VM Warmup Blows Hot and Cold

Edd Barrett, Carl Friedrich Bolz, Rebecca Killick (Lancaster), Vincent Knight (Cardiff), Sarah Mount, Laurence Tratt

Software Development Team
2016-07-18
What’s ‘Warmup’?
What’s ‘Warmup”?

CONSTANT PERFORMANCE OVER TIME

ITERATION TIME

IN-PROCESS ITERATION
With a JIT
With a JIT

IDEALISED JIT WARMUP

ITERATION TIME

COMPILATION

PROFILING INTERPRETER

IN-PROCESS ITERATION
With a JIT

Idealised JIT Warmup

Compilation

Profiling Interpreter

Peak Performance

Iteration Time

In-Process Iteration
With a JIT

IDEALISED JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

← WARMUP →
MORE REALISTIC VM WARMUP
MORE REALISTIC VM WARMUP

ITERATION TIME

SOME NOISE

IN-PROCESS ITERATION
MORE REALISTIC VM WARMUP

ITERATION TIME

IN-PROCESS ITERATION

COMPILATION TIERS

SOME NOISE
Why care?
Users hate noticeable warmup.

Iteration time vs in-process iteration.

- Frustrating
- Happy days!
VM AUTHORS HATE ALL WARMUP
Warmup is bad for everyone.
The Warmup Experiment

Measure warmup of modern language implementations
Measure warmup of modern language implementations

*Hypothesis*: Small, deterministic programs exhibit classical warmup behaviour
The language benchmark games are perfect for us (unusually)
Method 1: Which benchmarks?

The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism
Method 1: Which benchmarks?

The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism

We added checksums to all benchmarks
Method 2: How long to run?

2000 *in-process iterations*
Method 2: How long to run?

2000 in-process iterations

10 process executions
Method 3: VMs

- Graal-0.13
- HHVM-3.12.0
- JRuby/Truffle (git #f82ac771)
- Hotspot-8u72b15
- LuaJit-2.0.4
- PyPy-4.0.1
- V8-4.9.385.21
- GCC-4.9.3

Note: same GCC (4.9.3) used for all compilation
Method 4: Machines

- Linux-Debian8/i4790K, 24GiB RAM
- Linux-Debian8/i4790, 32GiB RAM
- OpenBSD-5.8/i4790, 32GiB RAM
Method 4: Machines

- Linux-Debian8/i4790K, 24GiB RAM
- Linux-Debian8/i4790, 32GiB RAM
- OpenBSD-5.8/i4790, 32GiB RAM

- Turbo boost and hyper-threading disabled
- SSH blocked from non-local machines
- Daemons disabled (cron, smtpd)
Benchmark runner: tries to control as many confounding variables as possible
Method 5: Krun

Benchmark runner: tries to control as many confounding variables as possible e.g.:

- Minimises I/O
- Sets fixed heap and stack ulimits
- Drops privileges to a ‘clean’ user account
- Automatically reboots the system prior to each proc. exec
- Checks `dmesg` for changes after each proc. exec
- Checks system at (roughly) same temperature for proc. execs
- Enforces kernel settings (tickless mode, CPU governors, ...)

[BenchmarkRunner: TriesteControlAsManyConfoundingVariablesAsPossible]

- Minimises I/O
- Sets fixed heap and stack ulimits
- Drops privileges to a ‘clean’ user account
- Automatically reboots the system prior to each proc. exec
- Checks `dmesg` for changes after each proc. exec
- Checks system at (roughly) same temperature for proc. execs
- Enforces kernel settings (tickless mode, CPU governors, ...)

[13/30 HTTP://SOFT-DEV.ORG/]
Preliminary results
Classical Warmup

Richards, Graal, Linux1/i7-4790K, Process execution #3

In-process iteration vs. Time(s)
Classical Warmup

Fasta, V8, Linux2/i7-4790, Process execution #1

Time(s)
In-process iteration
Classical Warmup

(Different machines)
Richards, Hotspot, Linux2/i7-4790, Process execution #2

In-process iteration

Time(s)
In-process iteration

Time(s)

Fannkuch Redux, Hotspot, Linux1/i7-4790K, Process execution #1

Software Development Team
Cycles

Fannkuch Redux, Hotspot, OpenBSD/i7-4790, Process execution #4

Time(s)

In-process iteration
Never-ending Phase Changes

Fasta, LuaJIT, OpenBSD/i7-4790, Process execution #5

In-process iteration

Time(s)
Inconsistent Process-executions

(Note: same machine)
Inconsistent Process-executions

(Note: different machines. Bouncing ball pattern Linux-specific)
Classical warmup occurs for only:
Classical warmup occurs for only:

50% of process executions
Classical warmup occurs for only:

50% of process executions

25% of (VM, benchmark) pairs
Summary

Classical warmup occurs for only:

- 50% of process executions
- 25% of (VM, benchmark) pairs
- 0% of benchmarks for all VMs, machines & proc execs.
**Hypothesis Invalidated**

*Hypothesis:* Small, deterministic programs exhibit classical warmup behaviour.
How can we measure anything any more?
How can we measure anything any more?

For how long has this been going on?
How can we measure anything any more?

For how long has this been going on?

Is this really the fault of the VMs?
Automated Analyses
Outlier Detection

Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #1

Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #2

Measurement

Outliers
Outlier Detection

Outliers outside $5\sigma$ of rolling average
Outlier Detection

In-process iteration

Measurement

Unique outliers (0.05%)

Common outliers (0.40%)

Recurring outliers

Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #1

Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #2

Unique outliers (0.05%)
Change-point Analysis

fasta:V8:default–javascript, run: 5
fannkuch Redux: Hotspot: default-java, run: 1

Change-point Analysis
Change-point Analysis

binarytrees:PyPy/default–python, run: 1
Future Work

The obvious (control more variables; more benchmarks; more 6-S; etc.)

Why we see what we see? E.g. is that spike at $x = 78$ actually {G#, compilation, ...}.
The ‘obvious’ (control more variables; more benchmarks; more VMs; etc.)
The ‘obvious’ (control more variables; more benchmarks; more VMs; etc.)

Can we work out why we see what we see? e.g. is that spike at $x = 78$ actually \{GC, compilation, … \}
Full Results

https://archive.org/download/softdev_warmup_experiment_artefacts/v0.2/

• all_graphs.pdf All plots in one huge PDF.
• warmup_results*.json.bz2 Raw results.

(Note: newer results available)
VM Warmup Blows Hot and Cold
E. Barrett, C. F. Bolz, R. Killick, V. Knight, S. Mount and L. Tratt.

Rigorous Benchmarking in Reasonable Time
T. Kalibera and R. Jones

Specialising Dynamic Techniques for Implementing the Ruby Programming Language
C. Seaton (Chapter 4)

Quantifying performance changes with effect size confidence intervals
T. Kalibera and R. Jones
Thanks for listening