The Measured Performance of Declarative Approaches to Finding Data in Unstructured Heaps

Gregory M. Kapfhammer

Department of Computer Science
Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

Department of Mathematics and Computer Science
Westminster College, December 2009

In conjunction with William Jones (Allegheny College)
Featuring an image from www.CampusBicycle.com
Overview: Extend and empirically evaluate the efficiency and effectiveness of declarative approaches to finding data in the unstructured heap of a Java virtual machine.
Overview: Extend and empirically evaluate the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine.
Overview: Extend and empirically evaluate the efficiency and effectiveness of declarative approaches to finding data in the unstructured heap of a Java virtual machine.
Overview: Extend and empirically evaluate the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine.
Analysis: Develop and use tree and random forest statistical models and data visualizations that help to identify efficiency and effectiveness trade-offs for data location strategies.
The virtual machine enables platform independence, handles migration, manages limited resources, provides optimization.
The virtual machine enables platform independence, handles migration, manages limited resources, provides optimization.
The virtual machine enables platform independence, handles migration, manages limited resources, provides optimization.
The virtual machine manages resources for the program
The virtual machine manages resources for the program.
The virtual machine manages resources for the program.
The unstructured heap stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)
A memory leak may occur when a Java program incorrectly maintains object references (Xu and Rountev, ICSE 2008)
Why is my program “leaking”? The standard method of iterating through large collections is often challenging and error prone!
JQL: Declaratively Finding Objects

Java Query Language (JQL)

- Features
  - Pre-compilation
  - AOP with AspectJ
  - Method queries
  - Caching
  - Optimizations

- References
  - Willis et al. ECOOP 2006
  - Willis et al. OOPSLA 2008
Java Query Language (JQL)

- **Features**
  - Pre-compilation
  - AOP with AspectJ
  - Method queries
  - Caching
  - Optimizations

- **References**
  - Willis et al. ECOOP 2006
  - Willis et al. OOPSLA 2008
JQL: Declaratively Finding Objects

Java Query Language (JQL)

- **Features**
  - Pre-compilation
  - AOP with AspectJ
  - Method queries
  - Caching
  - Optimizations

- **References**
  - Willis et al. ECOOP 2006
  - Willis et al. OOPSLA 2008
The Measured Performance of Declarative Approaches to Finding Data in Unstructured Heaps
JoSQL: Declaratively Finding Objects

Java Objects SQL (JoSQL)

- **Features**
  - SQL statements
  - String parsing
  - Java reflection
  - Query facilities

- **References**
  - [http://josql.sf.net/](http://josql.sf.net/)
Java Objects SQL (JoSQL)

- **Features**
  - SQL statements
  - String parsing
  - Java reflection
  - Query facilities

- **References**
  - http://josql.sf.net/
Efficiency: Low wind resistance and time to destination
Effectiveness: Transports all required materials and no break downs
Object Query Languages and Bicycles

Cost: Frame material and components cause price to vary considerably
**Benchmarks for Query Languages**

**Features**
- Operations (Query, Join, Sub-Query, Others)
- Objects (Integers, Strings, Graphs, Complex Objects)
- Object and Collection Size (Small, Medium, Large)

**Query Languages**
- JQL 0.3.1 with ANTLR 2.2.7, and AspectJ 1.5
- JoSQL 1.8
- Enhancements
Benchmarks for Query Languages

- **Features**
  - Operations (Query, Join, Sub-Query, Others)
  - Objects (Integers, Strings, Graphs, Complex Objects)
  - Object and Collection Size (Small, Medium, Large)

- **Query Languages**
  - JQL 0.3.1 with ANTLR 2.2.7, and AspectJ 1.5
  - JoSQL 1.8
  - Enhancements
Benchmarks for Query Languages

**Features**
- Operations (Query, Join, Sub-Query, Others)
- Objects (Integers, Strings, Graphs, Complex Objects)
- Object and Collection Size (Small, Medium, Large)

**Query Languages**
- JQL 0.3.1 with ANTLR 2.2.7, and AspectJ 1.5
- JoSQL 1.8
- Enhancements
Analysis Method: Regression Tree Models

Method: HC, JQL

Collection Type: ArrayList, Vector

Mean Value

Tree Models: Use recursive partitioning to create hierarchical view of data
**Analysis Method: Regression Tree Models**

