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

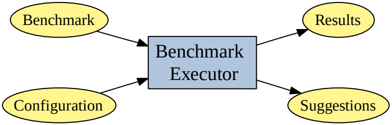
Gregory M. Kapfhammer

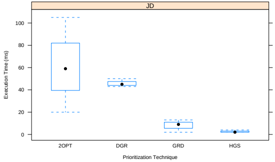
Department of Computer Science
Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

Department of Mathematics and Computer Science
Westminster College, December 2009

In conjunction with William Jones (Allegheny College)
Featuring an image from www.CampusBicycle.com

Important Contributions



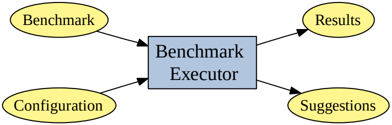


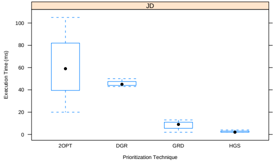
Performance Evaluation

Detailed Empirical Study

Overview: Extend and empirically evaluate the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine

Important Contributions



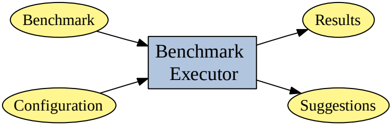


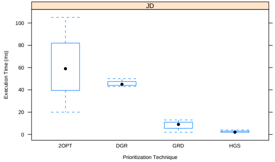
Performance Evaluation

Detailed Empirical Study

Overview: Extend and empirically evaluate the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine

Important Contributions



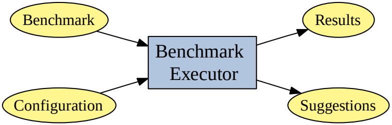


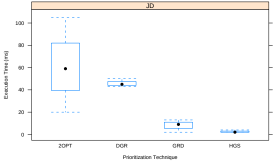
Performance Evaluation

Detailed Empirical Study

Overview: Extend and empirically evaluate the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine

Important Contributions



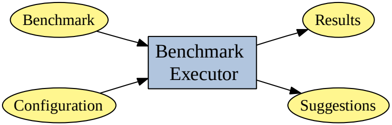


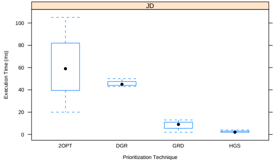
Performance Evaluation

Detailed Empirical Study

Overview: Extend and empirically evaluate the **efficiency** and **effectiveness** of declarative approaches to finding data in the unstructured heap of a Java virtual machine

Important Contributions





Performance Evaluation

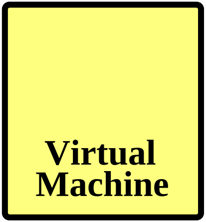
Detailed Empirical Study

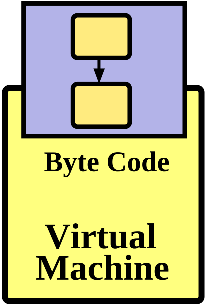
Analysis: Develop and use **tree** and **random forest** statistical models and data visualizations that help to identify efficiency and effectiveness **trade-offs** for data location strategies

The Value of Virtual Machines

The virtual machine enables platform independence, handles migration, manages limited resources, provides optimization

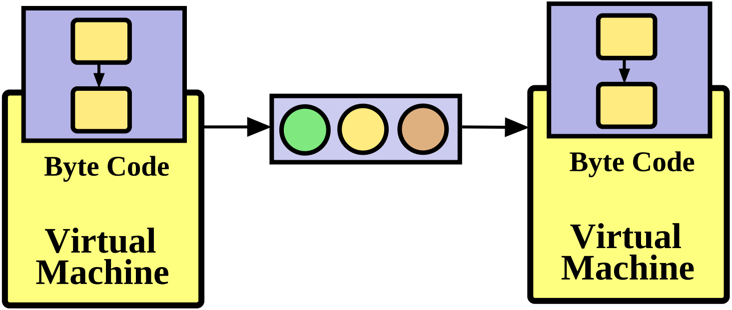
The Value of Virtual Machines





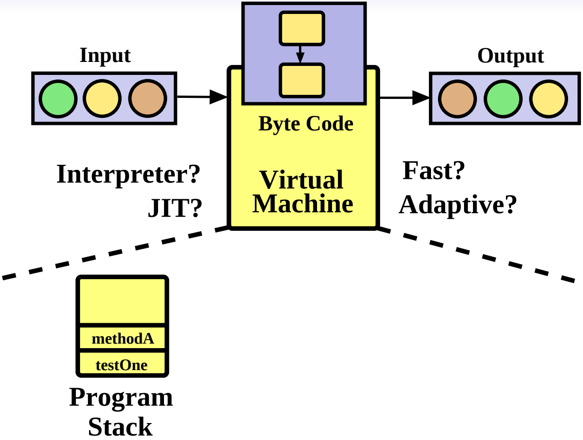
The virtual machine enables platform independence, handles migration, manages limited resources, provides optimization

