Optimizing streaming dataflow execution

Greg Barish
Fetch Technologies

Craig Knoblock
USC Information Sciences Institute
Outline

• Network query engines
  – Focus: optimizing streaming dataflow

• Speculative plan execution
  – Background + working example
  – The basic technique
  – Learning to predict plan execution

• Summary
Streaming dataflow model

- **Dataflow**
  - Operations scheduled by data availability
    - Independent operations execute in parallel
    - Maximizes *horizontal parallelism*
  - Dataflow computers [Dennis 1974] [Arvind 1978]
  - Example: computing \((a*b) + (c*d)\)

- **Streaming**
  - Operations emit data as soon as possible
    - Independent data processed in parallel
    - Maximizes *vertical parallelism*
  - Network query engines
    - [Ives et al. 1999] [Naughton et al. 2000] [Hellerstein et al. 2001]
Network query engines

• Tukwila, Telegraph, Niagara
  – Dataflow & pipelining similar to Theseus
  – Execution system with support for efficient query execution from remote data sources
  – Automatically generate query plans from XML queries
  – No support for loops, conditionals, or external interactions
  – Designed for querying only, not monitoring (except for NiagaraCQ)

• Like Theseus, each of these systems go beyond simply executing dataflow plans
  – They each have techniques for optimizing execution
Tukwila (Ives et al. 1999)

• Adaptive network query processing for XML data
  – Interleaved execution and optimization
  – Inter-operator adaptivity
  – Dynamic operator re-ordering based on events
    • Memory overflow, wrapper timeout

• Notable new operators
  – X-SCAN: Efficient querying of streaming XML docs
  – JOIN: Double pipelined hash (probe is LHS or RHS)
  – DYNAMIC COLLECTOR: Efficient unioning of sources
Tukwila – Interleaved Planning and Execution

- Generates initial plan
- Can generate partial plans and expand them later
- Uses rules to decide when to reoptimize

From Ives et al., SIGMOD’99

```
WHEN end_of_fragment(0)
IF card(result) > 100,000
THEN re-optimize
```
Tukwila – Adaptive Double Pipelined Hash Join

From Ives et al., SIGMOD’99

**Hybrid Hash Join**
- No output until inner read
- Asymmetric (inner vs. outer)

**Double Pipelined Hash Join**
- Outputs data immediately
- Symmetric
- More memory
Tukwila – Dynamic Collector Op

• Smart union operator

• Supports
  – Timeouts
  – slow sources
  – overlapping sources

From Ives et al., SIGMOD’99

```
WHEN timeout(CustReviews)
DO activate(NYTimes),
   activate(alt.books)
```
Niagara (Naughton, DeWitt, et al. 2000)

• Adaptive network query processing for XML data
  – Interleaved execution + document search
  – Supports streaming over blocking operators
    • Synchronization by re-evaluating operators or by propagating the differential result
Execution with partial results

[Shanmugasundaram et al. 2000]

• Niagara uses partial results to reduce the effects of blocking operators
  – Reduces blocking nature of aggregation or joins

• Basic idea
  – Execute future operators as data streams in, refine as slow operators catch up
    • Execution is driven by the availability of real data
    • Results are refined as additional data are processed
Approaches to Refining Results

• Re-evaluation
  – As new data becomes available, the operators re-output the results and the downstream operators are re-executed
  – Can be costly, but simple to implement

• Differential Algorithm
  – Each operator must support additions, deletes, and updates
  – Changed results must then be propagated to downstream operators
Telegraph (Hellerstein et al. 2000)

• Tuple-level adaptivity

• Rivers (optimize horizontal parallelism)
  – Adaptive dataflow on clusters (i.e., data partitioning)

• Eddies (optimize vertical parallelism)
  – Leverage commutative property of query operators to dynamically route tuples for processing
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

• Would like to reorder *in-flight* (pipelined) joins

• Base case: swap inputs to a join
  – What about per-input state?

