Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps

William F. Jones and Gregory M. Kapfhammer

Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

20th International Conference on Software Engineering and Data Engineering, June 20 - 22, 2011
Overview of the Presentation

Finding Data in Unstructured Heaps
Overview of the Presentation

Finding Data in Unstructured Heaps
Overview of the Presentation

Finding Data in Unstructured Heaps

Challenges

Solutions
Overview of the Presentation

Finding Data in Unstructured Heaps

Challenges

Solutions

Benchmarking Framework
Overview of the Presentation

- Finding Data in Unstructured Heaps
- Challenges
- Solutions
- Benchmarking Framework
  - JQL: Java Query Language

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Overview of the Presentation

Finding Data in Unstructured Heaps

Challenges

Solutions

Benchmarking Framework

JQL: Java Query Language

JoSQL: Java Objects Structured Query Language
Overview of the Presentation

Finding Data in Unstructured Heaps

Challenges

Solutions

Benchmarking Framework

JQL: Java Query Language

JoSQL: Java Objects Structured Query Language

HC: Hand Coded Iterative Methods

Jones & Kapfhammer

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Overview of the Presentation

Finding Data in Unstructured Heaps

Benchmarking Framework

JQL: Java Query Language

JoSQL: Java Objects Structured Query Language

HC: Hand Coded Iterative Methods

Challenges

Solutions

Comprehensive Empirical Study

Jones & Kapfhammer Allegheny College

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Overview of the Presentation

Experiments Reveal Trade-offs in Performance and Overall Viability

Finding Data in Unstructured Heaps

Benchmarking Framework

Challenges

Solutions

Comprehensive Empirical Study

JQL: Java Query Language

JoSQL: Java Objects Structured Query Language

HC: Hand Coded Iterative Methods

Jones & Kapfhammer

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Correctly and Efficiently Finding Objects in the Heap

The unstructured heap in a Java virtual machine stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)
Correctly and Efficiently Finding Objects in the Heap

The unstructured heap in a Java virtual machine stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)

When is an Object Allocated to the Heap?

LinkedList list = new LinkedList()
Correctly and Efficiently Finding Objects in the Heap

The unstructured heap in a Java virtual machine stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)

When is an Object Allocated to the Heap?

```
LinkedList list = new LinkedList()
```

Let’s Allocate Some Objects to the Heap!
Correctly and Efficiently Finding Objects in the Heap

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Correctly and Efficiently Finding Objects in the Heap
Correctly and Efficiently Finding Objects in the Heap

55 → 119 → 5 → 238 → 94 → Ø

Introduction
Query Methods
Empirical Study
Conclusion

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Correctly and Efficiently Finding Objects in the Heap

55 → 119 → 5 → 238 → 94 → ∅

Jones & Kapfhammer
Allegheny College

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Correctly and Efficiently Finding Objects in the Heap

\[ \begin{align*}
55 & \rightarrow 119 \\
5 & \rightarrow 238 \\
94 & \rightarrow \emptyset
\end{align*} \]

\[
\begin{array}{c}
\text{Root} \\
\text{...} \\
22 \\
\text{5} \\
\text{19}
\end{array}
\]

\[
\begin{array}{c}
\text{Start} \\
75 \\
\text{...}
\end{array}
\]

\[
\begin{array}{c}
u_{k-1} \\
w_{k-1} \\
x_{k-1} \\
v_{k-1} \\
z_{k-1}
\end{array}
\]

\[
\begin{array}{c}
u_k \\
w_k \\
x_k \\
v_k \\
z_k
\end{array}
\]

\[
\begin{array}{c}
u_{k+1} \\
w_{k+1} \\
x_{k+1} \\
v_{k+1} \\
z_{k+1}
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
A \\
B \\
B
\end{array}
\]

\[
\begin{array}{c}
H \\
H \\
H \\
H \\
H
\end{array}
\]

\[
\begin{array}{c}
S_0 \\
T_0 \\
S_0 \\
T_0 \\
S_0
\end{array}
\]

\[
\begin{array}{c}
S_1 \\
T_1 \\
S_1 \\
T_1 \\
S_1
\end{array}
\]

\[
\begin{array}{c}
S_2 \\
T_2 \\
S_2 \\
T_2 \\
S_2
\end{array}
\]

