Adaprive Query Processing: Eddies and Adaptive Joins

Instructor: Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

References


Slides modified from talks by J. Hellerstein and M. Franklin

Road Map

- Adaptive Query Processing: Setting
- Intra-join adaptivity
  - Synchronization Barriers
  - Moments of Symmetry
- Adaptive Join Algorithms
  - Ripple Joins
  - X-joins
- Eddies
  - Encapsulated, adaptive dataflow
- Conclusions
Querying in Volatile Environments

- Federated query processors a reality
  - Cohera, DataJoiner, RDBMSs
  - No control over stats, performance, administration
- Shared-Nothing Systems “Scaling Out”
  - E.g. NOW-Sort
  - No control over “system balance”
- User “CONTROL” of running queries
  - E.g. Online Aggregation
  - No control over user interaction
- Sensor Nets: the next killer app
  - E.g. “Smart Dust”
  - No control over anything!

Motivation

- Adaptivity in System R:
  Repeat:
  1. **Observe (model) environment**: daily/weekly (runstats)
  2. **Use observation to choose behavior** (optimizer)
  3. **Take action** (executor)
     - Adaptivity at a per-week frequency!
     - Not suited for volatile environments

- What happens if, during execution, a source is much slower than the other?

Toward Continuous Adaptivity

- What happens if, during execution, a source is much slower than the other?

- Need much more frequent adaptivity
  - Goal: *adapt per tuple of each relation*
  - The traditional runstats-optimize-execute loop is far too coarse-grained
  - So, continuously perform all 3 functions, at runtime
Goals towards adaptive joins

- Minimize Synchronization Barriers
- Maximize # of Moments of symmetry

Adaptable Joins, Issue 1

- Synchronization Barriers
  - One input frozen, waiting for the other
  - Can’t adapt while waiting for barrier!
  - So, favor joins that have:
    - no barriers
    - at worst, adaptable barriers

Adaptable Joins, Issue 2

- Moments of Symmetry
  - Suppose you can adapt an in-flight query plan
  - How would you do it?
  - Base case: reorder inputs of a single join
    - Nested loops join
Moments of Symmetry

- Suppose you can adapt an *in-flight* query plan
  - How would you do it?
- Base case: reorder inputs of a single join
  - Nested loops join
  - Cleaner if you wait till end of inner loop

Hybrid Hash
- Reorder while "building"?

Moments of Symmetry, cont.

- Can swap join inputs w/o state modification
- Nested Loops join: end of each inner loop
- Hybrid Hash join?
- Sort-Merge join?

- More frequent moments of symmetry → more frequent adaptivity
Ripple Joins: Prime for Adaptivity

- Ripple Joins
  - Pipelined hash join (a.k.a. hash ripple, Xjoin)
  - No synchronization barriers
  - Continuous symmetry
  - Good for equi-join
  - Simple (or block) ripple join
    - Synchronization barriers at "corners"
    - Moments of symmetry at "corners"
    - Good for non-equi-join
  - Index nested loops
    - Short barriers
    - No symmetry

- Note: Ripple corners are adaptable!
  - Accommodate barriers in simple/block ripple

Beyond Binary Joins

- Think of swapping "inners"
  - Can be done at a global moment of symmetry
- Example: how to reorder
  - R join S join T?
- Intuition: like an n-ary join
  - Except that each pair can be joined by a different algorithm!
- So...
  - Need to introduce n-ary joins to a traditional query engine

Continuous Adaptivity: Eddies

- A pipelining tuple-routing iterator (just like join or sort)
  - works well with ops that have frequent moments of symmetry
Continuous Adaptivity: Eddies

- Adjusts flow adaptively
- Tuples flow in different orders
- Visit each op once before output
- Naive routing policy:
  - All ops fetch from eddy as fast as possible
  - Previously-seen tuples precede new tuples

Back-Pressure

- SELECT * FROM R WHERE s1() AND s2()
- Selectivity is 50%
- Cost(s2) = 5. Vary cost of s1.
- Backpressure favors faster op!

Back-Pressure Not Enough!

- SELECT * FROM R WHERE s1() AND s2()
- Cost 5
- Selectivity(s2)=50%, vary selectivity of s1

Oops! Why does eddy not adapt?
An Aside: *n*-Arm Bandits

- A little machine learning problem:
  - Each arm pays off differently
  - Explore? Or Exploit?
    - Sometimes want to randomly choose an arm
    - Usually want to go with the best
    - If probabilities are stationary, dampen exploration over time

Eddies with Lottery Scheduling

- Operator gets 1 ticket when it takes a tuple
  - Favor operators that run fast (low cost)
- Operator loses a ticket when it returns a tuple
  - Favor operators with low selectivity
- Lottery Scheduling:
  - When two ops vie for the same tuple, hold lottery
  - Never let any operator go to zero tickets
    - Support occasional random "exploration"

Lottery-Based Eddy

- Two expensive selections, cost 5
- Selectivity(s2) = 50%. Vary selectivity of s1.

Why does lottery eddy work better?
In a Volatile Environment

- Two index joins (two external indices)
  - Slow: 5 second delay; Fast: no delay
  - Toggle after 30 seconds

XJoin

- XJoin is a new query operator that:
  - Produces results incrementally as they become available.
  - Allows progress to be made in highly erratic situations.
  - Has a small memory footprint.
  - Tolerates bursty and slow behavior.

Symmetric Hash Join

- Symmetric Hash Join (SHJ) blocks only if both stall.
- Processes tuples as they arrive from sources.
- Produces all tuples in the join and no duplicates.
Partitioning

- XJoin is a partitioned hash join method.
- When allocated memory is exhausted, a partition is flushed to disk.
- Join processing continues on memory-resident data.
- Disk-resident tuples are handled in background.

The 3 Stages of XJoin

- Stage 1 - Symmetric hash join (memory-to-memory)
- Stage 2 - Disk-to-memory
  - Separate thread - runs when stage 1 blocks.
  - Stage 1 and 2 trade off until all input has been received.
- Stage 3 - Clean up stage
  - Stage 1 misses pairs that were not in memory concurrently.
  - Stage 2 misses pairs when both are on disk, and may not get to run to completion.

Summary

- Eddies: Continuously Adaptive Dataflow
  - Suited for volatile performance environments
  - Changes in operator/machine performance
  - Changes in selectivities (e.g. with sorted inputs)
  - Changes in data delivery
  - Changes in user behavior (CONTROL, e.g. online agg)
  - Currently adapts join order
    - Competitive methods to adapt access & join methods?
- Requires well-behaved join algorithms
  - Pipelining
  - Avoid synch barriers
  - Frequent moments of symmetry