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

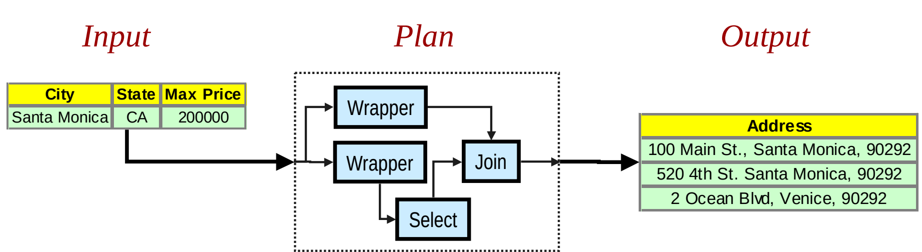
Traditional Approaches

- 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 Execution Systems

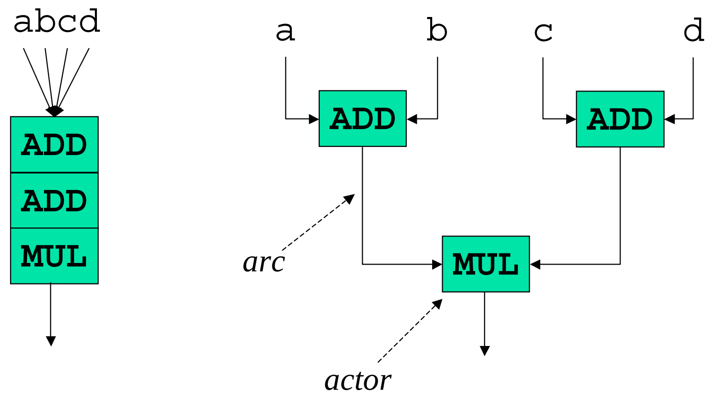
Streaming Dataflow

- 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) * (c + d))

