Plan Execution for Information Gathering

Craig Knoblock
University of Southern California

This talk is based in part on slides from Greg Barish

Outline of talk

• Introduction
• Streaming dataflow execution systems
• A streaming dataflow plan language
• Optimizing execution of streaming dataflow plans
  • Streaming operators
  • Tuple-level adaptivity
  • Partial results for blocking operators
  • Speculative execution
• Discussion
Motivation

- Problem
  - Information gathering may involve accessing and integrating data from many sources
  - Total time to execute these plans may be large
- Why?
  - Unpredictable network latencies
  - Varying remote source capabilities
  - Thus, execution is often I/O-bound
- Complicating factor: binding patterns
  - During execution, many sources cannot be queried until a previous source query has been answered

Traditional Approaches

- Executing information gathering plans
  - Generate a plan
  - Plan typically consists of a partial ordering of the operators
  - Execute the plan based on the given order
  - Operators process all of their input data before transmitting any results to consumer(s)
    - Operators as fast as their most latent input
    - Long delays due to the dependencies in the plan
Streaming Dataflow

- Plans consist of a network of operators
  - Each operator like a function
    - Example: Wrapper, Select, etc.
  - Operators produce and consume data
  - Operators “fire” when any part of any input data becomes available
  - Data routed between operators are relations
    - Zero or more tuples with one or more attributes
**Dataflow vs Von-Neumann**

\[((a + b) \times (c + d))\]

<table>
<thead>
<tr>
<th>abcde</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>ADD</td>
<td>MUL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parallelism of Streaming Dataflow**

- **Dataflow (horizontal parallelism)**
  - Decentralized, independent operator execution
  - Enables "maximally parallel" operator execution
    - Also known as the "dataflow limit"

- **Streaming/pipelining (vertical parallelism)**
  - Producer emits tuples to consumer ASAP
    - Producer & consumer can process same relation simultaneously
  - Effective because information gathering latencies can be high – even at the tuple level
    - Data often "trickles" out of I/O-bound operators
Example: The RepInfo Agent

**INPUT**
- Any street address
  - e.g., 4767 Admiralty Way, Marina del Rey, CA, 90292

**OUTPUT**
- Federal reps
  - 2 senators,
  - 1 house member
- For each rep:
  - Recent news
  - Real-time funding information

RepInfo Sources

Vote-Smart:
- List of officials

Project Vote Smart

The Most Trusted Source for Political Information

California Elected Officials:
- President:  
  - George W. Bush (Republican)
- Governor:  
  - Arnold Schwarzenegger (Republican)
- U.S. Senate:  
  - Senator Barbara Boxer (Democrat)
- U.S. House:  
  - Representative Jane Harman (Democrat)
RepInfo Sources

Vote-Smart:
- List of officials

Yahoo
- Recent news

Open Secrets
- Funding graph
OpenSecrets – Navigation + Fetching!

2002 Congressional Campaign Finance Profiles

Top 50 Industries

<table>
<thead>
<tr>
<th>Rank</th>
<th>Industry</th>
<th>Total</th>
<th>Dem Prop</th>
<th>SOP Prop</th>
<th>Top Recipient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lobbying/Law Firms</td>
<td>$3,393,365</td>
<td>69%</td>
<td>12%</td>
<td>John Barrasso (R-Wy)</td>
</tr>
<tr>
<td>2</td>
<td>Related</td>
<td>$3,394,324</td>
<td>41%</td>
<td>59%</td>
<td>Paul Wolfowitz (R-Ma)</td>
</tr>
<tr>
<td>3</td>
<td>Health Professionals</td>
<td>$3,095,703</td>
<td>42%</td>
<td>57%</td>
<td>Kay Hagan (D-Nc)</td>
</tr>
<tr>
<td>4</td>
<td>Real Estate</td>
<td>$1,485,221</td>
<td>42%</td>
<td>48%</td>
<td>Charles E. Schumer (D-Ny)</td>
</tr>
<tr>
<td>5</td>
<td>Reception</td>
<td>$1,395,783</td>
<td>55%</td>
<td>45%</td>
<td>Charles E. Schumer (D-Ny)</td>
</tr>
<tr>
<td>6</td>
<td>Insurance</td>
<td>$3,397,372</td>
<td>30%</td>
<td>51%</td>
<td>Max Baucus (D-Mt)</td>
</tr>
</tbody>
</table>

OpenSecrets – Navigation + Fetching!