\[
\begin{array}{c}
S_3 \\
T_3 \\
S_3 \\
T_3 \\
S_3
\end{array}
\]

\[
\begin{array}{c}
S_4 \\
T_4 \\
S_4 \\
T_4 \\
S_4
\end{array}
\]

\[
\begin{array}{c}
S_5 \\
T_5 \\
S_5 \\
T_5 \\
S_5
\end{array}
\]

\[
\begin{array}{c}
S_6 \\
T_6 \\
S_6 \\
T_6 \\
S_6
\end{array}
\]

\[
\begin{array}{c}
S_7 \\
T_7 \\
S_7 \\
T_7 \\
S_7
\end{array}
\]

\[
\begin{array}{c}
S_8 \\
T_8 \\
S_8 \\
T_8 \\
S_8
\end{array}
\]

\[
\begin{array}{c}
S_9 \\
T_9 \\
S_9 \\
T_9 \\
S_9
\end{array}
\]

\[
\begin{array}{c}
S_{10} \\
T_{10} \\
S_{10} \\
T_{10} \\
S_{10}
\end{array}
\]

\[
\begin{array}{c}
S_{11} \\
T_{11} \\
S_{11} \\
T_{11} \\
S_{11}
\end{array}
\]

\[
\begin{array}{c}
S_{12} \\
T_{12} \\
S_{12} \\
T_{12} \\
S_{12}
\end{array}
\]

\[
\begin{array}{c}
S_{13} \\
T_{13} \\
S_{13} \\
T_{13} \\
S_{13}
\end{array}
\]

\[
\begin{array}{c}
S_{14} \\
T_{14} \\
S_{14} \\
T_{14} \\
S_{14}
\end{array}
\]

\[
\begin{array}{c}
S_{15} \\
T_{15} \\
S_{15} \\
T_{15} \\
S_{15}
\end{array}
\]

\[
\begin{array}{c}
S_{16} \\
T_{16} \\
S_{16} \\
T_{16} \\
S_{16}
\end{array}
\]

\[
\begin{array}{c}
S_{17} \\
T_{17} \\
S_{17} \\
T_{17} \\
S_{17}
\end{array}
\]

\[
\begin{array}{c}
S_{18} \\
T_{18} \\
S_{18} \\
T_{18} \\
S_{18}
\end{array}
\]

\[
\begin{array}{c}
S_{19} \\
T_{19} \\
S_{19} \\
T_{19} \\
S_{19}
\end{array}
\]

\[
\begin{array}{c}
S_{20} \\
T_{20} \\
S_{20} \\
T_{20} \\
S_{20}
\end{array}
\]

\[
\begin{array}{c}
S_{21} \\
T_{21} \\
S_{21} \\
T_{21} \\
S_{21}
\end{array}
\]

\[
\begin{array}{c}
S_{22} \\
T_{22} \\
S_{22} \\
T_{22} \\
S_{22}
\end{array}
\]

Jones & Kapfhammer
Allegheny College
Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Correctly and Efficiently Finding Objects in the Heap

55 → 119 → 5 → 238 → 94 → ∅

Root

... 22

Start

... 75

LinkedList Node(s) with Values Greater Than Those in the Trees

Jones & Kapfhammer

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Correctly and Efficiently Finding Objects in the Heap

55 → 119 → 5 → 238 → 94 → ∅

Root

... 22

Start

75

... 5 19

LinkedList Node(s) with Values Greater Than Those in the Trees
Correctly and Efficiently Finding Objects in the Heap

How Do We Find These Nodes?

Jones & Kapfhammer

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Correctly and Efficiently Finding Objects in the Heap

How Do We Find These Nodes?  Imperative - Give the Procedure
Correctly and Efficiently Finding Objects in the Heap

How Do We Find These Nodes?