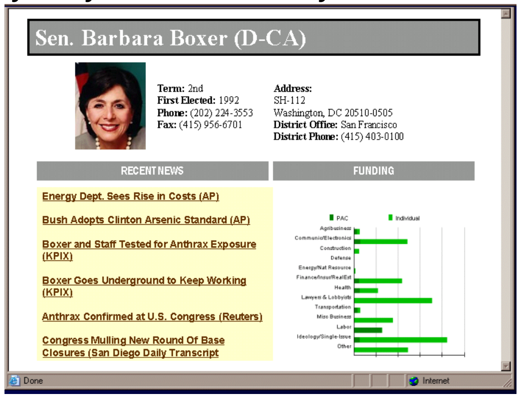


Parallelism of Streaming Dataflow

- 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

RepInfo Sources

Vote-Smart:
- –List of officials

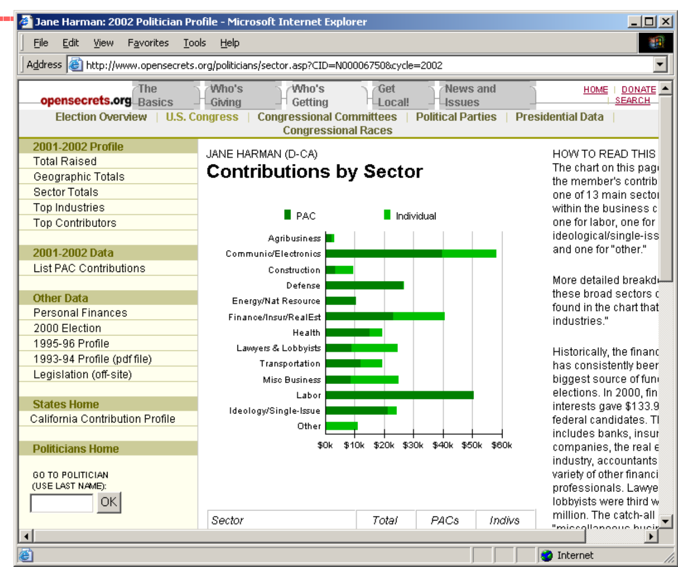
Yahoo
- –Recent news

RepInfo Sources

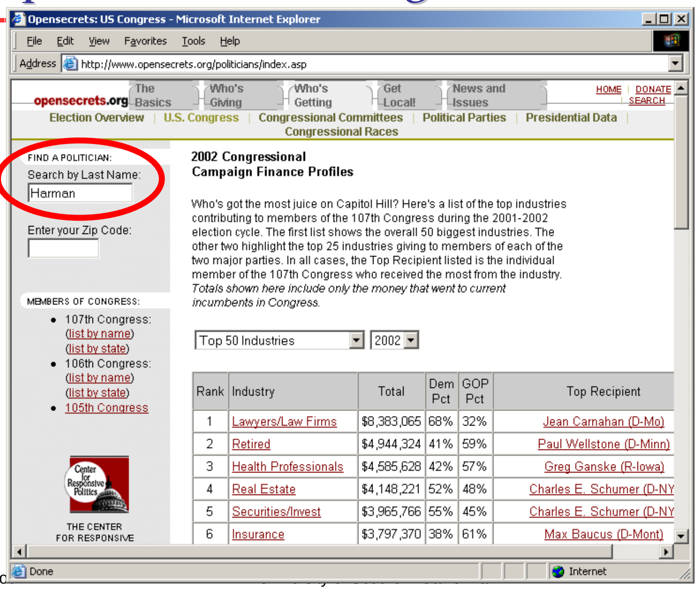
Vote-Smart:
- –List of officials

Yahoo
- –Recent news

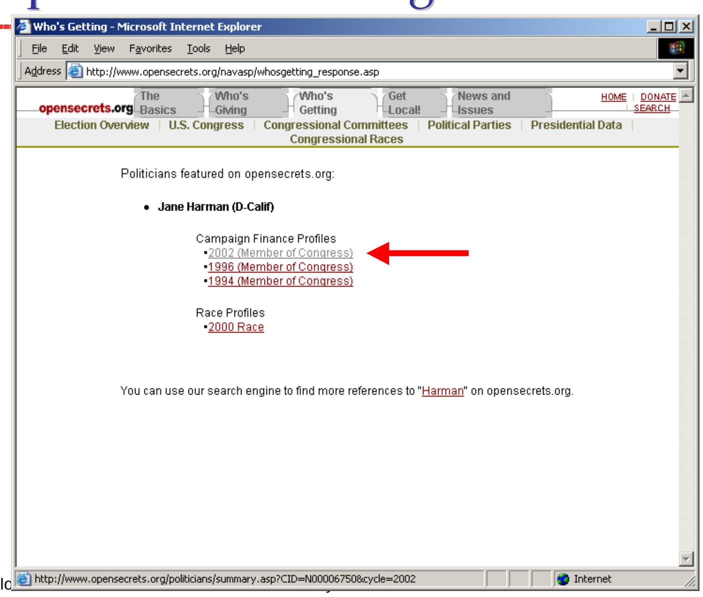
Open Secrets
- –Funding graph



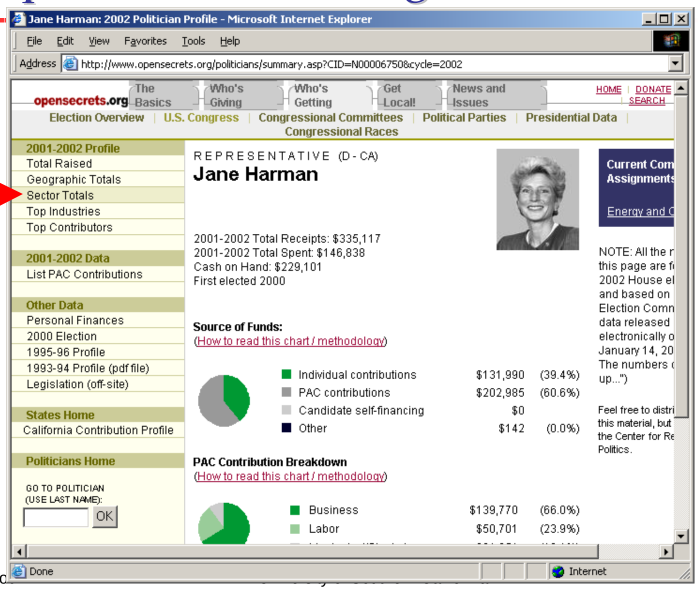
OpenSecrets – Navigation + Fetching!



OpenSecrets – Navigation + Fetching!

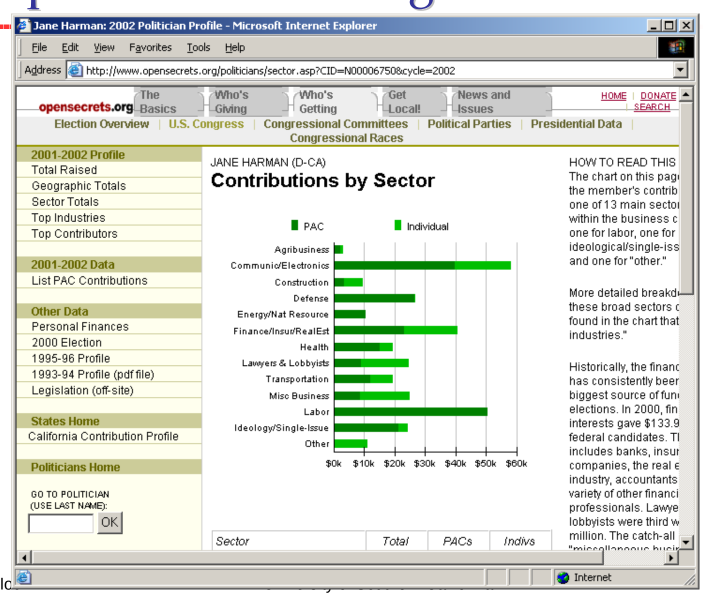


OpenSecrets – Navigation + Fetching!