• **Moment of symmetry:**
  – inputs can be swapped w/o state management

• E.g.
  – Nested Loops: at the end of each inner loop
  – Merge Join: any time*
  – Hybrid or Grace Hash: never!

• More frequent moments of symmetry
  \(\rightarrow\) 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
Beyond Binary Joins

• Think of swapping “inners”
  – Can be done at a global moment of symmetry

• 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
Telegraph – Beyond Reordering Joins

Eddy

• A pipelining tuple-routing iterator (just like join or sort)

• Adjusts flow adaptively
  – Tuples flow in different orders
  – Visit each op once before output

• Naïve routing policy:
  – All ops fetch from eddy as fast as possible
  – Previously-seen tuples precede new tuples

From Avnur & Hellerstein, SIGMOD 2000
Speculative plan execution

• Streaming dataflow optimization in Theseus
  – Motivation:
    • Great success of CPU speculative execution (Wall 1991)
    • Challenge of building smarter predictors than what is possible at the CPU level

• In this talk
  – An approach to speculative plan execution
    • Safe & fair
    • Yields arbitrary speedups
    • Algorithm for the automatic transformation of agent plans

  – An approach to value prediction
    • Combines caching, classification, and transduction
    • Better accuracy and space efficiency than strictly caching
The CarInfo agent

1. Locate cars that meet criteria
   - Edmunds.com

2. Filter out Oldsmobiles
The CarInfo agent

1. Locate cars that meet criteria
   - Edmunds.com

2. Filter out Oldsmobiles

3. Gather safety reviews for each
   - NHSTA.gov
The CarInfo agent

1. Locate cars that meet criteria
   - Edmunds.com

2. Filter out Oldsmobiles

3. Gather safety reviews for each
   - NHSTA.gov

4. Gather detailed reviews of each
   - ConsumerGuide.com
ConsumerGuide navigation

New Car Pricing & Reviews
2002 Dodge Stratus

Highlights for 2002

Stratus sedans share a design with the Chrysler Sebring sedan and convertible. Stratus coupes share a design with the Chrysler Sebring coupe.

Sedans come in SE, SXT, SE Plus, ES, and new R/T trim. The SXT and both SE versions come with a 4-cyl engine and offer an optional Chrysler-made 2.7-liter V6. The V6 is standard on the ES and R/T. All but the R/T have mandatory automatic transmission. All sedans have 4-wheel disc brakes, with ABS optional. Curtain side airbags are optional; no torso side airbags are offered. Added at midyear, the R/T sedan has antilock 4-wheel disc brakes, a 5-speed manual transmission, and offers at no extra charge Chrysler’s AutoStick automatic transmission with manual shift gate.

Coupes use powertrains and platforms from Mitsubishi’s Eclipse and Galant. They come in SE and R/T models. The SE has a 4-cyl engine or optional 3.0-liter V6. The V6 is standard on the R/T. Both coupes use manual transmission or optional automatic. R/T automatics come with traction control and can be ordered with AutoStick. Four-wheel disc brakes are included with the V6. Among coupes, ABS is optional only on the R/T.

Competition Perennial Best Buys Honda Accord and Toyota Camry continue to shine with refinement, model diversity, and comfort. Both come in coupe and sedan forms, offer economic 4-cylinder or sporty V6 power, have room for four adults, and are reasonably priced.
CarInfo Agent Plan

1. Get list of cars from Edmunds.com that meet specified criteria.
2. Remove any Oldsmobiles from that list.
3. Get the search results for each of those cars from NHTSA.gov, extracting the safety ratings.
4. Get the search results for each car at CG.com, extracting the link to the summary page.
5. Get the summary page for each car, extracting the link to the full review.
6. Get the full review page for each car, extracting the review itself.
Agent Execution Performance

• **Standard von Neumann model**
  – Execute one operation at a time
  – Each operation processes all of its input before output is used for next operation
  – **Assume**: 1000ms per I/O op, 100ms per CPU op

• **Execution time** = **13.4 sec**
Dataflow-style CarInfo agent plan

(Midsize coupe/hatchback, $4000 to $12000, 2002)