The Value of Virtual Machines



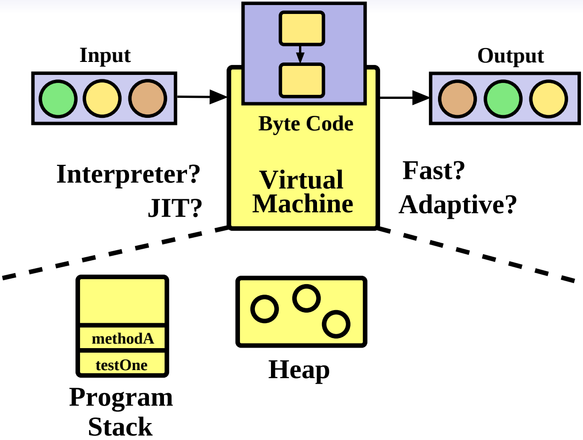
The virtual machine enables platform independence, handles migration, manages limited resources, provides optimization

A Look Inside the Java Virtual Machine



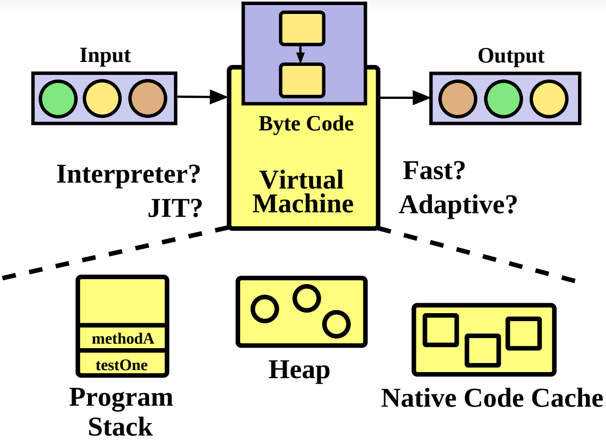
The **virtual machine** manages **resources** for the program

A Look Inside the Java Virtual Machine



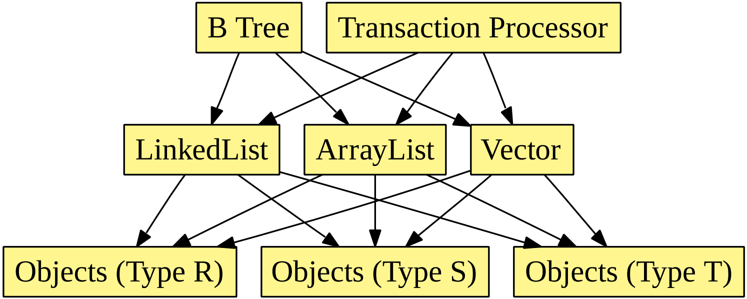
The **virtual machine** manages **resources** for the program

A Look Inside the Java Virtual Machine



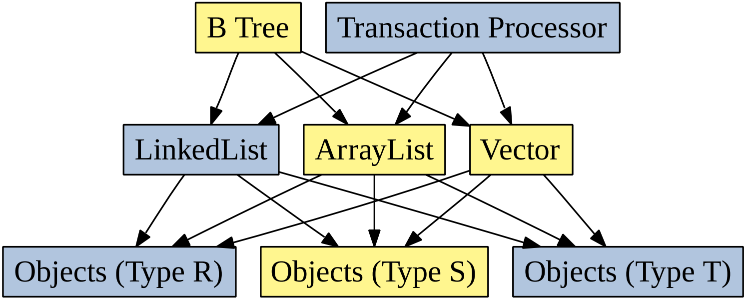
The **virtual machine** manages **resources** for the program

The Container Hierarchy in the Heap



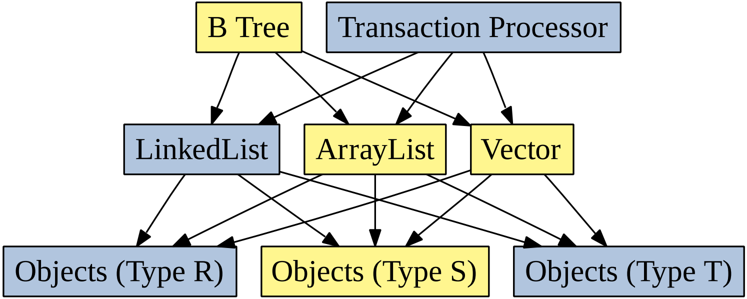
The unstructured heap stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)

The Container Hierarchy in the Heap



A memory leak may occur when a Java program incorrectly maintains object references (Xu and Rountev, ICSE 2008)

The Container Hierarchy in the Heap

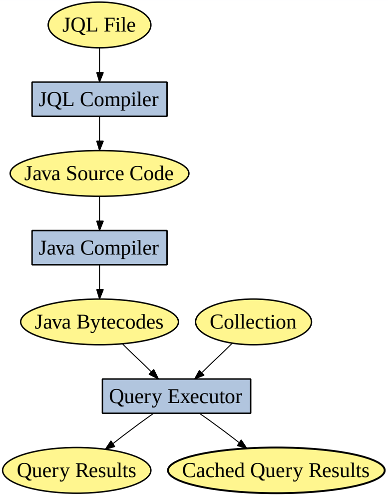


Why is my program "leaking"? The standard method of iterating through large collections is often challenging and error prone!