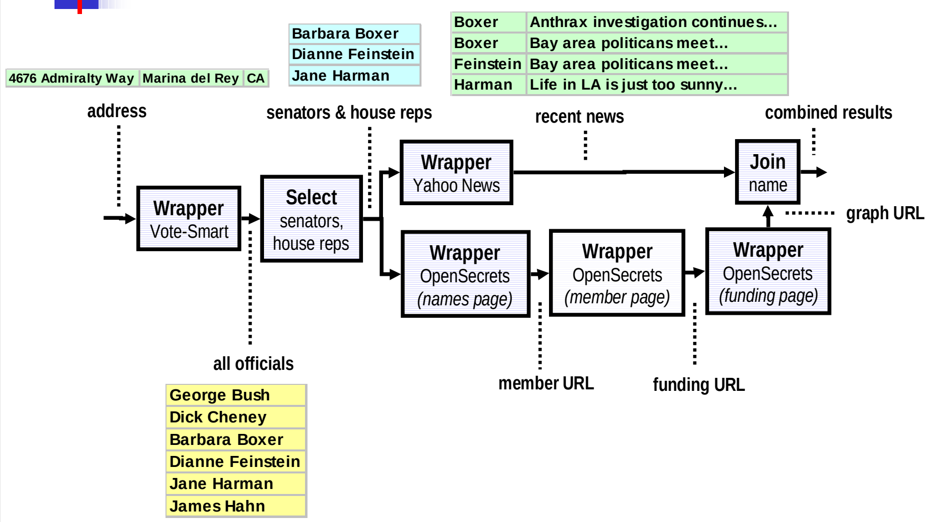
Craig Knoblo… 15

OpenSecrets – Navigation + Fetching!



Craig Knoblo… 16

RepInfo agent plan



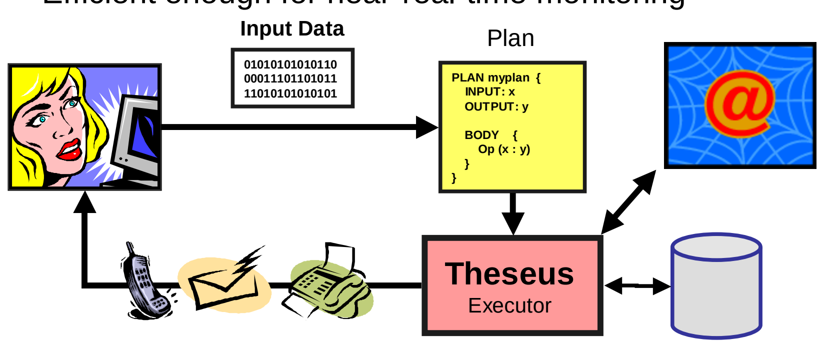
Streaming Dataflow Systems for Network Environments

- 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 <u>EOS for second input</u> has arrived**
  - Email: **after <u>EOS for all inputs</u> 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
```

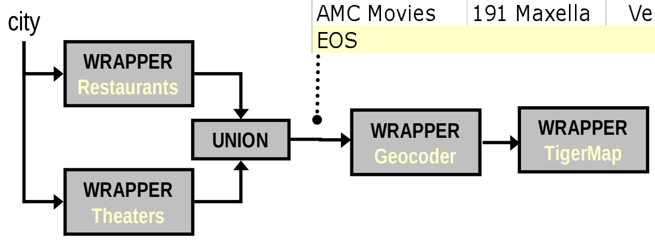

Plans and Subplans

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

- Plans can be called just like operators (subplans)

Example plan: TheaterLoc

NAME	ADDRESS	CITY	STATE
Rock	187 Maxella	Venice	CA
AMC Movies	191 Maxella	Venice	CA
EOS			



TheaterLoc Plan

```
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

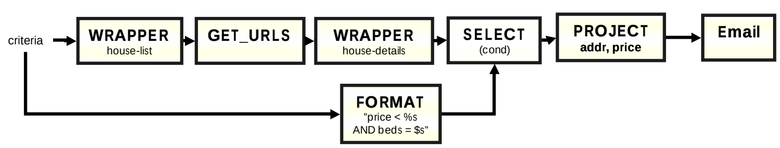
```
Null(inStream : outStreamTrue,outStreamFalse)
```

- Plans can be called recursively
  - Termination defined by conditional operators
  - Transactions support recursive calls in same execution environment
  - System provides tail-recursion optimization

