Speculative Plan Execution for Information Agents

Greg Barish
University of Southern California
Information Sciences Institute

Advisor: Professor Craig A. Knoblock
Outline

1. Review and motivating example
2. Speculative plan execution
3. Value prediction for speculative execution
4. Related work
5. 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]
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**

---

**Time (seconds)**

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

**Operations**

- CPU-bound operation
- I/O-bound operation
Dataflow-style CarInfo agent plan

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

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

search criteria

WRAPPER
Edmunds Search

SELECT
maker != "Oldsmobile"

WRAPPER
ConsumerGuide Search

WRAPPER
ConsumerGuide Summary

WRAPPER
ConsumerGuide Full Review

WRAPPER
NHTSA Search

(safety reports)

JOIN

(car reviews)

((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)
Expressing the CarInfo agent plan

PLAN car-info {

  INPUT: criteria
  OUTPUT: reviews-and-ratings

  BODY {
    Wrapper ("Edmunds", criteria : cars)
    Select (cars, "maker != 'Oldsmobile'" : filtered-cars)
    Wrapper ("NHTSA", filtered-cars : safety-ratings)
    Wrapper ("CG Search", filtered-cars : summary-urls)
    Wrapper ("CG Summary", summary-urls : full-urls)
    Wrapper ("CG Full", full-urls : car-reviews)
    Join (safety-ratings, car-reviews, "l.make=r.make and l.model=r.model" : reviews-and-ratings)
  }
}

Streaming dataflow executor

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

Example:

- WRAPPER
  - Edmunds Search
  - (Midsize cpe/hatchbk, $4000 to $12000, 2002)

- SELECT
  - maker != "Oldsmobile"
  - ((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
Focus of 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
Outline

1. Review and motivating example

2. Speculative plan execution

3. Value prediction for speculative execution

4. Related work

5. Summary
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
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

Time = 0.0 sec
Issuing predictions

Oldsmobile Olero T1
Dodge Stratus T2
Pontiac Grand Am T3
Mercury Cougar T4

Time = 0.1 sec
Speculative parallelism

Time = 0.2 sec
Answers to hints

Time = 1.0 sec
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

Dodge Stratus (safety) (review)
Pontiac Grand Am (safety) (review)
Mercury Cougar (safety) (review)
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 sec
  – Speedup over streaming dataflow: \(\frac{4.2}{1.6}\) = 2.63
Example: TheaterLoc

- **INPUT**
  - City & state

- **OUTPUT**
  - Map of region annotated w/ theaters & restaurants
Example: TheaterLoc

• **Original plan:**

```
  WRAPPER
  Dine.com
  city

  UNION

  WRAPPER
  Geocoder
  WRAPPER
  U.S. CensusTiger Map

  map
```

• **Modified for speculative execution:**

```
  SPEC

  city

  W

  U

  W

  CONFIRM

  W

  map
```
Example: The RepInfo Agent

• INPUT
  – Any street address
    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
Example: RepInfo

• Original

• Modified for speculative execution
Example: StockInfo

- **INPUT**
  - Company name

- **OUTPUT**
  - Chart comparing company stock vs competitor stock
Example: StockInfo

• Original plan

• Modified for speculative execution
Web agent experiments

• **Time to first tuple**

![Graph showing time to first tuple](image)

• **Time to last tuple**

![Graph showing time to last tuple](image)
Web agent experiments

- **Time to first tuple**

- **Time to last tuple**
Distributed database experiments

• **Basic idea**
  – Measure the utility of speculative execution for distributed database queries
  – **Recall: basic query processing**
    • Most commercial relational databases:
      – parse SQL query → build dataflow plan → execute plan

• **TPC-H benchmark**
  – Transaction Processing Council (TPC):
    • Defines database benchmarking queries
  – TPC-H
    • Adhoc business queries for an order-entry schema
Distributed database experiments

- **Modeling**
  - TPC-H schema as a distributed database
Distributed database experiments

```
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
and p_brand = 'Brand#45'
and p_container = 'WRAP CAN'
and l_quantity < (  
    select
        0.2 * avg(l_quantity)
    from
        lineitem
    where
        l_partkey = p_partkey
);  
```