JQL: Declaratively Finding Objects

Java Query Language (JQL)

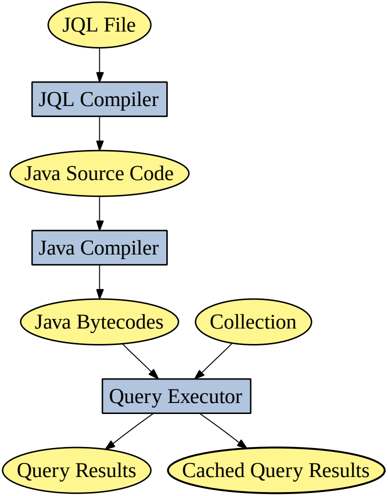
- **Features**
  - Pre-compilation
  - AOP with AspectJ
  - Method queries
  - Caching
  - Optimizations
- **References**
  - Willis et al. ECOOP 2006
  - Willis et al. OOPSLA 2008



JQL: Declaratively Finding Objects

Java Query Language (JQL)

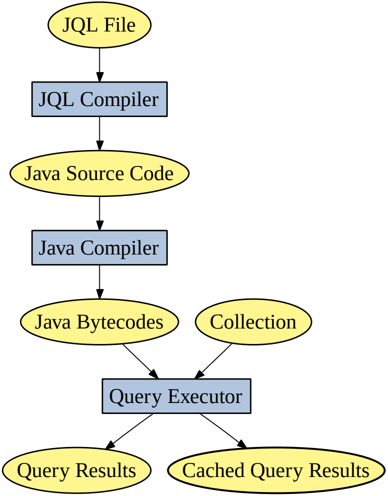
- **Features**
  - Pre-compilation
  - AOP with AspectJ
  - Method queries
  - Caching
  - Optimizations
- **References**
  - Willis et al. ECOOP 2006
  - Willis et al. OOPSLA 2008



JQL: Declaratively Finding Objects

Java Query Language (JQL)

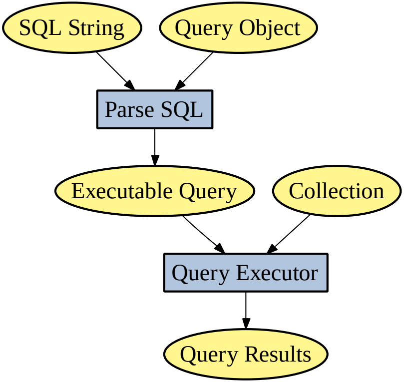
- **Features**
  - Pre-compilation
  - AOP with AspectJ
  - Method queries
  - Caching
  - Optimizations
- **References**
  - Willis et al. ECOOP 2006
  - Willis et al. OOPSLA 2008



JoSQL: Declaratively Finding Objects

Java Objects SQL (JoSQL)

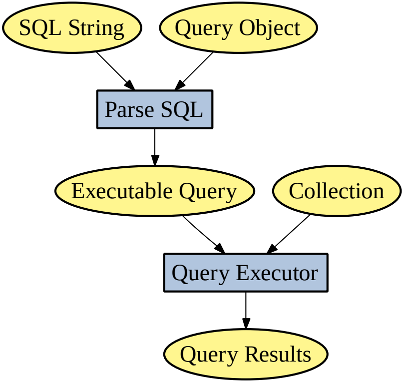
- **Features**
  - SQL statements
  - String parsing
  - Java reflection
  - Query facilities
- **References**
  - http://josql.sf.net/



JoSQL: Declaratively Finding Objects

Java Objects SQL (JoSQL)

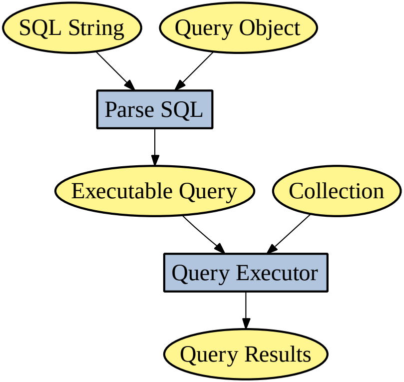
- **Features**
  - SQL statements
  - String parsing
  - Java reflection
  - Query facilities
- **References**
  - http://josql.sf.net/



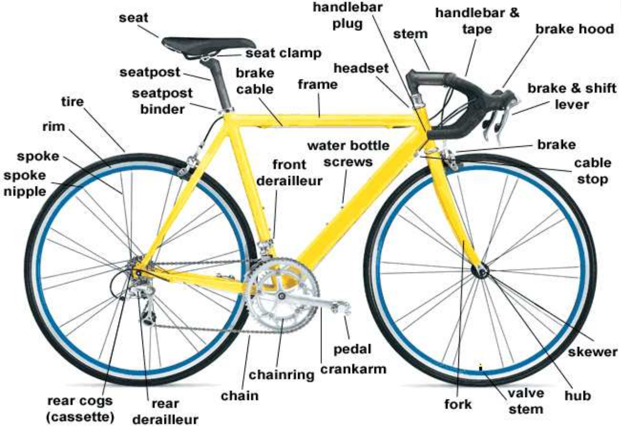
JoSQL: Declaratively Finding Objects

Java Objects SQL (JoSQL)

- **Features**
  - SQL statements
  - String parsing
  - Java reflection
  - Query facilities
- **References**
  - http://josql.sf.net/