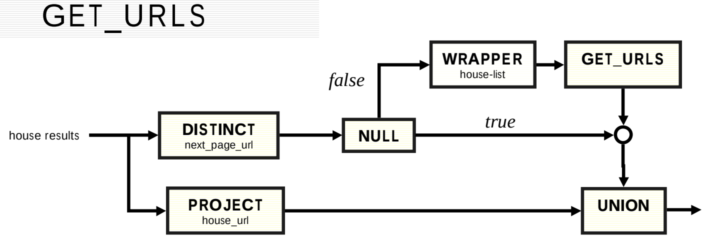
Real Estate Plan

Real Estate Plan

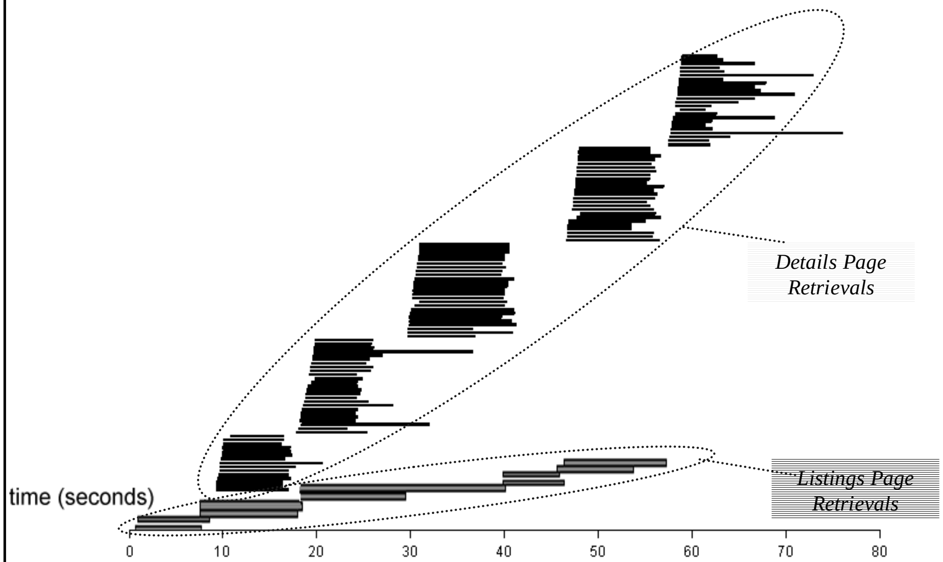
FIND_HOUSES



GET_URLS



Parallel Remote Data Retrievals



Optimizing Streaming Dataflow Plans

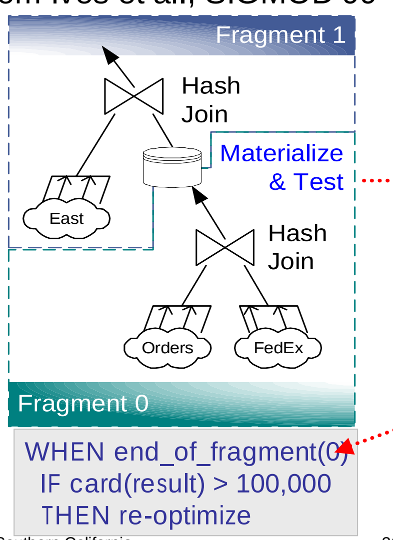
Adaptive Query Execution

- 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

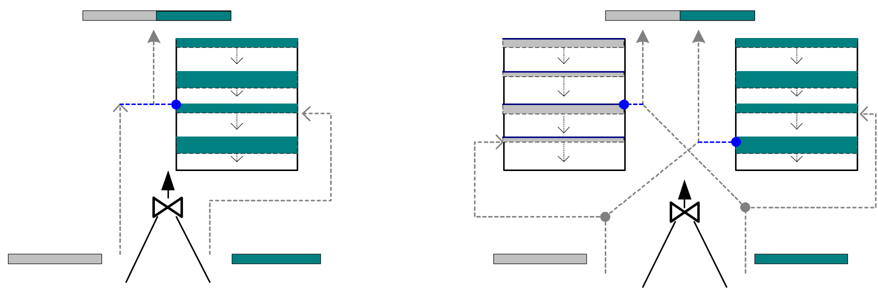
From Ives et al., SIGMOD'99

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



Adaptive Double Pipelined Hash Join Operator

From Ives et al., SIGMOD'99



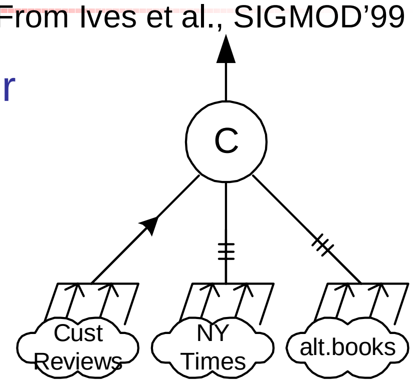
Hybrid Hash Join
- No output until inner read
- Asymmetric (inner vs. outer)

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

Dynamic Collector Operator

From Ives et al., SIGMOD'99

- Smart union operator
- Supports
  - Timeouts
  - slow sources
  - overlapping sources



WHEN
timeout(CustReviews)
DO activate(NYTimes),
activate(alt.books)

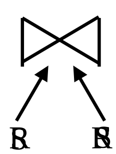
Tuple-level Adaptivity
(Hellerstein et al. 2000)

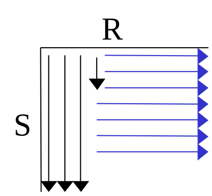
- Optimize horizontal parallelism
  - Adaptive dataflow on clusters (ie, data partitioning)
- Optimize vertical parallelism
  - Leverage commutative property of query operators to dynamically route tuples for processing
  - Result: adaptive streaming

When can processing order be changed?

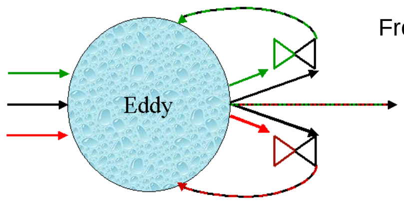
- Moment of symmetry:
  - Inputs can be swapped without state management
  - Nested Loops: at the end of each inner loop
  - Merge Join: any time
  - Hybrid Hash Join: never!

