Updating Gin's Profiler for Current Java

Myles Watkinson, Sandy Brownlee

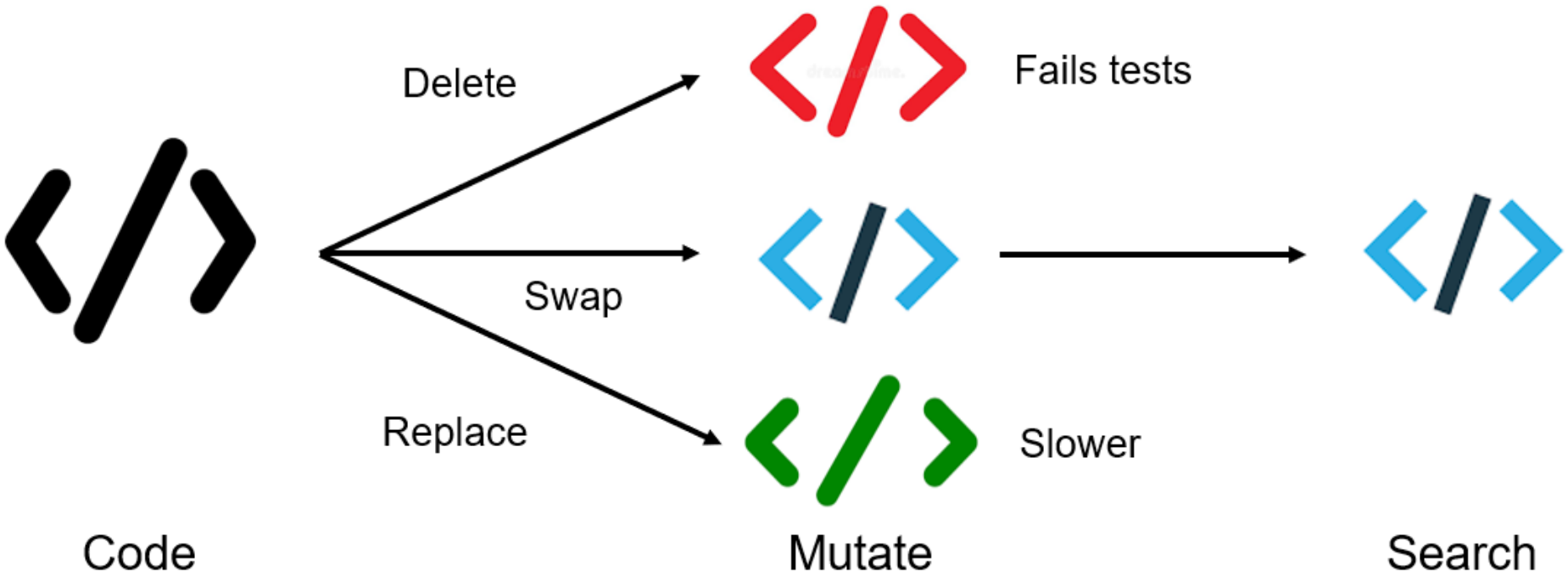
Presentation Structure

1. Explaining what Gin is
2. What role does a profiler play in a GI framework
3. Selecting a profiler for Gin
4. Comparing the new profiler to the old
5. Conclusion

Gin: A Toolbox for Genetic Improvement

- Created to stimulate genetic improvement research
- Designed to be simple and understandable
- Implementations of common edits, build pipeline, testing, speed and memory measurement, and profiling

How GI (in Gin) Works



Gin is stuck in Java 8

HPROF profiler

Only compatible in Java 8 and below

Gin

Java 8

Gin

Java 9 onwards

Reflection security

Only able to reflect easily in Java 8 and below

Why so much effort on the profiler?