Politics featured in opensecrets.org

- Jane Harman (D-Calif)

Campaign Finance Profiles

- [2002 Member of Congress]
- [2001 Member of Congress]
- [2000 Member of Congress]
- [Prior Member of Congress]
- [Prior Profile]

You can use our search engine to find more information on opensecrets.org.
### OpenSecrets – Navigation + Fetching!

**Jane Harman**

2001-2002 Total Receipts: $395,117
2001-2002 Total Spent: $145,959
Center for Main: $2,381,327
First election: 2020

#### Contributions by Sector

- **Individual contributions:** $139,410 (40.84%)
- **PAC contributions:** $129,425 (39.66%)
- **Candidate self-financing:** $0
- **Other:** $142 (0.01%)

#### Source of Funds

- **Business:** $139,774 (40.90%)
- **Labor:** $60,701 (18.69%)
- **Other:** $142 (0.01%)

#### How to Read This Chart:

The chart on this page gives the percentage contribution by sector. The largest source of contributions is the PAC sector, followed by the industry sector.

Note: The numbers displayed are for the entire election cycle, not just the current year.
Streaming Dataflow Systems for Network Environments

- **Focus**
  - Autonomous data sources on the Internet
  - Unpredictable network latencies

- **Network Query Engines**
  - Build plans to support queries
    - Tukwila
    - Telegraph
    - Niagara

- **Agent-based Execution System**
  - Support a richer plan language
    - Theseus
A Streaming Dataflow Plan Language

Theseus

- A plan language and execution system for Web-based information integration
  - Expressive enough for monitoring a variety of sources
  - Efficient enough for near-real-time monitoring
Expressivity

- Basic relational-style operators
  - Select, Project, Join, Union, etc.

- Operators for gathering Web data
  - Wrapper
    - Database-like access to a Web source
  - Xquery, Rel2Xml, and Xml2Rel
    - Enables better integration with XML sources

- Operators for monitoring Web data
  - DbExport, DbQuery, DbAppend, DbUpdate
    - Facilitates the tracking of online data
  - Email, Phone, Fax
    - Facilitates asynchronous notification

Expressivity

- Operators for extensibility
  - Apply: single-row functions (e.g., UPPER)
  - Aggregate: multi-row functions (e.g., SUM)

- Operators for conditional plan execution
  - Null: Tests and routes data accordingly

- Subplans and recursion
  - Plans are named and have INPUT & OUTPUT
    - We can use them as operators (subplans) in other plans
  - Subplans make recursion possible
    - Makes it easy to follow arbitrarily long list of result pages that are each separated by a NEXT page link
  - Subplans encourage modularity & reuse
Operators

operator (Input1, Input2, …: Output1, Output2, …)
    wait: waitInput1, waitInput2, ...
    enable: enableInput1, enableInput2, ...

• Data formats
  • Operators pass relations
  • Relations are composed of tuples
  • Each attribute of a tuple can be primitive, relation, or XML object

Operator Streaming

• Operators support stream-oriented processing
  • Firing rule met when any input receives a tuple
    • This enables ASAP processing of data
    • End of data signaled by end-of-stream (EOS)

• Operators vary on when they can begin output:
  • Union: immediately (i.e., for each input)
  • Minus: after EOS for second input has arrived
  • Email: after EOS for all inputs have arrived
Wrapper Operator

