Using the Theseus Plan Execution System

Greg Barish
CS 548
Feb 1st, 2005
Outline of talk

• Basic Theseus concepts
• Building and running a Theseus plan
• Theseus operators
• Extending Theseus
• System details and configuration options
• Questions
Review: Information gathering plans

• Fetch, manipulate, and combine data
  – Often using multiple sources (data integration)

• Plans consist of a network of operators
  – Each operator like a function
    • Example: Wrapper, Select, etc.
  – Data routed between operators are relations
    • Zero or more tuples with one or more attributes
Review: Efficient plan execution

• Standard techniques
  – **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
Building and running Theseus plans
Theseus

• Composed of
  – An information gathering plan language
  – Execution system

• To use Theseus, you need to know
  – How you can use the language to write plans…
  – …and how to execute the plans that you write

• Theseus requirements
  – Windows NT/2000 or XP
  – Java 1.4.2_06
Downloading & installing Theseus

• Read the release note at:

• What to do:
  – Download theseus.zip
  – Unzip it into any local directory
    • We suggest creating a c:\theseus directory and then unzipping the file from here
      – "theseus350" subdirectory created automatically
  – Make sure that you can run Theseus
    • Try executing the theseus.bat file

% theseus.bat
java theseus.tools.client.CmdLineClient <.plan file> <data file>
Very important

URL:

USERNAME:
theseus

PASSWORD:
info-agents
Theseus directory structure

- **Subdirectories**
  - bin
    - Binary files (currently none)
  - etc
    - Theseus.properties (configuration file)
  - plans
    - plans/examples contains many unit test examples
  - lib
    - All of the JAR files
  - src
    - Example Theseus API code
Designing plans

- To design a plan, you need to
  - Name the plan
  - Identify INPUT and OUTPUT
  - Design dataflow graph for how INPUT $\rightarrow$ OUTPUT

- Example:
  - Suppose you are given 4 sets of book data
    - Title, author, price, etc.
    - For example, from 4 different bookstores
  - and you want to determine
    - The unified set of books, where each is under $10.00
von Neumann style

mybooks (books1, books2, books3, books4)
{
    all = UNION(books1, books2);

    all = UNION(all, books3);

    all = UNION(all, books4);

    affordable = SELECT(all, "price < 10.00");

    return affordable;
}
Dataflow style

Input

books1
books2

Dataflow Graph

UNION
UNION
UNION

Output

SELECT
affordable

books3
books4
mybooks ()
{
    books1 = wrapper("amazon.com")
    books2 = wrapper("barnesnoble.com")
    books3 = wrapper("bookpool.com")
    books4 = wrapper("morebooks.com")
    all = UNION(books1, books2);
    all = UNION(all, books3);
    all = UNION(all, books4);
    affordable = SELECT(all, "price < 10.00");
    return affordable;
}
Fetching - Dataflow style

Input

Dataflow Graph

Output

WRAPPER

UNION

WRAPPER

UNION

WRAPPER

SELECT

UNION

affordable
Writing plans

• To write a plan, you will need to
  – Create a plan file
    • Name the plan
    • Specify INPUT and OUTPUT
    • Translate your dataflow graph (use operators)
  – Create an input file (e.g., books.data)

• Example:
  – books.plan, books.data

• Editing plan and input files
  – Use NOTEPAD, WORDPAD, whatever
Some sample data

• Books (mybooks.data)

# Books1
RELATION books1: title char, author char, pub_date date, pages number, price
Fellowship of the Ring|Tolkien|05-09-1954|733|12.99
Tale of Two Cities|Dickens|09-01-1909|526|8.99
Catcher in the Rye|Salinger|12-23-1948|186|7.99

# Books2
RELATION books2: title char, author char, pub_date date, pages number, price
(etc.)
/* 
* Sample Theseus plan 
* 
*/

PLAN mybooks
{
    INPUT: stream books1, stream books2, 
            stream books3, stream books4 
    OUTPUT: stream affordable 

    BODY 
    { 
        /* Combine books */ 
        union (books1, books2 : tmp1) 
        union (books3, books4 : tmp2) 
        union (tmp1, tmp2 : all) 

        /* Filter out affordable */ 
        select (all, “price < 10” : affordable) 
    } 
}
Running plans

• To run a plan, you use a Theseus client
  – theseus.bat

• Example:
  – % theseus mybooks mybooks.data
  – This also works: % theseus mybooks

• Make sure you edit the THESEUS.BAT file properly and that you call it from the directory that you wish to run plans in
  – cd examples\plans
  – ..\.\theseus uselect1
Writing a subplan

- Subplans in Theseus
  - Encapsulate some functionality
  - Are called just like any other operator