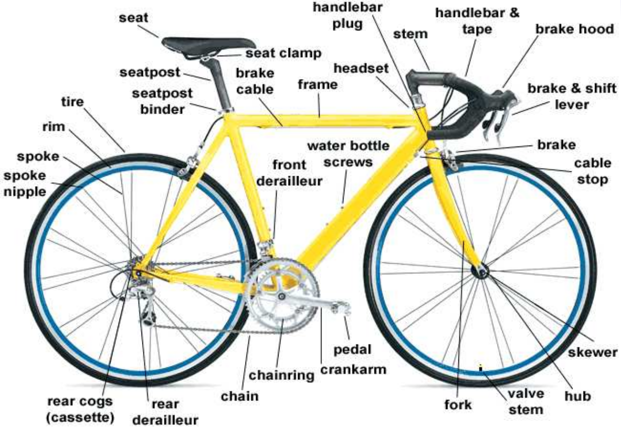


Object Query Languages and Bicycles



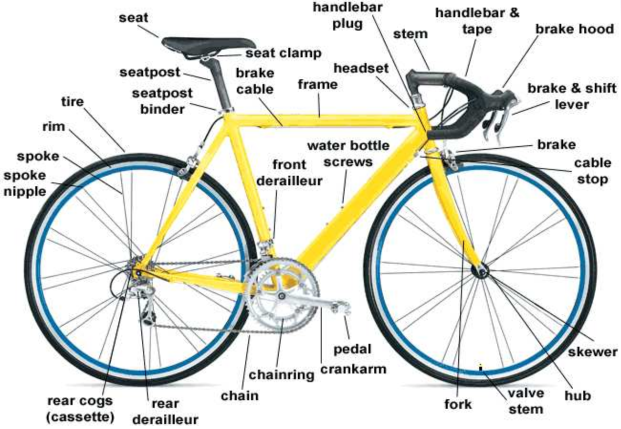
Efficiency: Low wind resistance and time to destination

Object Query Languages and Bicycles



Effectiveness: Transports all required materials and no break downs

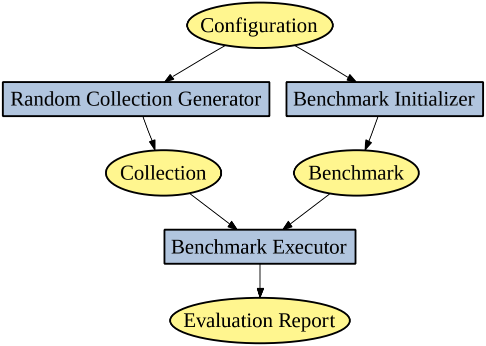
Object Query Languages and Bicycles



Cost: Frame material and components cause price to vary considerably

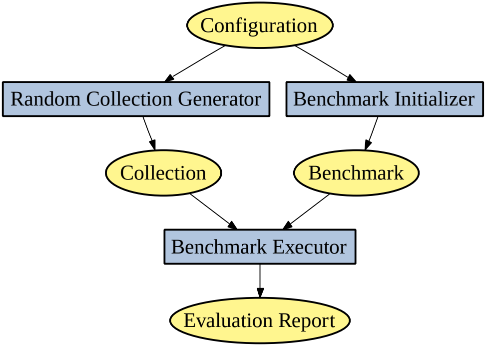
Benchmarks for Query Languages

- **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



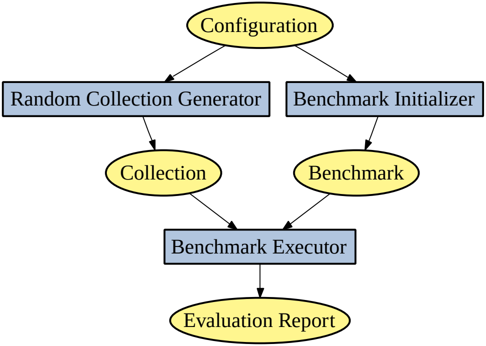
Benchmarks for Query Languages

- **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

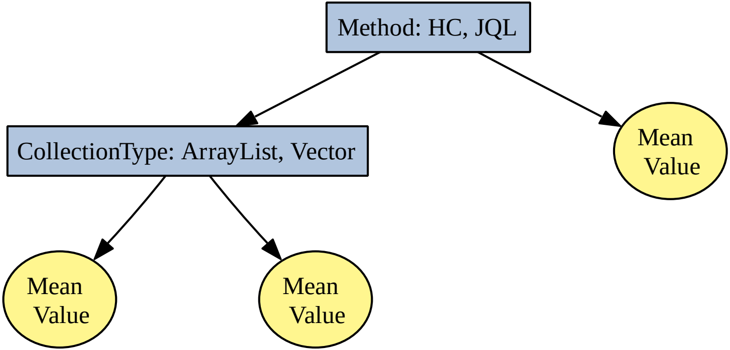


Benchmarks for Query Languages

- **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

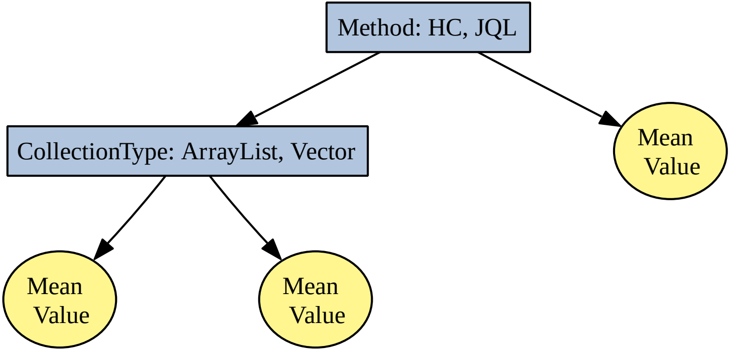


Analysis Method: Regression Tree Models



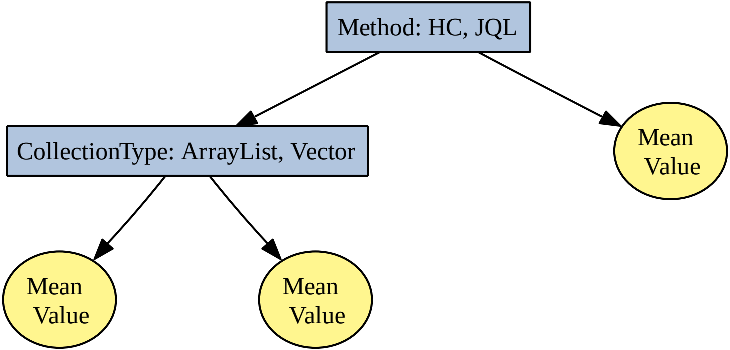
Tree Models: Use recursive partitioning to create hierarchical view of data