From Avnur & Hellerstein, SIGMOD 2000





Beyond Reordering Joins



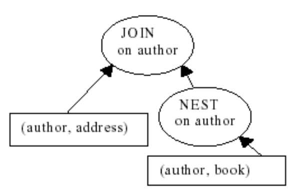
From Avnur & Hellerstein, SIGMOD 2000

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

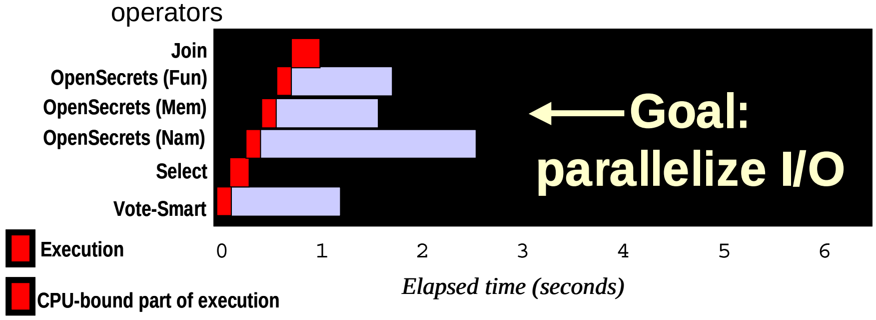
Execution with partial results [Shanmugasundaram et al. 2000]

- 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

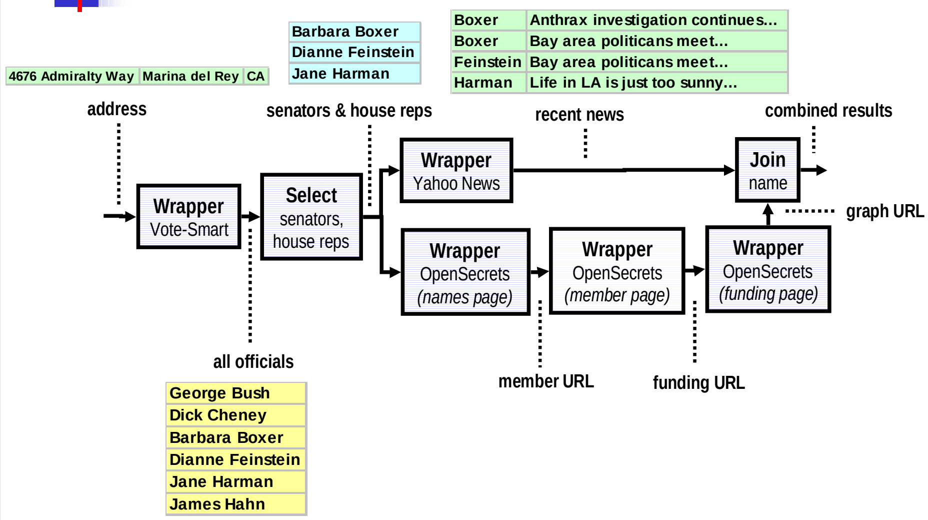
- Standard streaming dataflow execution
  - Still I/O-bound (most operators are I/O-bound), CPU underused
  - Binding patterns compound delays
- To further increase parallelism: speculate about execution
  - Use earlier data as hints to speculatively execute downstream operators



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

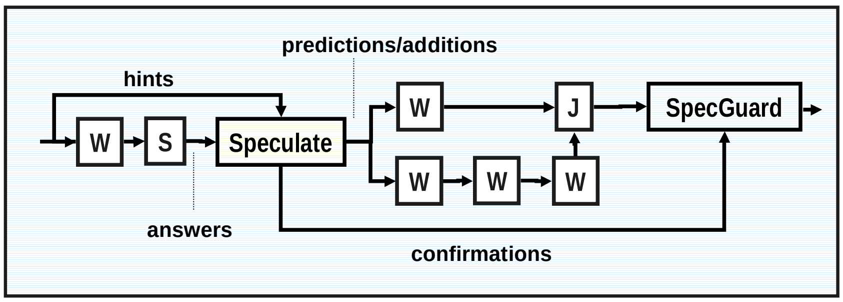
- Measuring performance
  - Amdahl's law
    - Execution is only as fast as the costliest linear sequence
- Thus:
  - Slowest single data flow = fastest possible overall performance

Flow	Time
Vote-Smart, Select, Yahoo, Join	3.3 sec
Vote-Smart, Select, OpenSecrets, Join	6.2 sec