- GI to improve runtime of Java code while retaining functionality
  - edits are targeted at “hot methods”
  - these are where the CPU spends the most time
- Profiler:
  - selects the hot methods
  - determines the order the hot methods are ranked
- Need to consider how the profiler plugs into Gin and is used by it

Choosing a new Profiler: Criteria

- The profiler should plug straight into Gin as HPROF did

- The profiler should produce a similar output to that below that can be read and utilised by Gin

```
CPU TIME (ms) BEGIN (total = 206770) Fri Jul 15 07:03:29 2011
rank   self  accum   count trace method
   1 26.23% 26.23%  575160 306454 com.lahti.game.gomoku.Line.checkBoard
   2 18.17% 44.41% 4026120 306447 com.lahti.game.gomoku.Board.safeCheckPiece
   3  9.39% 53.80%   28758 306475 com.lahti.game.gomoku2.EvaluationL2x.evalMove
   4  5.81% 59.61% 4026120 306446 com.lahti.game.gomoku.Board.squareId
   5  4.73% 64.34% 3194498 306451 com.lahti.game.gomoku.Board.getBoardArray
   6  4.56% 68.90% 3194498 306450 com.lahti.game.gomoku.Board.getSquareMaxId
   7  3.07% 71.97%  337192 306470 com.lahti.game.gomoku.LinePatternTable.encodeAsInt
   8  1.77% 73.74%  337192 306461 com.lahti.game.gomoku.Evaluation.getCountForLine
   9  1.76% 75.50%  337136 306465 com.lahti.game.gomoku2.EvaluationL2x.getLineKind
  10  0.96% 76.46%  170850 306474 com.lahti.game.gomoku2.EvaluationL2x.getIntersectionScore
  11  0.86% 77.32%  604106 306458 com.lahti.game.gomoku.Evaluation.getDebug
  12  0.86% 78.18%  575160 306449 com.lahti.game.gomoku.Line.setEvalPt
  13  0.85% 79.03%  575160 306452 com.lahti.game.gomoku.Line.setValueAfter
  14  0.85% 79.88%  575160 306457 com.lahti.game.gomoku.Line.setEval
  15  0.84% 80.72%  575160 306453 com.lahti.game.gomoku.Line.setOnBoard
  16  0.83% 81.55%  575160 306442 com.lahti.game.gomoku.Line.getValuesArray
  17  0.82% 82.37%  575160 306456 com.lahti.game.gomoku.Line.setLineKind
  18  0.82% 83.19%  575160 306443 com.lahti.game.gomoku.Line.getXPointsArray
```

Choosing a new Profiler: Criteria

- Input and output needs to be handled automatically by Gin, no visual interfaces.
- Low overhead is needed as running all unit tests may take time. It is preferable that a profiler adds a little time as possible.
- Gin is a research tool. External profilers used should be free and simple to use.
- The profiler needs to accurately count running functions from the specific codebase.

Candidates

Visual interface:

VisualVM, Java Mission Control, NetBeans profiler

Cost Associated:

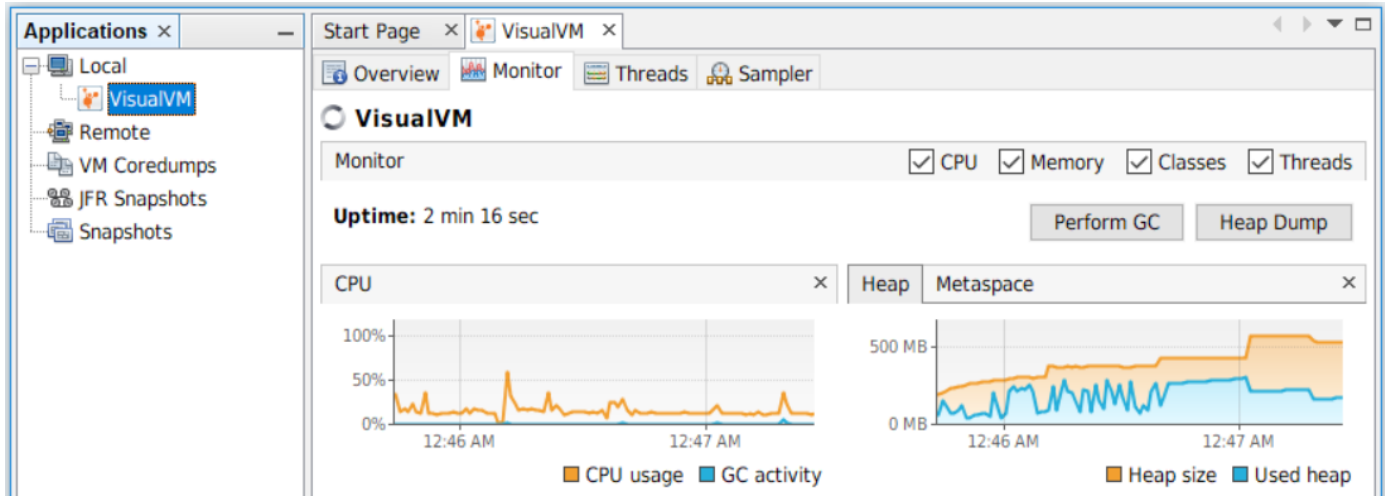
JProfiler

Large Overhead:

JConsole

Potential Profiler:

Java Flight Recorder (JFR)



Integrating JFR into Gin

HPROF

- Outputs to .txt file
- Gives most commonly seen methods
- Skips Java language functions
- Profiles all threads

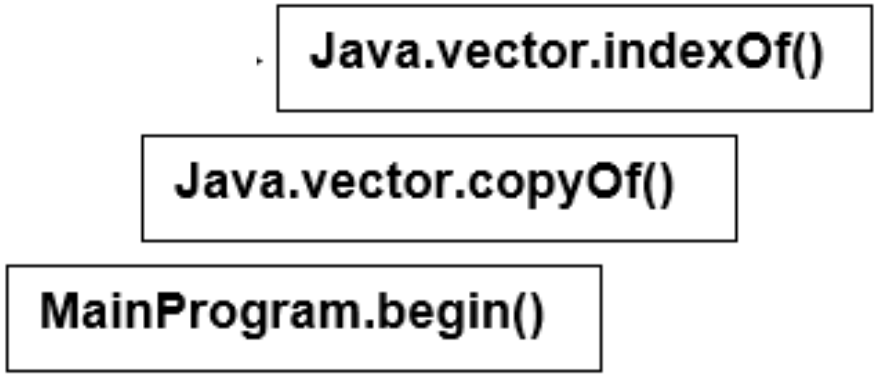
JFR

- Outputs to .jfr file
- Gives call stack
- Call stacks contain all Java language functions
- Doesn’t profile sleeping or waiting threads

Processing JFR call stacks

HPROF pre-processes call stacks and only outputs functions from the program being run most commonly seen in the call stack

JFR outputs a raw call stack which often contains Java language functions



On top of call stack before processing

Function added to profiled method count

Experiments

Two experiments were run to compare HPROF and JFR

1. Profiling a set of simple functions that calculate primes

2. Profiling a more realistic program

Profiling prime number calculations

Only 1 function running at a time with an understanding of how the calculation of primes scales with time

Calculating 5,000 through to 25,000 primes, taking raw time and method count

Prime calculation profiling results

Primes found	Time taken (ms)	Call stack samples found with HPROF	Call stack samples found with JFR
5,000	24	1.25	1.75
10,000	75	5.75	4
15,000	153	12.5	8.5
20,000	266	23	14.75
25,000	412	34	22.75
30,000	588	54.5	32.25

Reasons for different profiling results

Different Java versions?

- HPROF and JFR had to be run in Java 8 and 9 respectively, although, there was almost no difference in the runtime between each version

Thread in a state not profilable by JFR?

- JFR omits samples if the thread sampled is in a WAITING, SLEEPING or BLOCKED state. Although, the program simply adds numbers to a vector if they are primes, there is no waiting or sleeping done in the program.

Further Investigation

When profiling this code:

```
long start = System.currentTimeMillis();
Long now = 0;
while (now < 2000)
{
    Now = System.currentTimeMillis() - start;
    //JFR doesn't profile the above system call
}
```