Analysis Method: Regression Tree Models



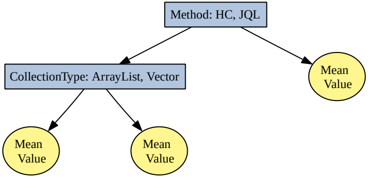
Explanatory Variable: Confi guration of the benchmark (e.g., “Method”)

Analysis Method: Regression Tree Models

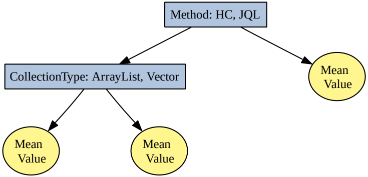


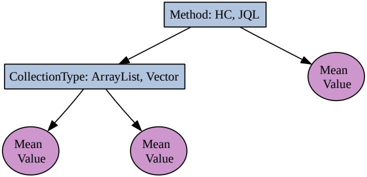
Response Variable: One of the evaluation metrics (e.g., “Response Time”)

Analysis Method: Random Forests

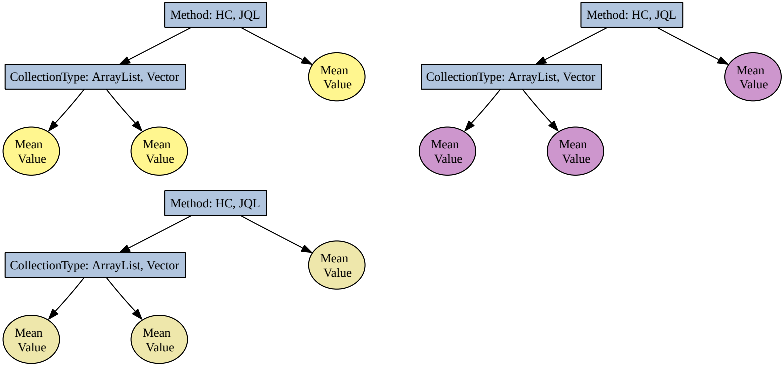


Analysis Method: Random Forests

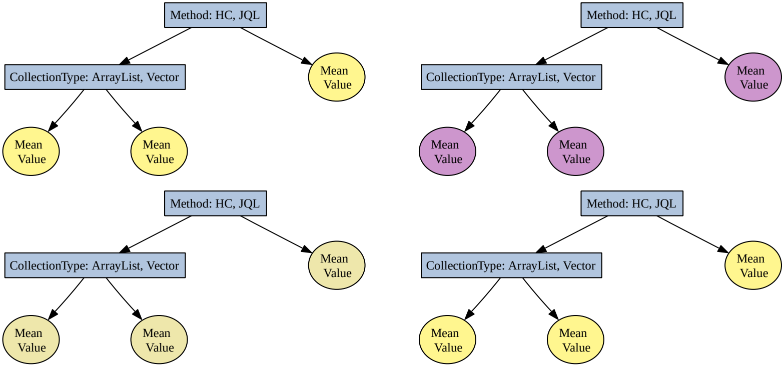




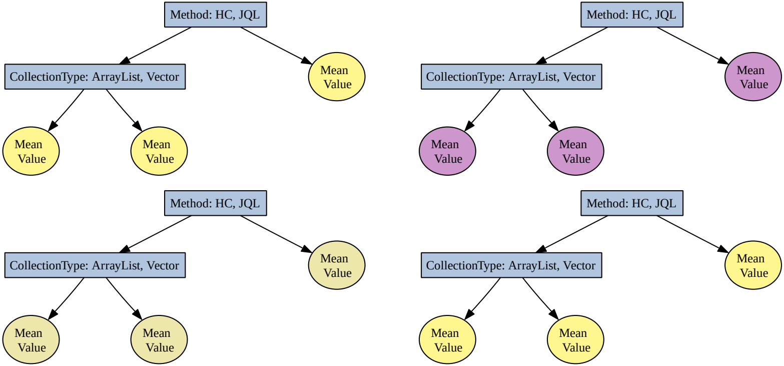
Analysis Method: Random Forests



Analysis Method: Random Forests

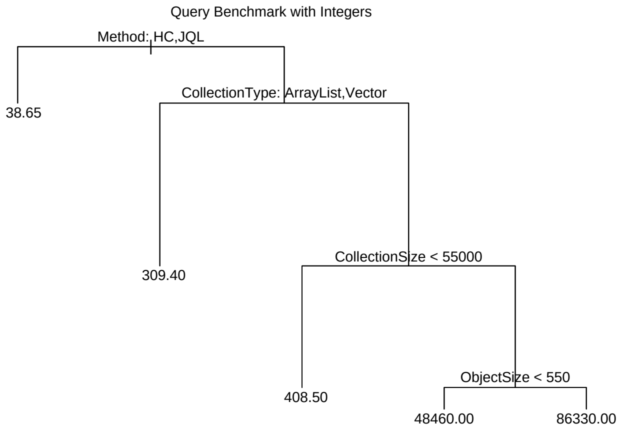


Analysis Method: Random Forests

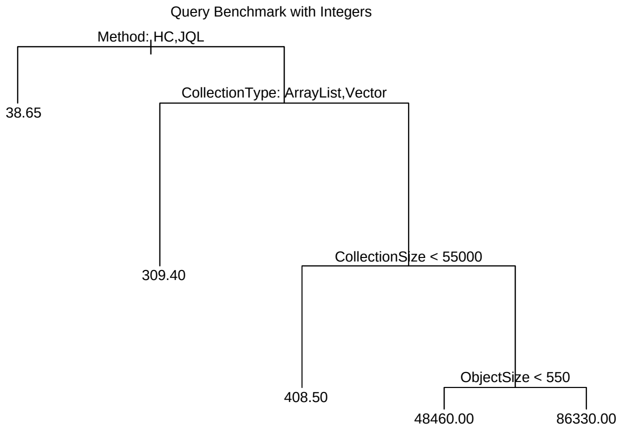