Declarative - Give the Specification
Object Query Languages and Bicycles

Efficiency - Bicycle: Low wind resistance and time to destination
Object Query Languages and Bicycles

Efficiency - Query: Minimal space overhead and a low response time
Object Query Languages and Bicycles

Effectiveness - Bicycle: Transports all item(s) with no break downs
Effectiveness - Query: Always returns the correct result(s) to a query
Object Query Languages and Bicycles

Cost - Bicycle: Frame material(s) and components cause price to vary
Object Query Languages and Bicycles

Cost - Query: Must consider installation and development challenges
JQL: Java Query Language

JQL Compiler
JQL: Java Query Language
JQL: Java Query Language

JQL File

JQL Compiler

Java Source Code
JQL: Java Query Language

JQL File

JQL Compiler

Java Source Code

Java Compiler

Jones & Kapfhammer

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
JQL: Java Query Language

JQL File

JQL Compiler

Java Source Code

Java Compiler

Java Bytecodes
JQL: Java Query Language

- JQL File
- JQL Compiler
- Java Source Code
- Java Compiler
- Java Bytecodes
- Collection
JQL: Java Query Language

<table>
<thead>
<tr>
<th>JQL File</th>
<th>JQL Compiler</th>
<th>Java Source Code</th>
<th>Java Compiler</th>
<th>Java Bytecodes</th>
<th>Collection</th>
<th>Query Executor</th>
</tr>
</thead>
</table>

Jones & Kapfhammer

Allegheny College

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
JQL: Java Query Language
JQL: Java Query Language

**Features**
- Pre-compilation
- AOP with AspectJ
- Method Queries
- Caching
- Optimizations
JQL: Java Query Language

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

References
- (Willis et al. ECOOP 2006)
- (Willis et al. OOPSLA 2008)

Flow Diagram:
- JQL File
  - JQL Compiler
  - Java Source Code
    - Java Compiler
    - Java Bytecodes
      - Collection
      - Query Executor
        - Results
        - Cache
JoSQL: Java Objects Structured Query Language

Parse SQL
JoSQL: Java Objects Structured Query Language

Parse SQL

SQL String

Query Object

Jones & Kapfhammer
Allegheny College

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
JoSQL: Java Objects Structured Query Language

- SQL String
- Query Object
- Parse SQL
- Executable Query
- Collection
JoSQL: Java Objects Structured Query Language
JoSQL: Java Objects Structured Query Language

- SQL String
- Query Object
- Parse SQL
- Executable Query
- Collection
- Query Executor
- Query Results
JoSQL: Java Objects Structured Query Language

Features
- SQL Statements
- String Parsing
- Java Reflection
- Query Facilities

SQL String → Parse SQL → Executable Query → Query Executor → Query Results

Query Object → Collection
JoSQL: Java Objects Structured Query Language

**Features**
- SQL Statements
- String Parsing
- Java Reflection
- Query Facilities

**Reference**
- http://josql.sf.net

---

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)
Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

JQL: Compile-Time
Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

**JQL:** Compile-Time

**JoSQL:** Run-Time
Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

**JQL**: Compile-Time

**JoSQL**: Run-Time

Performance Trade-Offs?
Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

**JQL**: Compile-Time

**JoSQL**: Run-Time

Performance Trade-Offs?

Effectiveness Concerns?
Comparison of Data Finding Methods

Benchmarking Framework Helps to Answer These Questions

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

**JQL**: Compile-Time  \n**JoSQL**: Run-Time

Performance Trade-Offs?  \nEffectiveness Concerns?
Benchmarking Framework to Evaluate Query Methods

Random Collection Generator
Benchmarking Framework to Evaluate Query Methods

Random Collection Generator

Benchmark Initializer
Benchmarking Framework to Evaluate Query Methods

- Configuration
  - Random Collection Generator
  - Benchmark Initializer
Benchmarking Framework to Evaluate Query Methods

Configuration

Random Collection Generator
Collection

Benchmark Initializer
Benchmark

Jones & Kapfhammer
Allegheny College
Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Benchmarking Framework to Evaluate Query Methods
Configuration of the Benchmarking Framework

Possible Configurations

- Operations
- Objects
- Sizes
- Methods

Explored a **wide variety** of benchmark configurations
What **operations** do we run to **evaluate** the query methods?
Configuration of the Benchmarking Framework

What operations do we run to evaluate the query methods?
Configuration of the Benchmarking Framework

What operations do we run to evaluate the query methods?
Configuration of the Benchmarking Framework

What objects will we allocate to the JVM’s heap?
Configuration of the Benchmarking Framework