**Explanatory Variable:** Configuration of the benchmark (e.g., “Method”)
Analysis Method: Regression Tree Models

Method: HC, JQL

CollectionType: ArrayList, Vector

Mean Value

Mean Value

Mean Value

Response Variable: One of the evaluation metrics (e.g., “Response Time”)
Analysis Method: Random Forests

CollectionType: ArrayList, Vector

Mean Value

Mean Value

Method: HC, JQL

Mean Value
Analysis Method: Random Forests

Method: HC, JQL

CollectionType: ArrayList, Vector

Mean Value

Mean Value

Mean Value

Mean Value

Mean Value

Mean Value

Mean Value

Mean Value

Mean Value
The Measured Performance of Declarative Approaches to Finding Data in Unstructured Heaps
Analysis Method: Random Forests

Method: HC, JQL
CollectionType: ArrayList, Vector
Mean Value
Mean Value
Method: HC, JQL
CollectionType: ArrayList, Vector
Mean Value
Mean Value
Method: HC, JQL
CollectionType: ArrayList, Vector
Mean Value
Mean Value
Method: HC, JQL
CollectionType: ArrayList, Vector
Mean Value
Mean Value

The Measured Performance of Declarative Approaches to Finding Data in Unstructured Heaps
**Analysis Method: Random Forests**

Many Trees: Randomly construct a large collection of trees in order to avoid bias and identify the most important explanatory variables
Query Benchmark with Integers

Method: HC, JQL
CollectionType: ArrayList, Vector
CollectionSize < 55000
ObjectSize < 550

38.65
309.40
408.50
48460.00 86330.00
Query Benchmark with Integers

Query Benchmark with Integers

Method: HC, JQL

CollectionType: ArrayList, Vector

CollectionSize < 55000

ObjectSize < 550

38.65

309.40

408.50

48460.00 86330.00

Reflection’s Impact: HC and JQL exhibit lower time values than JoSQL
Query Benchmark with Integers

Method: HC, JQL
CollectionType: ArrayList, Vector
CollectionSize < 55000
ObjectSize < 550

38.65
309.40
408.50
48460.00 86330.00

Random Forest: Query method and collection type have most impact
Query Benchmark with Strings

Method: HC,JQL

Collection Type: ArrayList, Vector

Collection Size < 27500

63.75

Collection Size < 275000

218.50

Collection Size < 27500

189.40

Collection Size < 275000

74530.00

120700.00

The Measured Performance of Declarative Approaches to Finding Data in Unstructured Heaps
Reflection’s Impact: HC and JQL exhibit lower time values than JoSQL
Reflection’s Impact: Strings further degrade JoSQL’s performance
Query Benchmark with Strings

Query Benchmark with Strings

Method: HC,JQL

CollectionType: ArrayList, Vector

63.75

218.50

189.40

74530.00 120700.00

CollectionSize < 27500

CollectionSize < 275000

Random Forest: Query method and collection type have most impact
Join Benchmark with Integers and Strings

Method: HC–HJ, JQL

CollectionSize < 2250

<table>
<thead>
<tr>
<th>247.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3651.0</td>
</tr>
<tr>
<td>8447.0</td>
</tr>
<tr>
<td>80720.0</td>
</tr>
</tbody>
</table>

CollectionType: ArrayList, Vector
Join Benchmark with Integers and Strings

Method: HC−HJ,JQL

<table>
<thead>
<tr>
<th>CollectionSize &lt; 2250</th>
</tr>
</thead>
<tbody>
<tr>
<td>247.4</td>
</tr>
<tr>
<td>3651.0</td>
</tr>
<tr>
<td>8447.0</td>
</tr>
<tr>
<td>80720.0</td>
</tr>
</tbody>
</table>

CollectionType: ArrayList, Vector

Reflection’s Impact: HC-HJ and JQL exhibit lower values than JoSQL
Join Benchmark with Integers and Strings

Method: HC–HJ,JQL

CollectionSize < 2250

247.4

CollectionType: ArrayList, Vector

3651.0

8447.0

80720.0

Reflection’s Impact: LinkedList still degrades JoSQL’s performance
Join Benchmark with Integers and Strings

**Random Forest**: Query method and collection type have most impact
# Impact of Object Size on Joining

## Small Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>JQL</td>
<td>57.2</td>
<td>390.2</td>
<td>981.8</td>
</tr>
<tr>
<td>HC-HJ</td>
<td>69.3</td>
<td>378.1</td>
<td>923.5</td>
</tr>
<tr>
<td>JoSQL</td>
<td>997.3</td>
<td>3620.2</td>
<td>8823.1</td>
</tr>
</tbody>
</table>