Many Trees: Randomly construct a large collection of trees in order to avoid bias and identify the most important explanatory variables

Query Benchmark with Integers

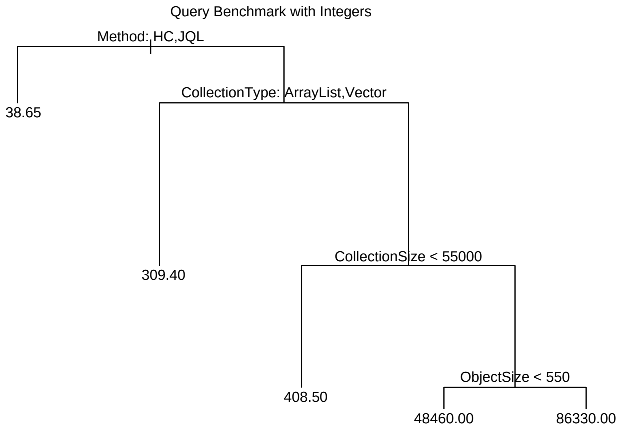


Query Benchmark with Integers



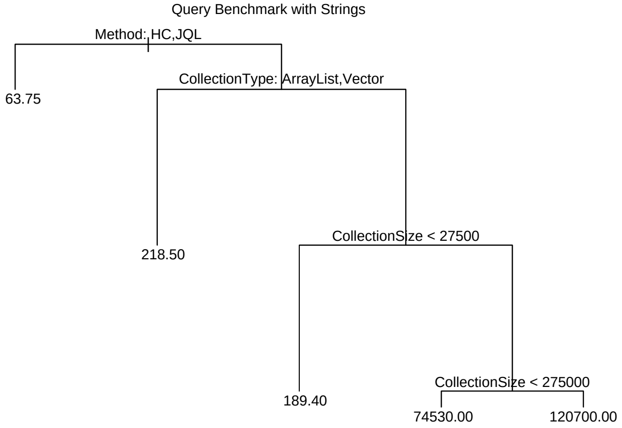
Reflection's Impact: HC and JQL exhibit lower time values than JoSQL

Query Benchmark with Integers

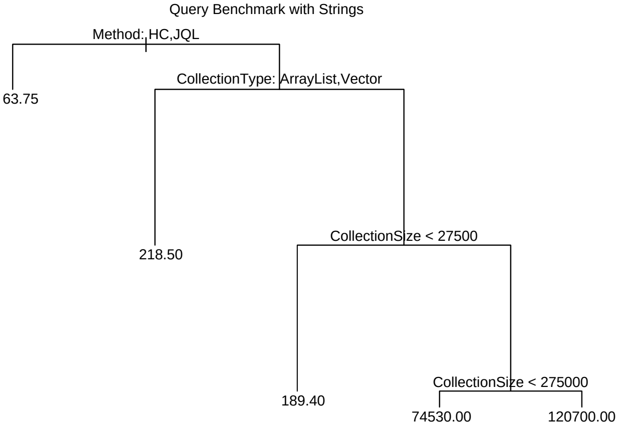


Random Forest: Query method and collection type have most impact

Query Benchmark with Strings

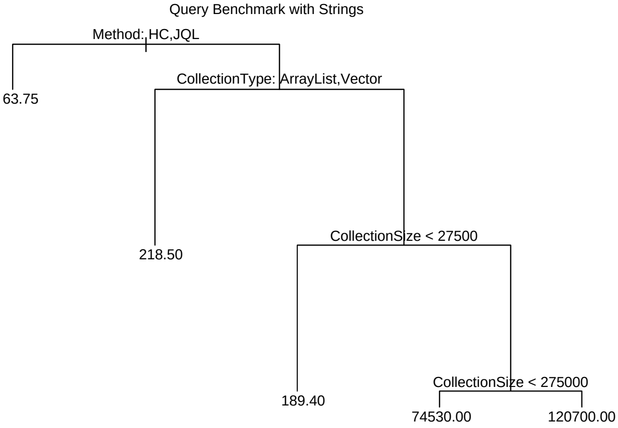


Query Benchmark with Strings



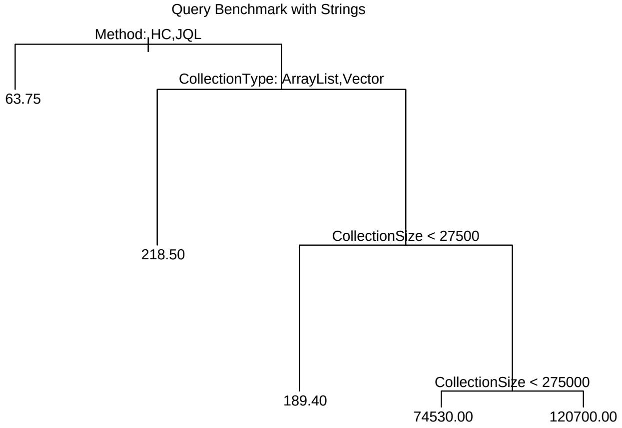
Reflection's Impact: HC and JQL exhibit lower time values than JoSQL