- Suppose you wanted to modularize the example `mybooks` plan
  - `combine`
    - Returns unified set of books
  - `mybooks`
    - Calls `combine` to union all the books, then filters out the affordable ones
PLAN combine {
  INPUT: stream books1, stream books2, 
         stream books3, stream books4 
  OUTPUT: stream all 

  BODY {
    union (books1, books2 : tmp1) 
    union (books3, books4 : tmp2) 
    union (tmp1, tmp2 : all) 
  } 
}

PLAN mybooks {
  INPUT: stream books1, stream books2, 
         stream books3, stream books4 
  OUTPUT: stream affordable 

  BODY {
    combine (books1, books2, books3, books4 : all) 
    select (all, "price < 10" : affordable) 
  } 
}
Theseus Operators
Some sample data

- Books (mybooks.data)

```plaintext
# Sample data
# RELATION books: title char, author char, pub_date
date, pages number, price
#
Fellowship of the Ring|Tolkien|05-09-1954|733|12.99
Tale of Two Cities|Dickens|09-01-1909|526|8.99
Catcher in the Rye|Salinger|12-23-1948|186|7.99
```
Standard relational manipulations

• Select, Project, Antiproject, Join
  – Filter and combine data

```plaintext
select (books, "price < 10" : affordable)
project (affordable, "title" : titles)
join (titles, reviews, "l.title = r.title" : answer)
```

• Union, Intersect, Minus, Distinct
  – Set-theoretic operations

```plaintext
union (d1, d2 : d3)
minus (d3, d2 : d4)
distinct (d4, "title" : d5)
```
Accessing wrappers

- **Xwrapper**
  - Get XML output from wrapper
    ```
    ```
  - Output
    ```
    Attrs = GENRE, AUTHOR, CXML
    Data = Science Fiction|Bradbury| (xml...)
    ```
XML manipulations

• Rel2xml, Xml2rel
  – Converts a relation to XML, vice versa

rel2xml(books, NULL, "xmldoc" : books-xml)

RELATION: urel2xml2_result
  attrs: xmldoc
  
  <OBJECT>
    <ROW>
      <title>Fellowship of the Ring</title>
      <author>Tokien</author>
      <pub_date>07-03-1954</pub_date>
      <pages>536</pages>
    </ROW>
    ...
  </OBJECT>
XML manipulations

- **Rel2xml, Xml2rel**
  - Simple example (for books.data)

```plaintext
PLAN booksdemo {
  INPUT: stream books
  OUTPUT: stream result

  BODY {
    rel2xml (books, NULL, "xml:xml" : x)
    xml2rel (x, "xml:xml", "/OBJECT/ROW", "row" : y)
    antiproject (y, "xml:xml" : result)
  }
}
```

- **Notes**
  - In Xml2Rel, you specify the “path” from which the conversion occurs (e.g., "/OBJECT/ROW") and you specify an “index” so that rows can be given IDs
XML manipulations

• Rel2xml, Xml2rel
  – Simple example (for books.data)

  ----------------------------------------------
  RELATION: uxml2rel2_i1_result
  attrs: row number, pub_date char, pages char,
         title char, author char
  ----------------------------------------------
  0|12-23-1948|186|Catcher in the Rye|Salinger
  1|09-01-1909|526|Tale of Two Cities|Dickens
  2|05-09-1954|733|Fellowship of the Ring|Tolkien
  ----------------------------------------------

  – Notes
  • In Xml2Rel, you specify the “XPath” from which the conversion occurs (e.g., “/OBJECT/ROW”) and you specify an “index” so that rows can be given IDs
XML manipulations

- Rel2xml, Xml2rel
  - Simple example (for books.data)

```xml
PLAN booksdemo {
  INPUT: stream books
  OUTPUT: stream result

  BODY {
    rel2xml (books, NULL, "xml:doc" : x)
    xml2rel (x, "xml:doc", "/OBJECT/ROW", "row" : y)
    antiproject (y, "xml:doc" : result)
  }
}
```

- Commonly used for wrappers...

```xml
xml2rel(x1, "cxml", "/AgentExecution/ExtractedData/Data/Row", "row" : x2)
```
**Xwrapper+Xml2Rel+Antiproject “pattern”**

- Sample plan that queries wrapper

```plaintext
PLAN amazon
{
  INPUT: stream in
  OUTPUT: stream out

  BODY
  {
    xwrapper("http://localhost:8080/agent/runner?plan=amazon/plans/production", "genreName=genre,
    authorName=author", in, "wrapper_data": wrapperout)

    xml2rel(wrapperout, "wrapper_data", "//Data/Row",
    "index" : relout)

    antiproject(relout, "index, wrapper_data" : out)

  }
}
```
Other operators

- **GroupBy**
  - Group data by attributes and/or aggregate measures

```plaintext
groupby (books, "author, MyFunc.sum(pages) sum_pages, MyFunc.average(pages) avg_pages" : result)

----------------------------------------------
RELATION: ugroupby1_result
  attrs: author, sum_pages, avg_pages
----------------------------------------------
Author3|32.0|32.0
Author2|2364.0|788.0
Author1|34.0|34.0
```
Other operators