- 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



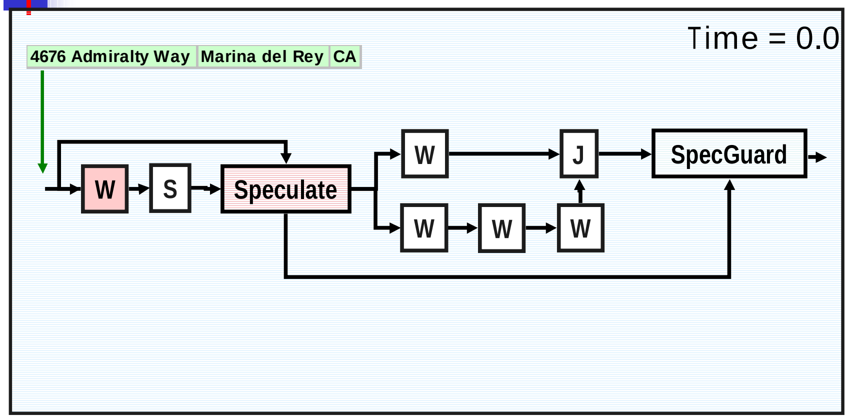
Resulting performance

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

Flow	Time
Vote-Smart, Select	1.4 sec
Yahoo, Join	1.9 sec
OpenSecrets, Join	4.8 sec

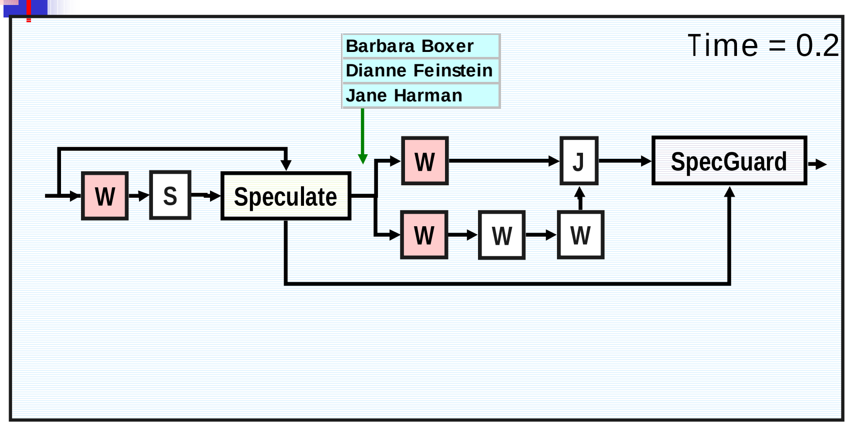
- Thus, execution time is now **4.8 sec**
  - Speedup = (6.2 / 4.8) = 1.3

Plan execution starts



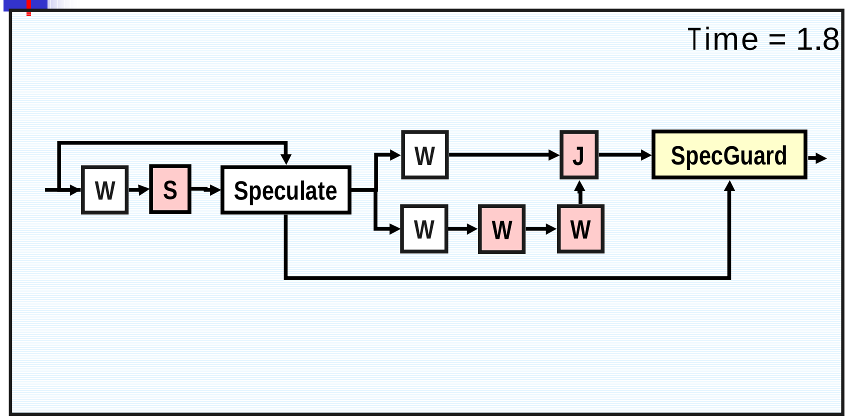
Craig Knoblock University of Southern California 49