Possible Configurations

Operations
Objects
Sizes
Methods

Parameter Values

What **objects** will we **allocate** to the JVM’s **heap**?
Configuration of the Benchmarking Framework

What **objects** will we **allocate** to the JVM’s **heap**?
Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

Sizes

Methods

How **big** should we make the **objects** and the **collections**?
Configuration of the Benchmarking Framework

Possible Configurations

Operations  Objects  Sizes  Methods

Parameter Values

How **big** should we make the **objects** and the **collections**?
Configuration of the Benchmarking Framework

Possible Configurations

Operations
Objects
Sizes
Methods

Parameter Values

Small
Medium
Large

How **big** should we make the **objects** and the **collections**?
### Configuration of the Benchmarking Framework

<table>
<thead>
<tr>
<th>Possible Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations</td>
</tr>
<tr>
<td>Methods</td>
</tr>
<tr>
<td>Sizes</td>
</tr>
<tr>
<td>Objects</td>
</tr>
</tbody>
</table>

**Which methods should be part of the framework?**
Configuration of the Benchmarking Framework

Possible Configurations

Operations
Objects
Sizes
Methods

Parameter Values

Which **methods** should be part of the **framework**?
Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

Sizes

Methods

Parameter Values

JQL

JoSQL

HC

HC-HJ

Which **methods** should be part of the **framework**?
Configuration of the Benchmarking Framework

Possible Configurations

Operations
Objects
Sizes
Methods

Parameter Values

JQL
JoSQL
HC
HC-HJ

See the paper for further operator and configuration details
Analysis Techniques: Regression Tree Models

Method: HC-HJ, JQL

Tree Models: Recursive partitioning creates hierarchical view of data
Analysis Techniques: Regression Tree Models

Method: HC-HJ, JQL

247.40

CollectionSize < 2250

Tree Models: Recursive partitioning creates hierarchical view of data
Analysis Techniques: Regression Tree Models

Method: HC-HJ, JQL

247.40

CollectionSize < 2250

3651.00

CollectionType: ArrayList, Vector

Tree Models: Recursive partitioning creates hierarchical view of data
Analysis Techniques: Regression Tree Models

Explanatory Variable: Configuration of the benchmarking framework
Non-parametric techniques that handles different variable types
Analysis Techniques: Regression Tree Models

Method: HC-HJ, JQL

Collection Size < 2250

Collection Size

Collection Type: ArrayList, Vector

Variable Types:
- Categorical
- Numerical

Non-parametric techniques that handle different variable types

Jones & Kapfhammer
Allegheny College
Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps
Analysis Techniques: Regression Tree Models

Method: HC-HJ, JQL

CollectionSize < 2250

247.40

CollectionType: ArrayList, Vector

3651.00

Categorical

Response Variable: Response time of the benchmark

Numerical
### Join Benchmark with Integers and Strings

**Method:** HC-HJ, JQL
Join Benchmark with Integers and Strings

Method: HC-HJ, JQL

247.40 < 2250
Join Benchmark with Integers and Strings

Method: HC-HJ, JQL

CollectionSize \leq 2250

CollectionType: ArrayList, Vector

247.40

3651.00
Join Benchmark with Integers and Strings

Method: HC-HJ, JQL

CollectionSize $< 2250$

CollectionType: ArrayList, Vector
Join Benchmark with Integers and Strings

Method: HC-HJ, JQL

CollectionSize ≤ 2250

CollectionType: ArrayList, Vector

247.40

3651.00

8447.00

80720.00
Join Benchmark with Integers and Strings

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

Method: HC-HJ, JQL

CollectionSize < 2250

CollectionType: ArrayList, Vector

Reflection’s Impact: LinkedList further degrades JoSQL’s performance
Impact of Object Size on Joining