(((Dodge Stratus), (Pontiac Grand Am), (Mercury Cougar))

SELECT
maker != "Oldsmobile"

WRAPPER
Edmunds Search

WRAPPER
ConsumerGuide Search

WRAPPER
ConsumerGuide Summary

WRAPPER
ConsumerGuide Full Review

WRAPPER
NHTSA Search

JOIN

(search criteria)

((Oldsmobile Alero),
(Dodge Stratus),
(Pontiac Grand Am),
(Mercury Cougar))

(((http://cg.com/summ/20812.htm),
other summary review URLs)

(((http://cg.com/full/20812.htm),
other full review URLs)

((http://cg.com/full/20812.htm),
safety reports)

(car reviews)
Streaming dataflow executor

- **Thread pool architecture**
  - Enables bounded, dynamic parallelism

Example:

- **Plan operators** (e.g., Wrapper, Select, etc.)
- **Thread pool** (Midsize cpe/hatchbk, $4000 to $12000, 2002)
- **Plan output**
- **Select**
  - maker != "Oldsmobile"
- **Wrapper**
  - Edmunds Search
  - ((Oldsmobile Olero), (Dodge Stratus), (Pontiac Grand Am), (Mercury Cougar))
Streaming dataflow performance

- Improved, but plan remains I/O-bound (76%)
- **Main problem**: remote source latencies
  - Meanwhile, local resources are wasted
- **Complicating factor**: binding constraints
  - Remote queries dependent on other remote queries

- **Question**: How can execution be more efficient?
Speculative plan execution

- Execute operators ahead of schedule
  - Predict data based on past execution

- Allows greater degree of parallelism
  - Solves the problem caused by binding constraints

- Can lead to speedups > streaming dataflow
How to speculate?

- **General problem**
  - Means for issuing and confirming predictions

- **Two new operators**
  - **Speculate**: Makes predictions based on "hints"
  - **Confirm**: Prevents errant results from exiting plan
How to speculate?

• Example: CarInfo
  – Make predictions about cars based on search criteria
  – Makes practical sense:
    • Same criteria will typically yield same cars

BEFORE
How to speculate?

- **Example: CarInfo**
  - Make predictions about cars based on search criteria
  - Makes practical sense:
    - Same criteria will typically yield same cars
Detailed example

2002
Midsize coupe
$4000-$12000

W → Speculate → S

W → J → Confirm

W → W → W

Time = 0.0 sec
Issuing predictions

Time = 0.1 sec
Speculative parallelism

Time = 0.2 sec
Answers to hints

Time = 1.0 sec

Oldsmobile Olero
Dodge Stratus
Pontiac Grand Am
Mercury Cougar
Continued processing

Additions (corrections), if any

Time = 1.1 sec
Generation of final results

Time = 4.2 sec
Confirmation of results

Time = 4.3 sec
In practice: how it works

• Speculate generates *speculative tuples*

• These tuples are run by a separate pool of “speculative threads”
  – These threads only execute operator methods on speculative tuples

• Thus, the Speculate operator elicits more agent run-time parallelism
  – Greater thread-level parallelism (TLP)
  – Beyond the dataflow limit
Safety and fairness

• **Safety**
  – Confirm operator

• **Fairness**
  – **CPU**
    • Speculative operations executed by "speculative threads"
      – Lower priority threads

  – **Memory and bandwidth**
    • Speculative operations allocate "speculative resources"
      – Drawn from "speculative pool" of memory
      – Other solutions exist, such as RSVP (Zhang et al 1994)
Getting better speedups

- Cascading speculation
  - Single speculation allows a max speedup of 2
    - Time spent either speculating or confirming
  - Cascading speculation allows arbitrary speedups
    - Up to the length of the longest plan flow
Automatic plan transformation

• One important step is determining the set of candidate transformations

• However:
  – Determining this set is an expensive proposition
  – Assuming:
    • A candidate transformation can include one or more speculations
    • A given speculation is consumed by one and only one operator
  – The # of possible transformations:
    \[ ST(n) = (n-1) + n \times ST(n-1), \quad ST(1) = 0 \]
  – A single flow of 10 consecutive operators has over 3 million possible speculative schedules!
Automatic plan transformation