Speculation about representatives

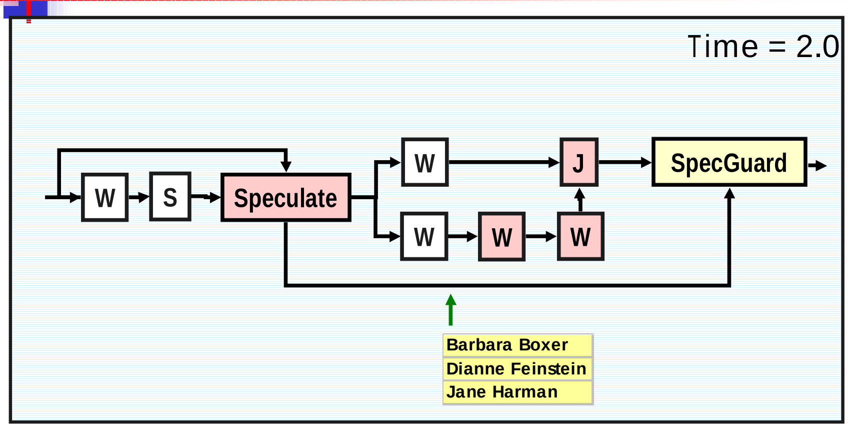


Craig Knoblock University of Southern California 50

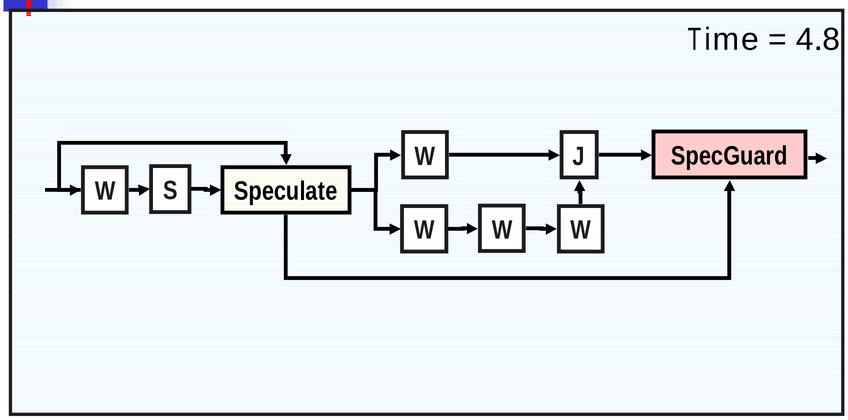
Speculation results received



Speculation results recieved

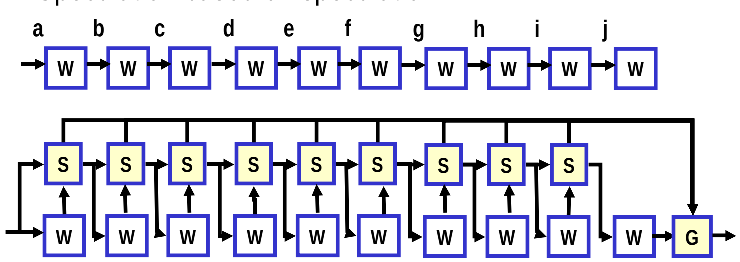


Confirming speculation



Cascading speculation

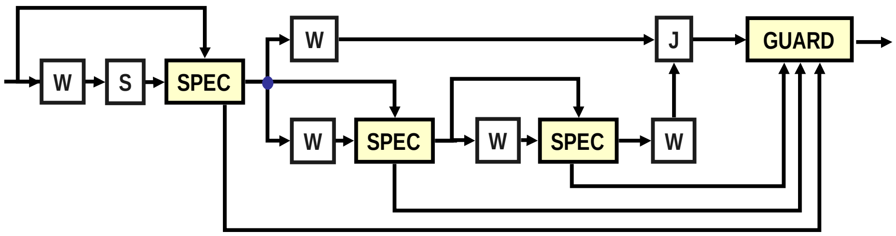
- Major limitation thus far:
  - We are only speculating once
- Cascading speculation
  - Speculation based on speculation



  - Theoretical speedup of above example= (10/1)= 10

Cascading speculation

- RepInfo 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

Category	Hint	Prediction
A	Previously seen	Previously seen
B	Never seen	Previously seen
C	Never seen	Never seen

- How do we deal with…?
  - Prediction given new hints
  - Making new predictions

Caching

- Associate answers with previously seen hints

Key	Value
4676 Admiralty Way, Marina del Rey, CA, 90292	Boxer, Feinstein, Harman
14044 Panay Way, Marina del Rey, CA 90292	Boxer, Feinstein, Harman
4065 Lincoln Blvd, Venice, CA 90405	Boxer, Feinstein, Waxman

- 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, → **city** and **zip** are key attributes

hint answer

Street	City	State	Zip	Representative
14044 Panay Way	Marina del Rey	CA	90292	Jane Harman
4676 Admiralty Way	Marina del Rey	CA	90292	Jane Harman
101 Washington Blvd	Venice	CA	90292	Jane Harman
1301 Main St	Venice	CA	90291	Jane Harman
1906 Lincoln Blvd	Venice	CA	90291	Jane Harman
2107 Lincoln Blvd	Santa Monica	CA	90405	Henry Waxman
2222 S Centinela Ave	Los Angeles	CA	90064	Henry Waxman
4065 Glencoe Ave	Marina del Rey	CA	90292	Diane Watson
3970 Berryman Ave	Los Angeles	CA	90066	Diane Watson
11461 Washington Blvd	Los Angeles	CA	90066	Diane Watson

city = Marina del Rey: Jane Harman (2)
city = Venice: Jane Harman (3)
city = Santa Monica: Henry Waxman (1)
city = Los Angeles:
:...zip <= 90064: Henry Waxman (1)
zip > 90064: Diane Watson (2)

- Since prediction is based on subset of attributes → prediction given **new hints** is possible

Transducers for hint translation

- Recall that we want to be able to predict

  http://www.opensecrets.org/politicians/**summary**.asp?CID=N00007364
  http://www.opensecrets.org/politicians/**sector**.asp?CID=N00007364

- 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

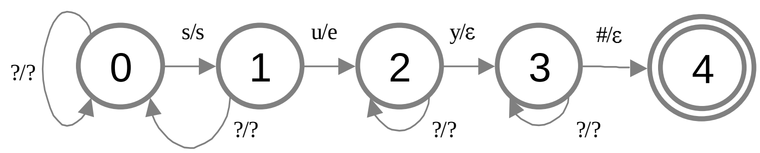
Transducers for hint translation

- Example
  - Construct alignment

| http:// | www.opensecrets.org | / | summary.asp | ? | CID | = | N00006692 | & | cycle | = | 2002 |