• Null
  – Conditionally routes data

  \[
  \text{minus (d1, d2 : d3)} \\
  \text{null (d3, my-data, your-data : answer, answer)}
  \]

• Format
  – Create new CHAR attributes based on other attributes

  \[
  \text{format (books, "The author of \%s is \%s", "title, author", "sentence" : formatted)} \\
  \text{project (formatted, "sentence" : answer)}
  \]

  The author of Title5 is Author2
Extending Theseus
Theseus extension: functions

• Performs computation and return a value
  – Input comes from one or more tuples
  – Output is the result of computation

• Functions take parameters
  – Example: ROUND(price)
  – Example: AVG(price)
Single and multi-row functions

• Single row functions
  – Compute a value for each tuple in a relation
  – Example
    • ROUND rounds values based on specified precision
    • **SQL**: `SELECT ROUND(price) FROM order`

• Multi-row (aggregate) functions
  – Compute a value based on a group of tuples
  – Example
    • MAX finds the maximum value of a column
    • **SQL**: `SELECT MAX(price) FROM order`
Writing a Theseus function

1. Decide if it is single-row (APPLY) or multi-row (AGGREGATE) type of computation

2. Write and compile Java function
   - Should be in your CLASSPATH

3. When writing your function
   - APPLY functions take a set of Objects, return an ArrayList
   - AGGREGATE functions take an ArrayList and return either String, int, double
Writing ROUND

PLAN applytest
{
  INPUT: stream books
  OUTPUT: stream result

  BODY
  {
    apply (books, "MyFunc.round(price)", "rounded" : result)
  }
}

RELATION books: title char, price number
Title1|12.376376
Title2|1.87287288
Title3|176.289
Title4|13.376376
Title5|89.72828
import java.util.*;

public class MyFunc
{
    public static ArrayList round (Object a_num)
    {
        ArrayList lst = new ArrayList();
        try {
            double num = Double.parseDouble(a_num.toString());
            lst.add(new Integer((int)Math.round(num)));
        }catch (Exception e) {
            lst.add(new Integer(-1));
        }
        return lst;
    }
}
> theseus applytest

----------------------------------------------
RELATION: applytest_result
  attrs: title, price, rounded
----------------------------------------------
Title3|176.289|176
Title1|12.376376|12
Title4|13.376376|13
Title5|89.72828|90
Title2|1.87287288|2
----------------------------------------------
>
System details and configuration options
How the executor works

• Recall
  – An operator has $\geq 1$ inputs and $\geq 0$ outputs
  – A plan has 1 or more operators

• Runtime representation
  – Operators are classes with input methods
  – A plan is a set of operator class instances

• Execution
  – Threads are used to service operator firings
  – Threads are drawn from a thread pool
    • The bigger the thread pool, the greater the potential degree logical concurrency
    • Physical concurrency depends on how many CPUs you have, what the operators actually do, etc.
Runtime plan –
internal data structure

Operator objects

Routing table

Work queue

Thread Pool

1

2

3

4

5

6
Theseus properties file

• Runtime configuration of the system

• Location of this file is set in theseus.bat, but you can change it as necessary

• Through this file, you can control:
  – Number of threads in thread pool
  – Logging and debugging options
  – Location of resources
  – and more...
Theseus as a Web Application

- It's not difficult to embed Theseus in a servlet
- We've tested deployment on Tomcat 5.0.x

```java
public class TheseusServlet extends HttpServlet {
    public TheseusServlet() {}

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws IOException, ServletException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html><head><title>Plan execution result</title></head>" ];
        out.println("<body> <h1>Plan results</h1><code>");
        long ms = System.currentTimeMillis();
        out.println(runPlan());
        ms = System.currentTimeMillis() - ms;
        out.println("</code>");
        out.println("Plan execution took "+ms+" milliseconds");
        out.println("</body></html>" ];
    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws IOException, ServletException {
        doGet(request, response);
    }
```
Theseus as a Web Application

• It's not difficult to embed Theseus in a servlet
• We've tested deployment on Tomcat 5.0.x

```java
private static String runPlan() {
    StringBuffer result = new StringBuffer();
    try {
        Theseus th = new Theseus();
        LoadedPlan lp = th.loadPlan(new FileReader("c:\mybooks.plan"));
        RelationList inRel = RelationList.load(new FileReader("c:\mybooks.data"));
        RelationList outRel = th.executePlan(lp, inRel);
        if (outRel != null) {
            for (int j=0; j<outRel.size(); j++) {
                result.append(outRel.getRelation(j).toString());
            }
        }
        lp.shutdown();
        th.shutdown();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return result.toString();
}
```
Questions