• **An alternative: leverage Amdahl's Law:**
  – Focus on most expensive path (MEP)

• **Basic algorithm**
  1. Find MEP
  2. Find **best** candidate speculative plan transformation
  3. IF no candidate found, THEN exit
  4. Transform plan accordingly
  5. **REPEAT** (anytime property)

• **The "best" candidate**
  – The one with the highest potential speedup

• **Algorithm assumes some addtl speculative overhead**
  – Function of the amount of data speculated about
CarInfo revisited

• Modified for speculative execution
  – Leverage potential of cascading speculation

• Optimistic performance
  – Execution time: \( \max \{1.2, 1.4, 1.5, 1.6\} = 1.6 \text{ sec} \)
  – Speedup over streaming dataflow: \( \frac{4.2}{1.6} = 2.63 \)
Another example: StockInfo

- **INPUT**
  - Company name

- **OUTPUT**
  - Chart comparing company stock vs competitor stock
StockInfo (cont’d)

• Original plan

• Modified for speculative execution
Web agent experiments

- **Time to first tuple**

- **Time to last tuple**
Web agent experiments

- **Time to first tuple**

- **Time to last tuple**
Value prediction

• Better value prediction = better speedups

• Prediction capability

<table>
<thead>
<tr>
<th>Category</th>
<th>Hint</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Previously seen</td>
<td>Previously seen</td>
</tr>
<tr>
<td>B</td>
<td>Never seen</td>
<td>Previously seen</td>
</tr>
<tr>
<td>C</td>
<td>Never seen</td>
<td>Never seen</td>
</tr>
</tbody>
</table>

• Examples:

Edmunds car list from search criteria

H → 2002 Midsize coupe 4K-12K

P → Olds Alero, Dodge Stratus, Pontiac Grand Am, Mercury Cougar

ConsumerGuide full review URL from summary URL

http://cg.com/summary/20812.htm
http://cg.com/full/20812.htm

http://cg.com/summary/12345.htm ?
Value prediction techniques

- **Caching**
  - Associate a hint with a predicted value

- **Classification**
  - Use features of a hint to predict value
  - **EXAMPLE**: Predicting car list from Edmunds

<table>
<thead>
<tr>
<th>Year</th>
<th>Type</th>
<th>Min</th>
<th>Max</th>
<th>Car list</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Midsize</td>
<td>8000</td>
<td>15000</td>
<td>(Oldmobile Alero, Dodge Stratus)</td>
</tr>
<tr>
<td>2002</td>
<td>Midsize</td>
<td>7500</td>
<td>14500</td>
<td>(Oldmobile Alero, Dodge Stratus)</td>
</tr>
<tr>
<td>2002</td>
<td>SUV</td>
<td>14000</td>
<td>20000</td>
<td>(Nissan Pathfinder, Ford Explorer)</td>
</tr>
<tr>
<td>2001</td>
<td>Midsize</td>
<td>11000</td>
<td>18000</td>
<td>(Honda Accord, Toyota Camry)</td>
</tr>
<tr>
<td>2002</td>
<td>SUV</td>
<td>18000</td>
<td>22000</td>
<td>(Nissan Pathfinder, Ford Explorer)</td>
</tr>
</tbody>
</table>

**Decision list**

- `type = SUV`: (Nissan Pathfinder, Ford Explorer)
- `type = Midsize`
- `...min <= 10000`: (Olds Alero, Dodge Stratus)
- `min > 10000`: (Honda Accord, Toyota Camry)
Value prediction techniques (cont'd)

- **Transduction**
  - Transducers are FSA that translate hint into prediction

Part of the prediction is based on the hint:

How do we extract & insert the dynamic part of the summary URL (e.g., 20812)?
Value transducers

