Concurrency Benchmark Suite and Metrics for Java Programs

Chenwei Wang, Eric Aubanel, David Bremner, Michael Dawson
University of New Brunswick, IBM Canada
Faculty of Computer Science
vicky.wang@unb.ca, aubanel@unb.ca,
bremner@unb.ca, Michael_Dawson@ca.ibm.com

Abstract

Java benchmarks available today may not reflect the highly concurrent applications we anticipate in the future and none of these benchmarks are dedicated to concurrency measurement. Existing Java benchmarks either report throughput or execution time and behave like a black box without telling users how threads interact with each other, therefore they lack the information of what kind of concurrency patterns are involved. In order to measure what concurrency patterns are used in Java concurrency benchmarks, a range of concurrency pertinent metrics are needed to assess the performance and different concurrency behaviors of Java programs, telling how threads interact with shared memory and communicate with each other.

Metrics

In order to characterize concurrency Java programs, we are using following fine-grained metrics.

Concurrency Thread Metrics:

- Thread Density: tells how many threads do a meaningful amount of work.
- Periodic Thread Density: measures how many threads contribute to the workload concurrently.

Shared Memory Metrics:

- Shared Read Rate: measures read operations on shared objects per second.
- Shared Write Rate: measures write operations on shared objects per second.
- Alternating Modification Rate: measures the rate which the ownership of a shared object is changed.
- Thread Density / shared: computes the thread density for objects that have become shared.
- Periodic Thread Density / shared: measures how many threads contribute to the shared objects concurrently.

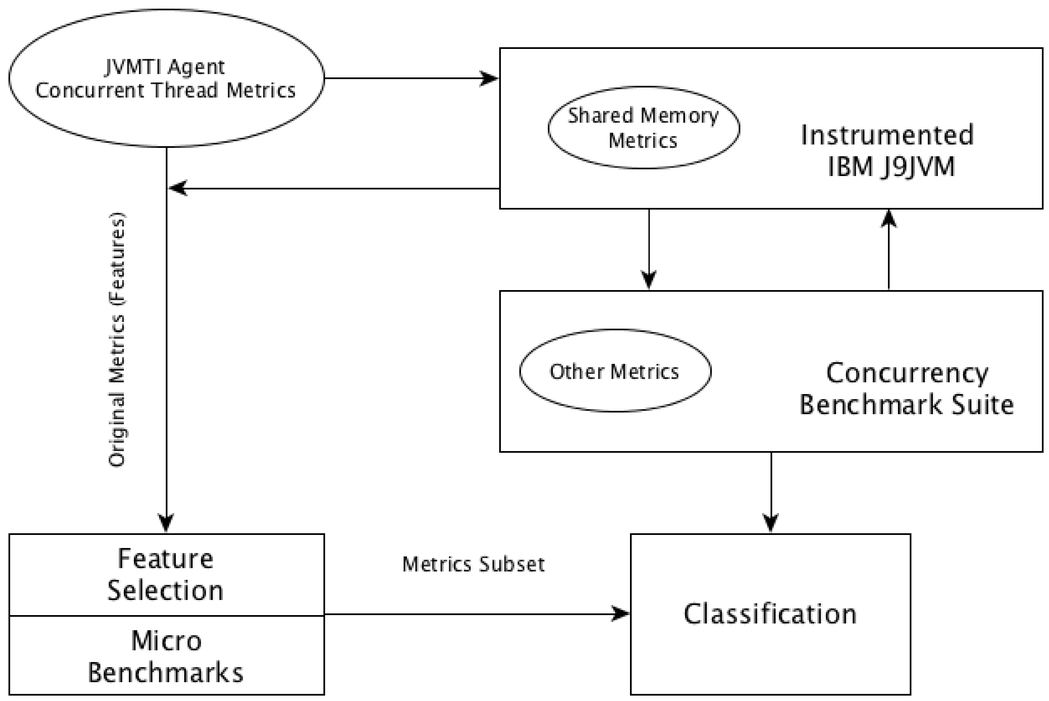
Other Metrics:

- Complexity of benchmarks
- Data Structures
- Dynamic Memory Use

Project Overview

The following figure shows the overview of the concurrency benchmark suite, and how we will get metrics to assess each sub-benchmark. Our concurrency benchmark suite is developed on the IBM J9JVM and there are three different ways to get all the metrics we proposed:

concurrency thread metrics can be obtained by JVMTI (JVM tool interface) agent; shared memory metrics can be acquired by JVM instrumentation; other metrics can be gained from the benchmark suite itself. We will use micro benchmarks to select the most useful metrics and use micro benchmarks' training data to classify the real world applications in our final benchmark suite to make sure it covers different concurrency patterns.



Micro Benchmarks:

The first step is to use a range of micro benchmarks where each covers a different parallel algorithm and a program structure.

Algorithm	Program Structure	Application
Task parallelism	Loop Parallelism + Master/Worker	Image-construction program
Divide and conquer	Fork/Join	Merge sort, Matrix diagonalization
Geometric Decomposition	Loop Parallelism	Matrix multiplication
Pipeline	Fork/Join	Fourier-transform computations with 3 stages
Event-Based Coordination	Fork/Join	Discrete event simulations

Feature Selection:

An exhaustive feature selection algorithm is used to select the most pertinent metrics subset. Best First Search algorithm will be used to reduce the search space.

Classification:

After the metrics subset is identified and the training data from micro benchmarks is obtained, the next step is to classify real world applications in our benchmark suites into different patterns. We consider two classification algorithms: K-Nearest Neighbor and Decision Tree.