Group Discussion

- For one of the following motivating situations of Tukwila
  - Absence of statistics
  - Unpredictable data arrival characteristics
  - Overlap and redundancy among sources
  - Optimizing the time to initial answers
- Q1: **Can you give some examples where the chosen topic matters?**
- Q2: **If you are a member of Tukwila team, what rules or policy would you have to deal with the problem?**
  - To help discussion, more specific situations will be given
  - But you may assume any problem or situation
- Discussion
  - Form 8 groups (3~4 person per group, two teams per topic)
  - Discuss Q1 and Q2 for one topic (5 ~ 7 minutes)

Examples

Orders

OrderNo	TrackNo
1234	01-23-45
1235	02-90-85
1399	02-90-85
1500	03-99-10

UPS

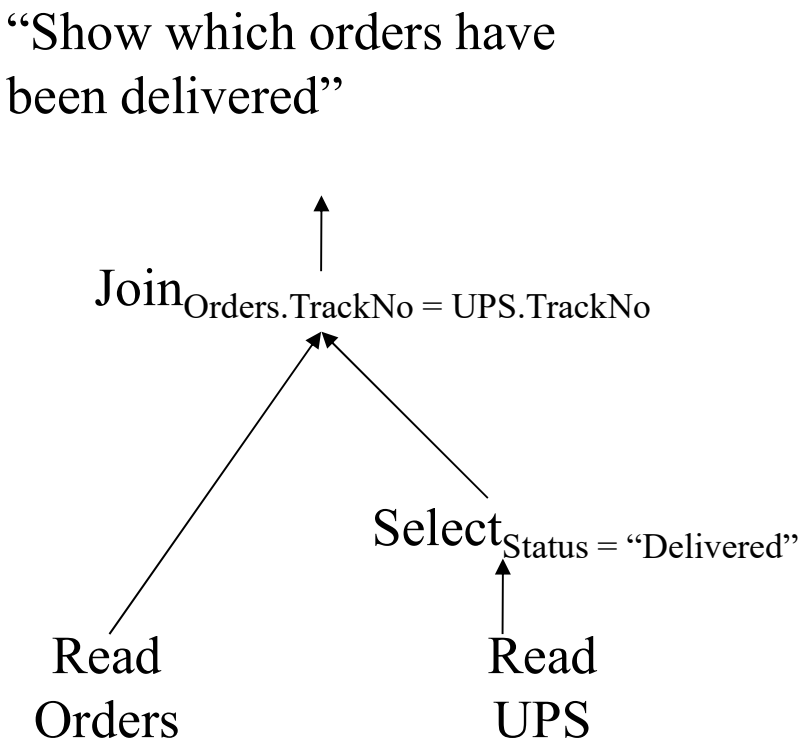
TrackNo	Status
01-23-45	In Transit
02-90-85	Delivered
03-99-10	Delivered
04-08-30	Undeliverable

$Join_{Orders.TrackNo = UPS.TrackNo}$ (Orders, UPS)

OrderNo	TrackNo	Status
1234	01-23-45	In Transit
1235	02-90-85	Delivered
1399	02-90-85	Delivered
1500	03-99-10	Delivered

Query Plan Execution

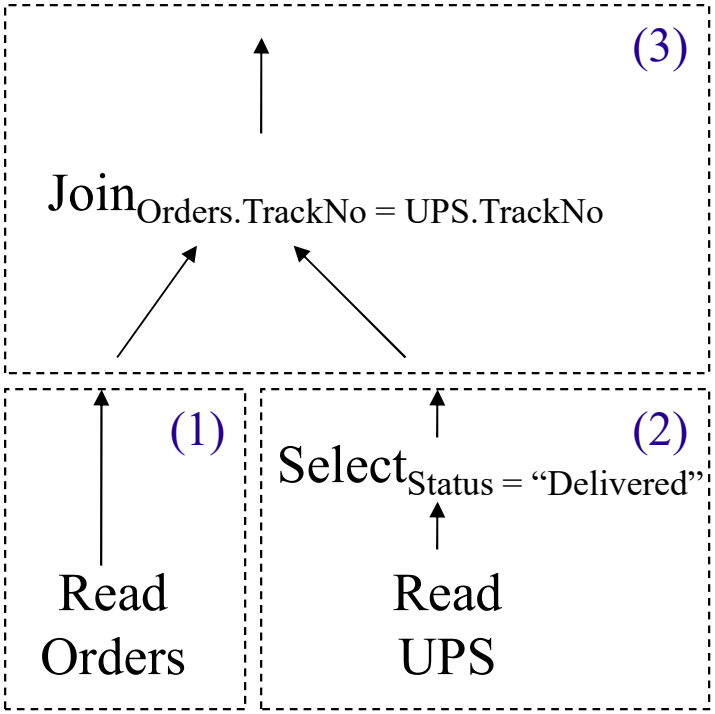
Query plan represented as data-flow tree:



- Control flow
  - Iterator (top-down)
    - Most common database model
    - Easier to implement
  - Data-driven (bottom-up)
    - Threads or external scheduling
    - Better concurrency

Tukwila Plans & Execution

- Multiple *fragments* ending at materialization points
- Rules triggered by events
  - Re-optimize remainder if necessary
  - Return statistics



When(closed(1)):
if size_of(Orders) > 1000
then reoptimize {2, 3}

Adaptive Query Operators
Double Pipelined Join

Conventional Joins

- Sort merge joins &indexed joins
  ---can not be pipelined
- Nested loops joins and hash joins
  ---Follow an asymmetric execution model

For Nested loops joins, we must wait for the entire inner table to be transmitted initially before pipelining begins

For hash joins, we must load the entire inner relation into a hash table before we can pipeline.