• Synthesize predictions from hints
• Identify predicted value "templates"
  – Alternating seq of STATIC/DYNAMIC elements
• Value transducers built from templates
  – State transitions (arcs) = high-level operations:
    • INSERT, CACHE, CLASSIFY, TRANSDUCE

http://cg.com/summary/20812.htm

TRANSDUCE

http://cg.com/full/20812.htm

Dodge Stratus

CACHE or CLASSIFY

http://cg.com/summary/20812.htm
Learning value transducers

- Identify **STATIC/DYNAMIC** template
  - Find LCS for the set of predicted values, using technique based on (Hirschberg 1975)

- For each **STATIC** element,
  - Construct **INSERT** arc to next automata state

- For each **DYNAMIC** element,
  - Construct **TRANSDUCE**, **CLASSIFY**, or **CACHE** arc to next automata state
    - Prefer **TRANSDUCE** and **CLASSIFY** because
      - Better predictive capability on average
      - Better space efficiency on average
Detailed example: CarInfo URLs

HINTS:
http://cg.com/summary/20812.htm
http://cg.com/summary/12345.htm

ANSWERS:
http://cg.com/full/20812.htm
http://cg.com/full/12345.htm

\[
\text{INSERT(“http://cg.com/full/“)} \rightarrow \text{TRANSduce} \rightarrow \text{INSERT(“.htm“)}
\]
Experimental results

• Better accuracy than strictly caching

Hint classification

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Average number of examples required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car-Full</td>
<td>3</td>
</tr>
<tr>
<td>Rep-Graph</td>
<td>8</td>
</tr>
<tr>
<td>Phone-Detail</td>
<td>3</td>
</tr>
</tbody>
</table>
Experimental results

- More space efficient than strictly caching

**Hint classification**
(CarInfo summary review URL)

**Hint transduction**
(CarInfo full review URL)

Space savings (over caching)

- Number of examples
- Number of examples

- 0.00%
- 20.00%
- 40.00%
- 60.00%
- 80.00%
- 100.00%
Effect on spec exec performance

- CarInfo
Effect on spec exec performance

- RepInfo

![Bar chart showing average agent execution time for different numbers of tuples seen.](chart.png)
Value prediction summary

• Value prediction
  – Important part of speculative plan execution
  – Better value prediction = better average speedups

• Our approach: learn value transducers
  – Construct predicted value based on past experience
  – Learn STATIC/DYNAMIC prediction template using LCS

• Build value transducer based on template
  – INSERT arc(s) corresponds to STATIC parts
  – TRANSDUCE, CLASSIFY, CACHE arc(s) correspond to DYNAMIC parts
Related Work

• **Speculative execution**
  – Approximate & partial query results
    • [Hellerstein et al. 1997] [Shanmugasundaram et al. 2000] [Raman and Hellerstein 2001]
  
  – Executing anticipated actions in advance
    • Continual computation [Horvitz 2001], time-critical decision making [Greenwald and Dean 1994]
  
  – Other types of speculative execution
    • File system prefetching [Chang and Gibson 1999], control speculation in workflow processing [Hull et al. 2000]

  – Network prefetching
Related Work

• **Learning value predictors**
  – Predicting commands
    • Command line prediction [Davison and Hirsh 1998, 2001]
    • Assisted browsing [Lieberman 1995] [Joachims et al. 1997]
  – Value prediction as speedup learning
    • [Fikes et al. 1972], [Mitchell 1983], [Minton 1988]
  – Transducer learning
    • Provably correct transducers [Oncina et al. 1993]
      – Issues: Requires many examples, generalization capability differs
    • Transducers for data extraction [Hsu and Chang 1999]
  – URL prediction
    • [Zukerman et al. 1999], [Su et al. 2000]
Summary

• Theseus, Tukwila, Telegraph, Niagara are all:
  – Streaming dataflow systems

  – Targeting network-based query processing
    • Large source latencies
    • Unknown characteristics of sources

  – Proposed various techniques for improving the efficiency of processing data
    • More efficient operators (e.g., double-pipelined join)
    • Tuple-level adaptivity
    • Partial results for blocking operators
    • Speculative execution
Thank you