<table>
<thead>
<tr>
<th>SELECT STATEMENT ()</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT (AGGREGATE)</td>
<td>2</td>
</tr>
<tr>
<td>FILTER ()</td>
<td>3</td>
</tr>
<tr>
<td>NESTED LOOPS ()</td>
<td>4</td>
</tr>
<tr>
<td>TABLE ACCESS (FULL)</td>
<td>LINEITEM</td>
</tr>
<tr>
<td>TABLE ACCESS (BY INDEX ROWID)</td>
<td>PART</td>
</tr>
<tr>
<td>INDEX (UNIQUE SCAN)</td>
<td>PART_PK</td>
</tr>
<tr>
<td>SORT (AGGREGATE)</td>
<td>4</td>
</tr>
<tr>
<td>TABLE ACCESS (FULL)</td>
<td>LINEITEM</td>
</tr>
</tbody>
</table>
Distributed database experiments

• **Experiment**
  – Ran 75% of TPC-H queries
    • Queries not run relied on operations that would be time-consuming to support in Theseus
  – Varied latency and database scale
  – Tested on recurring queries
Outline

1. Review and motivating example
2. Speculative plan execution
   3. Value prediction for speculative execution
4. Related work
5. Summary
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

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 ?
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>

**Cache**

**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)?

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

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

Dodge Stratus

http://cg.com/full/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:**

**ANSWERS:**

**Diagram:**
- INSERT("http://cg.com/full/")
- INSERT(".htm")
- TRANSDUCE
- h:ACCEPT u:ACCEPT /:ACCEPT /:COPY /:ACCEPT
- ε:ACCEPT ε:ACCEPT ε:ACCEPT ε:ACCEPT
Experimental results

- Better accuracy than strictly caching

**Hint classification**

![Graph showing accuracy over number of new examples](image)

**Hint transduction**

<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)

<table>
<thead>
<tr>
<th>Number of examples</th>
<th>Space savings (over caching)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.00%</td>
</tr>
<tr>
<td>400</td>
<td>20.00%</td>
</tr>
<tr>
<td>600</td>
<td>40.00%</td>
</tr>
<tr>
<td>800</td>
<td>60.00%</td>
</tr>
<tr>
<td>1000</td>
<td>80.00%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of examples</th>
<th>Space savings (over caching)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>100.00%</td>
</tr>
<tr>
<td>10</td>
<td>100.00%</td>
</tr>
<tr>
<td>100</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
Effect on spec exec performance

- CarInfo
Effect on spec exec performance

• RepInfo
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
Outline

1. Review and motivating example
2. Speculative plan execution
3. Value prediction for speculative execution
4. Related work
5. Summary
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]
  - 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]
Outline

1. Introduction and motivating example
2. Speculative plan execution
3. Value prediction for speculative execution
4. Related work
5. Summary
Summary

• An approach to **speculative execution of information agent plans**
  – Can yield arbitrary speedups
  – Safe, fair

• **Value prediction approach** that combines caching, classification, and transduction
  – More accurate & space efficient than strictly caching
Future work

• Placement of the Confirm operator

• Learning to compute speculative overhead

• Exploring more value prediction strategies
  – Example: Stride value prediction
    • Learning loop increments (e.g., [1,2,3], [2,4,6])
    • Similar to learning ["...page=1", "...page=2"] for URLs

• Predictor compression
  – Probabilistic classifiers

• Speculative execution of other types of agents
  – Example: Robot soccer agents
A final aside… CPU evolution

• For many wonderful years
  – We have been happily writing von Neumann style programs
  – Compilers have been optimizing these programs
    • To extract as much dataflow parallelism as possible
  – We run them on ever-more-powerful CPUs
  – They run fast
    • Speculative execution (branch prediction) yields greatest profit, by far (Wall 1991)

• But now…
  – We’re maxing out
  – Deeper pipelines aren’t much help
Changes in processor architecture

• Limits of ILP

In-order scheduling with perfect memory
(Intel Corporation Research Labs -- ~1998)
Simultaneous Multithreading (SMT)

- Reorganizing chip architecture so that:
  - Multiple functional units can be used per cycle (horiz waste)
  - AND multiple threads can exist (vert waste)
  - AND multiple threads can execute per cycle (>1 PCs & maps)
What does this all mean?

• CPUs running multithreaded code faster

• **Theseus**
  – Streaming dataflow via multiple threads
  – One problem: I/O delays on some of these threads

• **Speculative execution**
  – Allows us to increase the degree of thread level parallelism
  – We can better utilize available resources

• **Greater TLP with SMT processors**
  – Even better efficiency with fewer processors
Thank you
Summary of results

• Increased accuracy (recall)
  – Classification-based predictors
    • Can make correct predictions more often than strictly caching (some errors)
  – Transduction-based predictors
    • Quickly up to 100%!

• Space savings
  – Classification-based predictors
    • Up to 40% over strictly caching, increasing with number of examples
  – Transduction-based predictors
    • Quickly up to 100%!