Query Benchmark with Strings



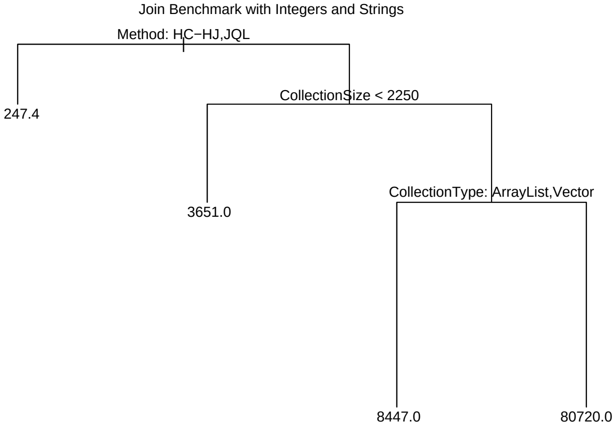
Reflection's Impact: Strings further degrade JoSQL's performance

Query Benchmark with Strings

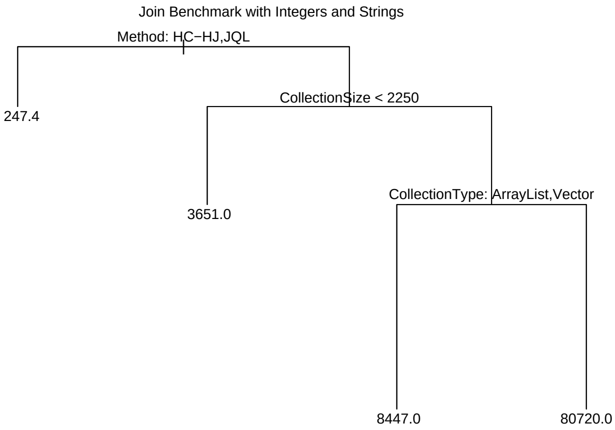


Random Forest: Query method and collection type have most impact

Join Benchmark with Integers and Strings

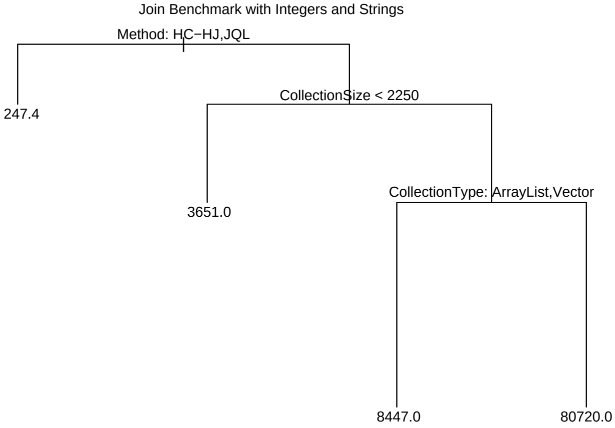


Join Benchmark with Integers and Strings



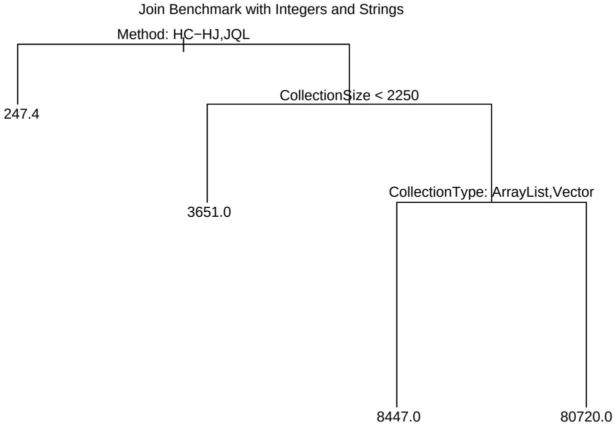
Reflection's Impact: HC-HJ and JQL exhibit lower values than JoSQL

Join Benchmark with Integers and Strings



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

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	57.2	390.2	981.8
HC-HJ	69.3	378.1	923.5
JoSQL	997.3	3620.2	8823.1

Large Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	35.4	80.8	255.4
HC-HJ	11.4	63.3	217.8
JoSQL	930.3	3107.3	8165.9

Impact of Object Size on Joining

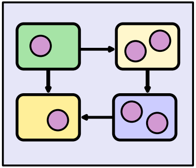
Small Objects

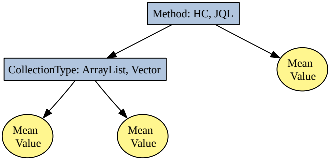
	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	57.2	390.2	981.8
HC-HJ	69.3	378.1	923.5
JoSQL	997.3	3620.2	8823.1

Large Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	35.4	80.8	255.4
HC-HJ	11.4	63.3	217.8
JoSQL	930.3	3107.3	8165.9

Future Work in Performance Evaluation



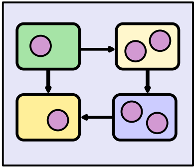


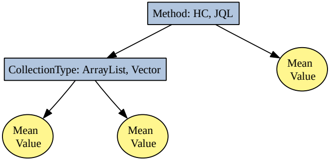
Framework Extension

Statistical Analysis

Incorporate new **benchmarks**, **object types**, and **query languages** in order to better characterize performance. Use **statistical analysis** to make reliable predictions.