HPROF consistently returns 127 samples

JFR returns between 10 and 20

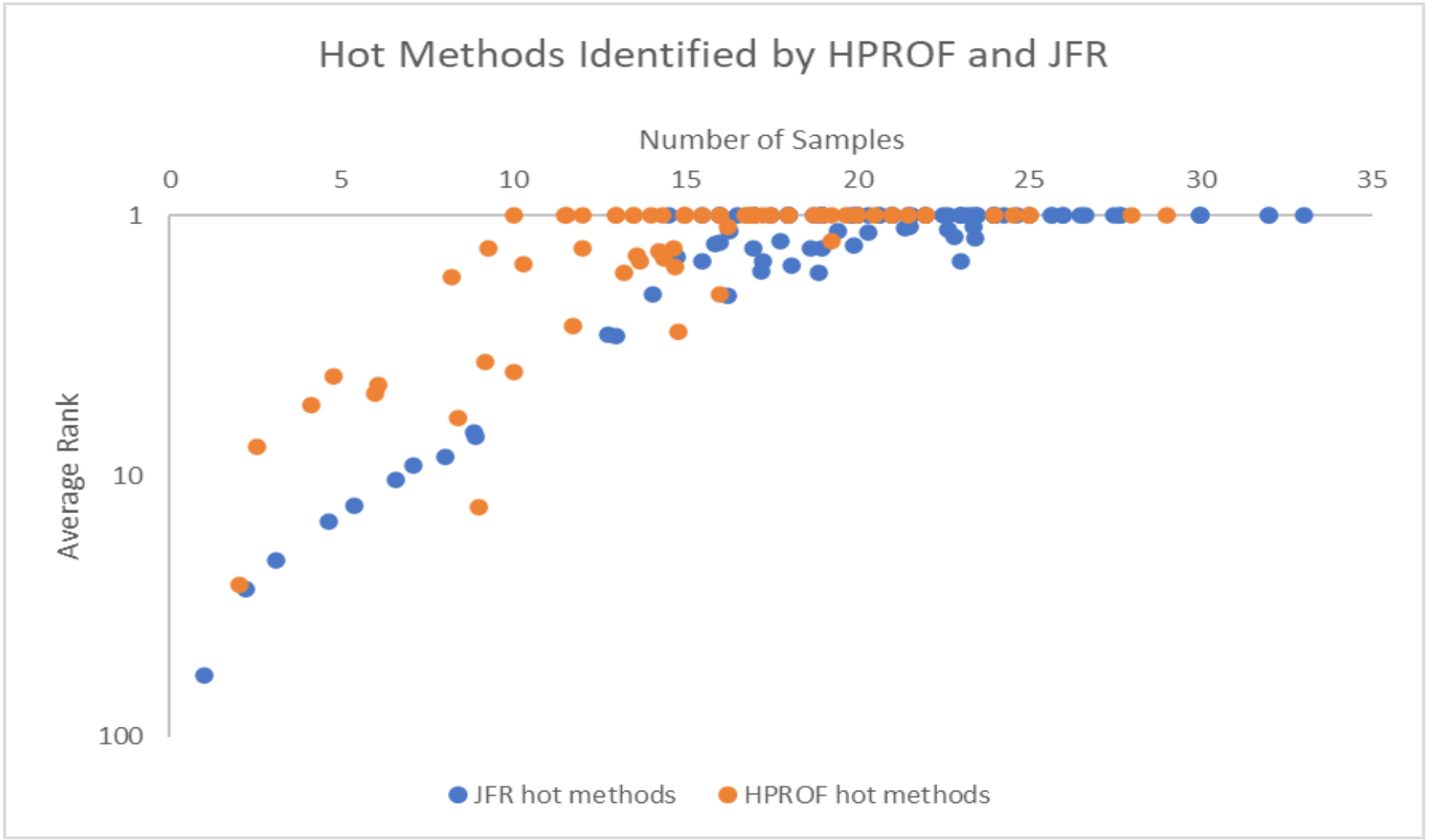
Profiling a more realistic program

The program profiled was Spark, a Java web framework

https://github.com/perwendel/spark

The standard Gin interface was used, Sparks unit tests were run and profiled to produce a hot method summary

Spark profiling results



Spearman Coefficient for top 10 JFR methods and corresponding HPROF methods: **0.29**

Spearman Coefficient for top 20 JFR methods and corresponding HPROF methods: **0.8**

Conclusion

- ➔ Gave an overview of the importance of a profiler in a GI framework
- ➔ Proposed a set of criteria for selecting a profiler
- ➔ Ran experiments to compare two profilers
- ➔ Integrated this profiler into Gin to boost it into current Java versions retaining its efficacy as a tool for GI research

Any Questions?

Link to Gin repository: https://github.com/gintool/gin
Email: myleswatkinson1@gmail.com