Double Pipelined Hash Join

- Proposed for parallel main-memory databases (Wilschut 1990)
  - Hash table per source
  - As a tuple comes in, add to hash table and probe opposite table
- Evaluation:
  - Results as soon as tuples received
  - Symmetric
  - Requires memory for two hash tables
- But data-driven!

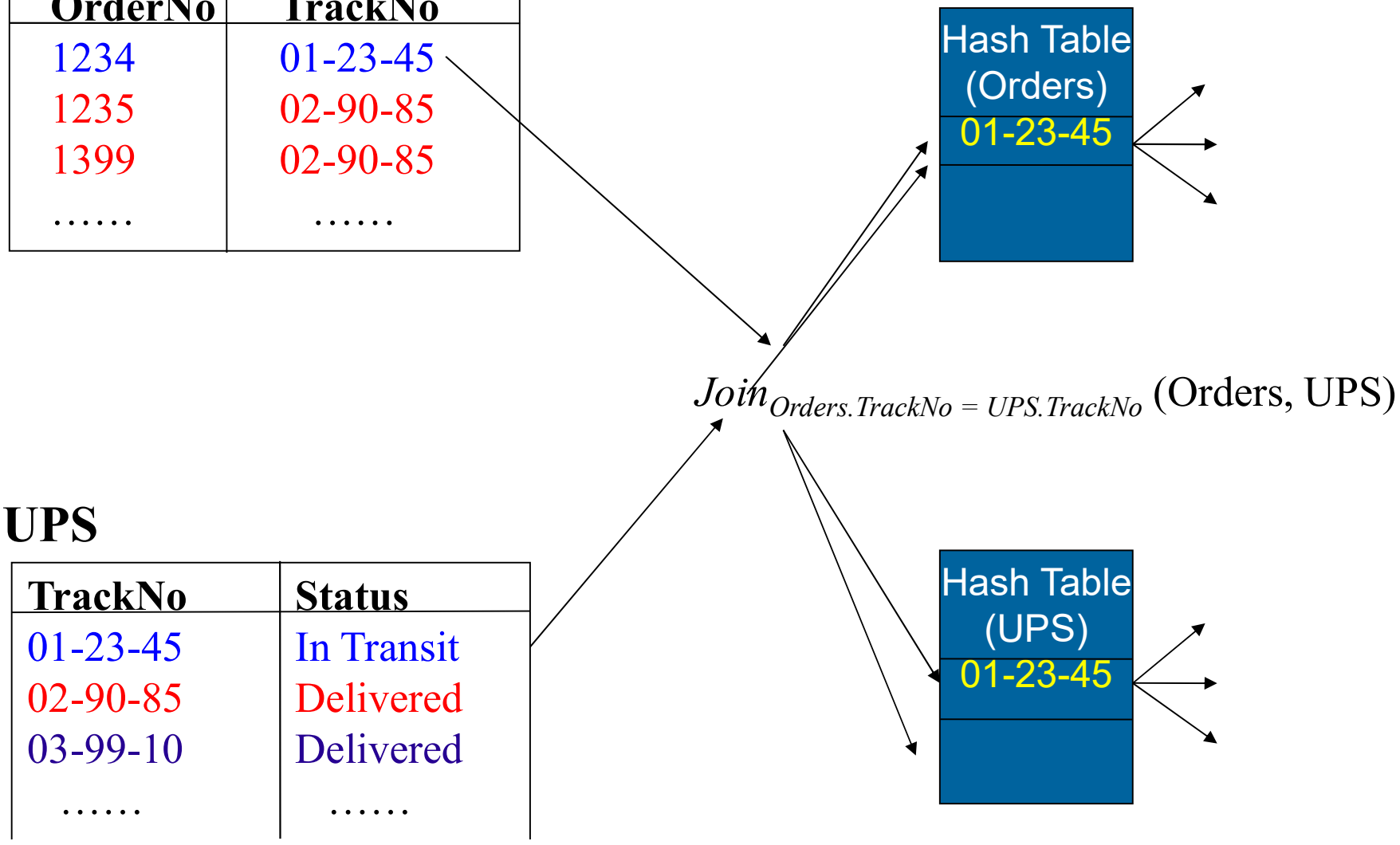
Example

Orders

OrderNo	TrackNo
1234	01-23-45
1235	02-90-85
1399	02-90-85
……	……

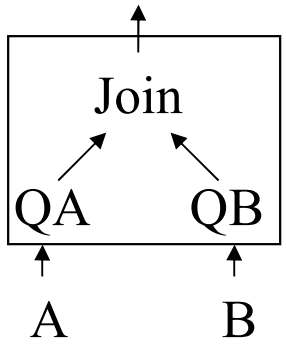
UPS

TrackNo	Status
01-23-45	In Transit
02-90-85	Delivered
03-99-10	Delivered
……	……



Double-Pipelined Join Adapted to Iterator Model

- Use multiple threads with queues
  - Each child (A or B) reads tuples until full, then sleeps & awakens parent
  - Join sleeps until awakened, then:
    - Joins tuples from QA or QB, returning all matches as output
    - Wakes owner of queue



Insufficient Memory?

- May not be able to fit hash tables in RAM
- Strategy for standard hash join
  - Swap some buckets to *overflow files*
  - As new tuples arrive for those buckets, write to files
  - After current phase, clear memory, repeat join on overflow files

Conclusions

- General Tukwila architecture
- Non-conventional characters of Tukwila
- Interleaving of optimization and execution
- Double pipelined hash join

Group Discussion

Groups of 3-4

Credit to Ehsan

- Would the adaptive behaviour of Tukwila be beneficial in general database systems?
- Would it boost efficiency?
- What could be some advantages and disadvantages of applying the same methods to general database systems?