**PURPOSE:** Extract data from web pages as relation

**INPUT:**
- **Name:** URL prefix of wrapper
- **bind_map:** Wrapper binding map
- **bind_dat:** Binding tuples

**OUTPUT:**
- **new_rel:** Incoming relation joined with new attributes

```
auth = USER  PASSWORD
  greg  secret

wrapper("http://fetch.com?wrapper=foo",
  "user=$user, pwd=$password", auth : quotes)

quotes = USER  PASSWORD  SYMBOL  PRICE
  greg  secret    ORCL    15.50
  greg  secret    CSCO    21.50
```

Plans and Subplans

```
plan planName
{
  input: planInput1, planInput2, ...
  output: planOutput1, planOutput2, ...
  body {
    operator (opInput1,... : opOutput1,...)
    operator ...
    ...
  }
}
```

- Plans can be called just like operators (subplans)
Example plan: TheaterLoc

```
PLAN theaterloc
{(INPUT: city
OUTPUT: latlons, map_url
BODY
{wrapper ("cuisinenet", "name, addr", city : restaurants)
wrapper ("yahoo_movies", "name, addr" city : theaters)
union (restaurants, theaters : addresses)
wrapper ("geocoder", "name, lat, lon", addresses : latlons)
wrapper ("tigermap", latlons : map_url)
})
```
Transactions

- Enable
  - Concurrent plan access by multiple clients
  - Recursive plan execution
- Transactions each assigned unique ID
- Individual transactions can be aborted
- All transactions are assigned a “time to live”
  - Unprocessed data is garbage collected by Theseus

Conditionals and Recursion

- Conditional outputs are defined by enabling outputs depending on the action results
  \[ \text{Null}(\text{inStream} : \text{outStreamTrue},\text{outStreamFalse}) \]
- Plans can be called recursively
  - Termination defined by conditional operators
  - Transactions support recursive calls in same execution environment
  - System provides tail-recursion optimization
Real Estate Plan

New Listing:
3br 2bath
200K

Send Email Notification

Craig Knoblock  University of Southern California
Parallel Remote Data Retrievals

Optimizing Streaming Dataflow Plans
Adaptive Query Execution

- Network Query Engines
  - Tukwila (Ives et al., 1999)
    - Operator reordering
    - Optimized operators
  - Telegraph (Hellerstein et al. 2000)
    - Tuple-level adaptivity
  - Niagara (Naughton, DeWitt, et al. 2000)
    - Partial results for blocking operators

- Agent Execution Systems
  - Theseus (Barish & Knoblock, 2002)
    - Speculative execution

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
Adaptive Double Pipelined Hash Join Operator

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

- Double Pipelined Hash Join
  - Outputs data immediately
  - Symmetric
  - More memory

Dynamic Collector Operator

- Smart union operator

- Supports
  - Timeouts
  - slow sources
  - overlapping sources

```
WHEN timeout(CustReviews)
DO activate(NYTimes),
activate(alt.books)
```
**Tuple-level Adaptivity**

*(Hellerstein et al., 2000)*

- **Optimize horizontal parallelism**
  - Adaptive dataflow on clusters (i.e., data partitioning)

- **Optimize vertical parallelism**
  - Leverage commutative property of query operators to dynamically route tuples for processing
  - Result: adaptive streaming

---

**When can processing order be changed?**

- **Moment of symmetry:**
  - Inputs can be swapped without state management
  - Nested Loops: at the end of each inner loop
  - Merge Join: any time
  - Hybrid Hash Join: never!

From Avnur & Hellerstein, SIGMOD 2000
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

Execution with partial results [Shanmugasundaram et al. 2000]

- Query execution involves evaluation of partial results
  - Reduces blocking nature of aggregation or joins
- Basic idea
  - Execute future operators as data streams in, refine as slow operators catch up
  - Execution is still driven by availability of real data
  - Notion of refinement is similar to "correction" in speculative execution
Speculative Execution

- Standard streaming dataflow execution
  - Still I/O-bound (most operators are I/O-bound), CPU underused
  - Binding patterns compound delays
- To further increase parallelism: speculate about execution
  - Use earlier data as hints to speculatively execute downstream operators

Elaboration:

- Goal: parallelize I/O
- Standard streaming dataflow execution is still I/O-bound (most operators are I/O-bound), and CPU underused. Binding patterns compound delays.
- To further increase parallelism, speculate about execution by using earlier data as hints to speculatively execute downstream operators.