## Large Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>JQL</td>
<td>35.4</td>
<td>80.8</td>
<td>255.4</td>
</tr>
<tr>
<td>HC-HJ</td>
<td>11.4</td>
<td>63.3</td>
<td>217.8</td>
</tr>
<tr>
<td>JoSQL</td>
<td>930.3</td>
<td>3107.3</td>
<td>8165.9</td>
</tr>
</tbody>
</table>
## Impact of Object Size on Joining

### Small Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Collection Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small</td>
</tr>
<tr>
<td>JQL</td>
<td>57.2</td>
</tr>
<tr>
<td>HC-HJ</td>
<td>69.3</td>
</tr>
<tr>
<td>JoSQL</td>
<td>997.3</td>
</tr>
</tbody>
</table>

### Large Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Collection Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small</td>
</tr>
<tr>
<td>JQL</td>
<td>35.4</td>
</tr>
<tr>
<td>HC-HJ</td>
<td>11.4</td>
</tr>
<tr>
<td>JoSQL</td>
<td>930.3</td>
</tr>
</tbody>
</table>
Incorporate new **benchmarks**, **object types**, and **query languages** in order to better characterize performance. Use **statistical analysis** to make reliable predictions.
Incorporate new benchmarks, object types, and query languages in order to better characterize performance. Use statistical analysis to make reliable predictions.
Incorporate new benchmarks, object types, and query languages in order to better characterize performance. Use statistical analysis to make reliable predictions.
Incorporate new **benchmarks, object types, and query languages** in order to better characterize performance. Use **statistical analysis** to make reliable predictions.
JQL: The Java Query Language

JQL is an extension for Java that provides support for querying collections of objects. These queries can be run over objects in collections in your program, or even used to check expressions on all instances of specific types at runtime.

Queries provide a powerful abstraction for dealing with sets of objects, allowing the query engine to take care of the implementation details. This allows for shorter, clearer code, and permits the query engine to dynamically optimize query evaluation strategies as the runtime context changes.

Queries can also be cached and that cache incrementally maintained - this greatly increases their efficiency, and can offer improved performance for many common collection operations.

A brief example!

Say we're building a crossword puzzle. We've got a list of candidate words for our puzzle, and a list of the lengths of the gaps we need to fill:

```java
ArrayList<String> words = dict.getWords(Puzzle.MEDIUM);
ArrayList<Integer> gapLengths = puzzle.getGapLengths();
```

Now we've got a truly marvelous algorithm for building a crossword puzzle (that this webpage is too narrow to contain), which relies on having a list of pairs of [length, word].

Using a JQL query, we can build this list with ease:

See the Web site of Dr. David J. Pearce for additional resources
JoSQL: Web Site Reference

JoSQL (SQL for Java Objects) provides the ability for a developer to apply a SQL statement to a collection of Java Objects. JoSQL provides the ability to search, order and group ANY Java objects and should be applied when you want to perform SQL-like queries on a collection of Java Objects.

For example, to find all the HTML files that have been modified in December 2004:

```
SELECT *
FROM java.io.File
WHERE name $LIKE "%.html"
AND lastModified BETWEEN toDate (01-12-2004)
AND toDate (31-12-2004)
```

Now to do this in Java code would require the creation of a custom function that will allow the comparison between 2 Java Date objects. Doesn’t sound fun? Perfectly not, but what if you have unsigned to sign...

http://josql.sourceforge.net/ provides tools and documentation
R Language for Statistical Computation

http://www.r-project.org/ provides amazing tools and documentation.
Summary: Extended and empirically evaluated the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine.

http://www.cs.allegheny.edu/~gkapfham/research/

Introduction
Query Languages
Empirical Evaluation

Concluding Remarks

Performance Evaluation
Detailed Empirical Study
Summary: Extended and empirically evaluated the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine.

http://www.cs.allegheny.edu/~gkapfham/research/
Summary: Extended and empirically evaluated the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine. 

http://www.cs.allegheny.edu/~gkapfham/research/
Summary: Extended and empirically evaluated the efficiency and effectiveness of declarative approaches to finding data in the unstructured heap of a Java virtual machine.

http://www.cs.allegheny.edu/~gkapfham/research/
Concluding Remarks

Summary: Extended and empirically evaluated the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine.

http://www.cs.allegheny.edu/~gkapfham/research/