<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>
## 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><strong>JQL</strong></td>
<td>57.2</td>
</tr>
<tr>
<td><strong>HC-HJ</strong></td>
<td>69.3</td>
</tr>
<tr>
<td><strong>JoSQL</strong></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><strong>JQL</strong></td>
<td>35.4</td>
</tr>
<tr>
<td><strong>HC-HJ</strong></td>
<td>11.4</td>
</tr>
<tr>
<td><strong>JoSQL</strong></td>
<td>930.3</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><strong>Small</strong></td>
<td></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>
<tr>
<td><strong>Medium</strong></td>
<td></td>
</tr>
<tr>
<td>JQL</td>
<td>390.2</td>
</tr>
<tr>
<td>HC-HJ</td>
<td>378.1</td>
</tr>
<tr>
<td>JoSQL</td>
<td>3620.2</td>
</tr>
<tr>
<td><strong>Large</strong></td>
<td></td>
</tr>
<tr>
<td>JQL</td>
<td>981.8</td>
</tr>
<tr>
<td>HC-HJ</td>
<td>923.5</td>
</tr>
<tr>
<td>JoSQL</td>
<td>8823.1</td>
</tr>
</tbody>
</table>

## Large Objects

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

## Small Objects

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

## Large Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Collection Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>JQL</td>
<td>35.4 80.8 255.4</td>
</tr>
<tr>
<td>HC-HJ</td>
<td>11.4 63.3 217.8</td>
</tr>
<tr>
<td>JoSQL</td>
<td>930.3 3107.3 8165.9</td>
</tr>
</tbody>
</table>
### SubQuery Benchmark with Graphs Containing Strings

![Graph](image.png)

**Technique**
- JQL
- HC
- JoSQL

<table>
<thead>
<tr>
<th>Collection Size</th>
<th>(LinkedList, ObjectSize = 50)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>JQL</td>
</tr>
<tr>
<td>Medium</td>
<td>JQL</td>
</tr>
<tr>
<td>Large</td>
<td>JQL</td>
</tr>
</tbody>
</table>

Time (ms)
- Large: 400 ms
- Medium: 300 ms
- Small: 200 ms

<table>
<thead>
<tr>
<th>Collection Size</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>400</td>
</tr>
<tr>
<td>Medium</td>
<td>300</td>
</tr>
<tr>
<td>Small</td>
<td>200</td>
</tr>
</tbody>
</table>

Jones & Kapfhammer

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps

Allegheny College
SubQuery Benchmark with Graphs Containing Strings

(LinkedList, ObjectSize = 50)

<table>
<thead>
<tr>
<th>Collection Size</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>JQL</td>
</tr>
<tr>
<td>Medium</td>
<td>HC</td>
</tr>
<tr>
<td>Large</td>
<td>JoSQL</td>
</tr>
</tbody>
</table>

JQL is Faster Than HC When the Collection Size is Small and Medium
SubQuery Benchmark with Graphs Containing Strings

(LinkedList, ObjectSize = 50)

<table>
<thead>
<tr>
<th>Collection Size</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>400</td>
</tr>
<tr>
<td>Medium</td>
<td>300</td>
</tr>
<tr>
<td>Small</td>
<td>100</td>
</tr>
</tbody>
</table>

Technique
- JQL
- HC
- JoSQL

HC is Faster Than JQL When the Collection Size is Large
SubQuery Benchmark with Graphs Containing Strings

(LinkedList, ObjectSize = 50)

Collection Size

Small
Medium
Large

Why? JQL Must Track All of the Objects in the Heap
SubQuery Benchmark with Graphs Containing Strings

(LinkedList, ObjectSize = 100)

Trend is Even More Pronounced as the Object Size Increases
Conclusions and Future Work

Concluding Remarks

- Comprehensive empirical study of query methods
- Interesting trends concerning JQL, JoSQL, HC, and HC-HJ
- Refer to the paper for many more insights

Future Work

- Integrate new benchmarks and object types
- Consider different sizes of objects and collections
- Incorporate different data finding methods
- Leverage additional statistical analysis techniques
Conclusions and Future Work

Concluding Remarks

- Comprehensive empirical study of query methods
- Interesting trends concerning JQL, JoSQL, HC, and HC-HJ
- Refer to the paper for many more insights

Future Work

- Integrate new benchmarks and object types
- Consider different sizes of objects and collections
- Incorporate different data finding methods
- Leverage additional statistical analysis techniques
Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps

Thank you for your attention!
Questions?

“Ask, and you will receive. Search, and you will find. Knock, and the door will be opened for you.”
Matthew 7:7 (GWT)  http://bible.cc/matthew/7-7.htm