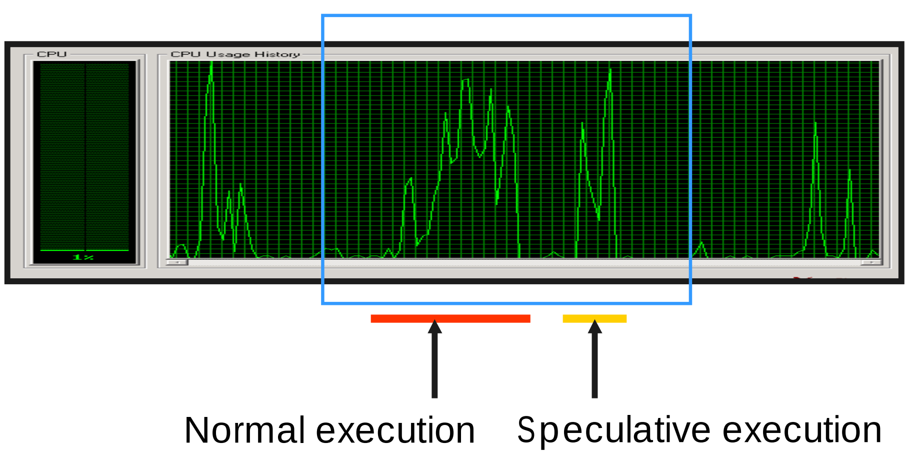
| http:// | www.opensecrets.org | / | sector.asp | ? | CID | = | N00006692 | & | cycle | = | 2002 |

  - Build transducer



Experimental results

- **CPU impact of sample run**



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
    - Dennis, Jack B. (1974). First version of a data-flow procedure language. Lecture Notes in Computer Science vol. 19, pp 362—376.
    - Arvind and R.S. Nikhil (1990). Executing a program on the MIT tagged-token dataflow architecture. IEEE Transactions on Computers (1990), pp 300–318.
  - Dataflow / von Neumann hybridization
    - Iannucci, Robert A. (1988) Toward a dataflow/von Neumann hybrid architecture. In Proceedings of the 19th Annual International Conference on Computer Architecture (ICSA), pp 131—140.
    - Papadopolous, Gregory M. and Kenneth R. Traub. (1991) Multithreading: a revisionist view of dataflow architectures. In Proceedings of the 18th Annual Symposium on Computer Architecture, pp 342—351.

Bibliography

- Parallel database systems
  - Shared nothing architectures
    - DeWitt, David J. and Jim Gray (1992). Parallel database systems: the future of high-performance database systems. Communications of the ACM 35(6), pp 85-98.
  - Parallel query execution
    - Wilschut, Annita N. and Peter M.G. Apers. (1991) Dataflow query execution in a main memory environment. In Proceedings of the First International Conference on Parallel and Distributed Information Systems, pp 68–77.
    - Graefe, Goetz (1994) Volcano – an extensible and parallel query evaluation system. IEEE Transactions on Knowledge and Data Engineering 6(1), pp 120–135 .

Bibliography

- Network information gathering
  - Niagara
    - Naughton, Jeffrey F., David J. DeWitt, David Maier, and many others. (2001). The niagara internet query system. IEEE Data Engineering Bulletin 24(2): 27–33.
  - Telegraph
    - Hellerstein, Joseph M., Michael J. Franklin, Sirish Chandrasekaran, Amol Deshpande, Kris Hildrum, Sam Madden, Vijayshankar Raman and Mehul A. Shah (2000). Adaptive query processing: technology in evolution. IEEE Data Engineering Bulletin 23(2): 7--18.

Bibliography

- Network information gathering
  - Theseus
    - Barish, Greg and Craig A. Knoblock. An expressive and efficient language for information gathering on the web. (2002) Proceedings of the Sixth International Conference on AI Planning and Scheduling Workshop: Is There Life Beyond Operator Sequencing? - Exploring Real-World Planning. pp. 5–12.
  - Tukwila
    - Ives, Zachary G., Daniela Florescu, Marc Friedman, Alon Levy and Daniel S. Weld (1999). An adaptive query execution system for data integration. In Proceedings of the ACM SIGMOD International Conference on Management of Data. pp 299–310.

Bibliography

- Adaptive query processing
  - Adaptive tuple routing
    - Avnur, Ron and Joseph M. Hellerstein (2000). Eddies: continuously adaptive query processing. Proceedings of the ACM SIGMOD International Conference on the Management of Data. pp. 261--272.
  - Evaluation of partial results
    - Shanmugasundaram, Jayavel, Kristin Tufte, David J. DeWitt, Jeffrey F. Naughton and David Maier (2000). Architecting a network query engine for producing partial results. Proceedings of the ACM SIGMOD 3rd International Workshop on Web and Databases (WebDB). pp. 17-22.
    - Raman, Vijayshankar and Joseph M. Hellerstein (2002). Partial results for online query processing. Proceedings of the ACM SIGMOD International Conference on the Management of Data.
  - Speculative execution
    - Barish, Greg and Craig A. Knoblock (2002) Speculative execution for information gathering plans. In Proceedings of the Sixth International Conference on AI Planning and Scheduling, pp 259–268.