Speculating about plan execution

- Speculate about input to plan operators
  - Increase the level of operator-level parallelism
- Research questions
  - How to speculate?
    - What mechanism allows speculation to occur?
  - When to speculate?
    - What triggers speculation?
  - What to speculate about?
    - How do we predict data?
- Additional challenges
  - Maintaining correctness and fairness
RepInfo agent plan

Execution performance

- Measuring performance
  - Amdahl's law
    - Execution is only as fast as the costliest linear sequence
  - Thus:
    - Slowest single data flow = fastest possible overall performance

<table>
<thead>
<tr>
<th>Flow</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vote-Smart, Select, Yahoo, Join</td>
<td>3.3 sec</td>
</tr>
<tr>
<td>Vote-Smart, Select, OpenSecrets, Join</td>
<td>6.2 sec</td>
</tr>
</tbody>
</table>

- Execution time = MAX (3.3, 6.2) = \(6.2\) sec
Overview of approach

- **Automatically augment plan with 2 operators**
  - **Speculate**: Makes predictions and corrections
  - **SpecGuard**: Halts errant speculation

![Diagram showing the flow of information with Speculate and SpecGuard]

Resulting performance

- **RepInfo (original plan)**
  - Execution time: **6.2 sec**
- **RepInfo-Spec**
  - Individual flow performance:

<table>
<thead>
<tr>
<th>Flow</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vote-Smart, Select</td>
<td>1.4 sec</td>
</tr>
<tr>
<td>Yahoo, Join</td>
<td>1.9 sec</td>
</tr>
<tr>
<td>OpenSecrets, Join</td>
<td>4.8 sec</td>
</tr>
</tbody>
</table>

- Thus, execution time is now **4.8 sec**
  - Speedup = ( 6.2 / 4.8 ) = 1.3
Plan execution starts

Speculation about representatives
Speculation results received

Time = 1.8

Speculate

W → S → W → J → SpecGuard

Time = 2.0

Speculate

W → S → W → J → SpecGuard

Barbara Boxer
Dianne Feinstein
Jane Harman
Confirming speculation

Time = 4.8

Cascading speculation

- Major limitation thus far:
  - We are only speculating once
- Cascading speculation
  - Speculation based on speculation
  - Theoretical speedup of above example = \( \frac{10}{1} = 10 \)
Cascading speculation

ReplInfo Example:
- Use predicted officials to speculate about the OpenSecrets member and funding URLs

• Estimated performance
  - Slowest existing flow = MAX(1.4, 1.9, 1.4, 2.4) = 2.4 seconds
  - Speedup = (6.2 / 2.4) = 2.59

Ensuring correctness and fairness

Correctness
- SpecGuard does this
- Never emits tuples unless confirmed
- Must be placed prior to
  - Plan exit
  - Any operators that change the external world

Fairness
- Speculation must never usurp normal execution
- Plan execution involves multiple concurrent threads
  - Operators are associated with individual threads
- One simple solution:
  - Make Speculate and SpecGuard lower priority threads
  - Let the CPU handle fair scheduling
Where and when to speculate?

- Generally speaking:
  - Speculate about those operators that are:
    - Dynamic (not FDs)
    - Not the initial set of operators executed
  - Remember: Dataflow ≠ von-Neumann
    - Execution is not sequential
    - Instead: a set of independent data flow paths
  - Amdahl's law
    - Most expensive path (MEP) is the prime concern
    - Optimizing anything BUT the MEP is a waste

Automatic plan augmentation

- Focus on most expensive path (MEP)
  - Specifically on bottleneck operators (e.g., Wrapper)
- Algorithm sketch
  - Locate MEP
  - Find "best" candidate transformation for that path
  - If no candidate found, then exit
  - Transform plan accordingly
  - Repeat
- Finding the "best" candidate
  - Identify path with highest likely average execution time
The challenge