Future Work in Performance Evaluation



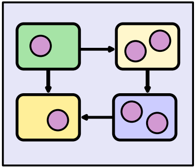


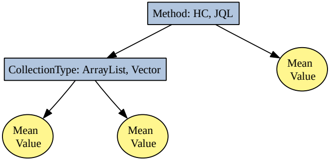
Framework Extension

Statistical Analysis

Incorporate new **benchmarks**, **object types**, and **query languages** in order to better characterize performance. Use **statistical analysis** to make reliable predictions.

Future Work in Performance Evaluation



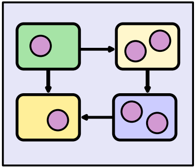


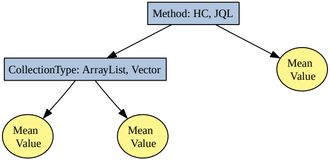
Framework Extension

Statistical Analysis

Incorporate new **benchmarks**, **object types**, and **query languages** in order to better characterize performance. Use **statistical analysis** to make reliable predictions.

Future Work in Performance Evaluation





Framework Extension

Statistical Analysis

Incorporate new **benchmarks**, **object types**, and **query languages** in order to better characterize performance. Use **statistical analysis** to make reliable predictions.

JQL: Web Site Reference

JQL

Java Query Language

About JQL
Examples
Download
Development
Caching
Papers
Help

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!

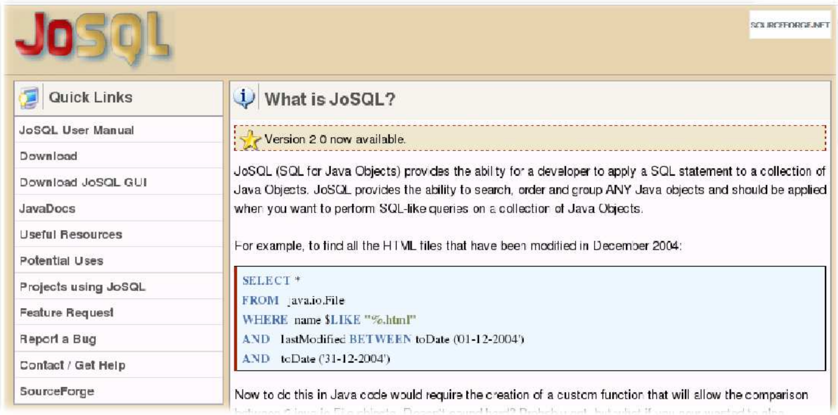
Say we're building a crossword puzzle. We've got a list of candidate words for our puzzle, and a list of the lengths of the gaps we need to fill:

```
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

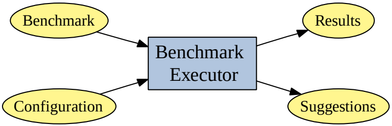


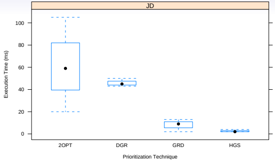
http://josql.sourceforge.net/ provides tools and documentation

R Language for Statistical Computation

http://www.r-project.org/ provides amazing tools and documentation

Concluding Remarks





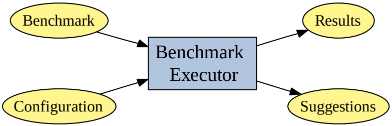
Performance Evaluation

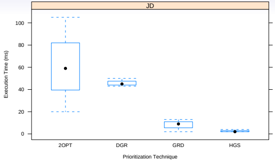
Detailed Empirical Study

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/

Concluding Remarks



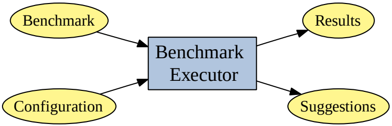


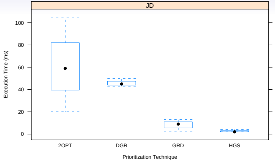
Performance Evaluation

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/

Concluding Remarks





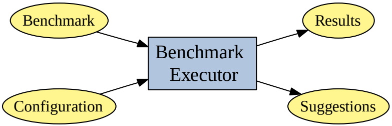
Performance Evaluation

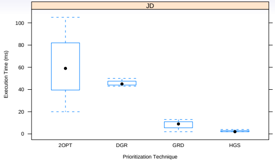
Detailed Empirical Study

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/

Concluding Remarks





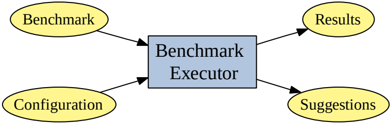
Performance Evaluation

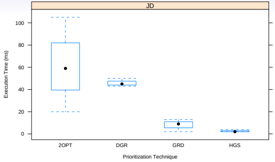
Detailed Empirical Study

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/

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/