- We need to be able to predict data
- Example
  - Predict federal officials given an address
- Categories of predictions
  
<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>
- How do we deal with…?
  - Prediction given new hints
  - Making new predictions

Caching

- Associate answers with previously seen hints

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4676 Admiralty Way, Marina del Rey, CA, 90292</td>
<td>Boxer, Feinstein, Waxman</td>
</tr>
<tr>
<td>14044 Panay Way, Marina del Rey, CA 90292</td>
<td>Boxer, Feinstein, Harman</td>
</tr>
<tr>
<td>4065 Lincoln Blvd, Venice, CA 90405</td>
<td>Boxer, Feinstein, Harman</td>
</tr>
</tbody>
</table>

- Method of prediction
  1. When hint arrives, locate value in table
  2. If hint not in table, do not issue prediction
  3. Otherwise, predict the value found
- Problems
  - Only handles predictions of category A
  - Cannot deal with new hints or issue new predictions
  - Space inefficient
Decision trees

- Can be used to learn that, when predicting officials, 
  $\rightarrow$ **city** and **zip** are key attributes

<table>
<thead>
<tr>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Representative</th>
</tr>
</thead>
<tbody>
<tr>
<td>14044 Panay Way</td>
<td>Marina del Rey</td>
<td>CA</td>
<td>90292</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>4676 Admiralty Way</td>
<td>Marina del Rey</td>
<td>CA</td>
<td>90292</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>101 Washington Blvd</td>
<td>Venice</td>
<td>CA</td>
<td>90292</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>1301 Main St</td>
<td>Venice</td>
<td>CA</td>
<td>90291</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>1906 Lincoln Blvd</td>
<td>Venice</td>
<td>CA</td>
<td>90291</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>2107 Lincoln Blvd</td>
<td>Santa Monica</td>
<td>CA</td>
<td>90405</td>
<td>Henry Waxman</td>
</tr>
<tr>
<td>2202 S Centinela Ave</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90064</td>
<td>Henry Waxman</td>
</tr>
<tr>
<td>4065 Glencoe Ave</td>
<td>Marina del Rey</td>
<td>CA</td>
<td>90292</td>
<td>Diane Watson</td>
</tr>
<tr>
<td>3970 Berryman Ave</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90066</td>
<td>Diane Watson</td>
</tr>
<tr>
<td>11461 Washington Blvd</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90066</td>
<td>Diane Watson</td>
</tr>
</tbody>
</table>

- Since prediction is based on subset of attributes 
  $\rightarrow$ prediction given new hints is possible

Transducers for hint translation

- Recall that we want to be able to predict
  

- Prediction viewed as a translation
  
  - Simple subsequential transducers are used in NLP research for language translation
  
  - General idea
    - Construct alignment between tokens of L1 and L2
    - Build transducers that generate L2 sentences from L1 sentences
      - Transduction can be applied at the word or letter level
Transducers for hint translation

- Example
  - Construct alignment

- Build transducer

![Diagram](image)

Experimental results

- CPU impact of sample run

![Graph](image)
Discussion

- Theseus, Tukwila, Telegraph, Niagara are all:
  - Streaming dataflow systems
  - Target network-based query execution
    - Large source latencies
    - Unknown characteristics of sources
  - Focus on techniques for improving the efficiency of plan execution
- Challenges in Plan Execution
  - How to interleave planning and execution
  - How to interleave sensing actions
  - Other approaches to improve performance
  - Improved techniques for making predictions

Bibliography

- Dataflow computing
  - Foundations
  - Dataflow / von Neumann hybridization
Bibliography

- Parallel database systems
  - Shared nothing architectures
  - Parallel query execution

- Network information gathering
  - Niagara
  - Telegraph
Bibliography

• Network information gathering
  • Theseus
  • Tukwila

• Adaptive query processing
  • Adaptive tuple routing
  • Evaluation of partial results
  • Speculative execution