Diagnosing Issues in Java Apps using Thermostat and Byteman

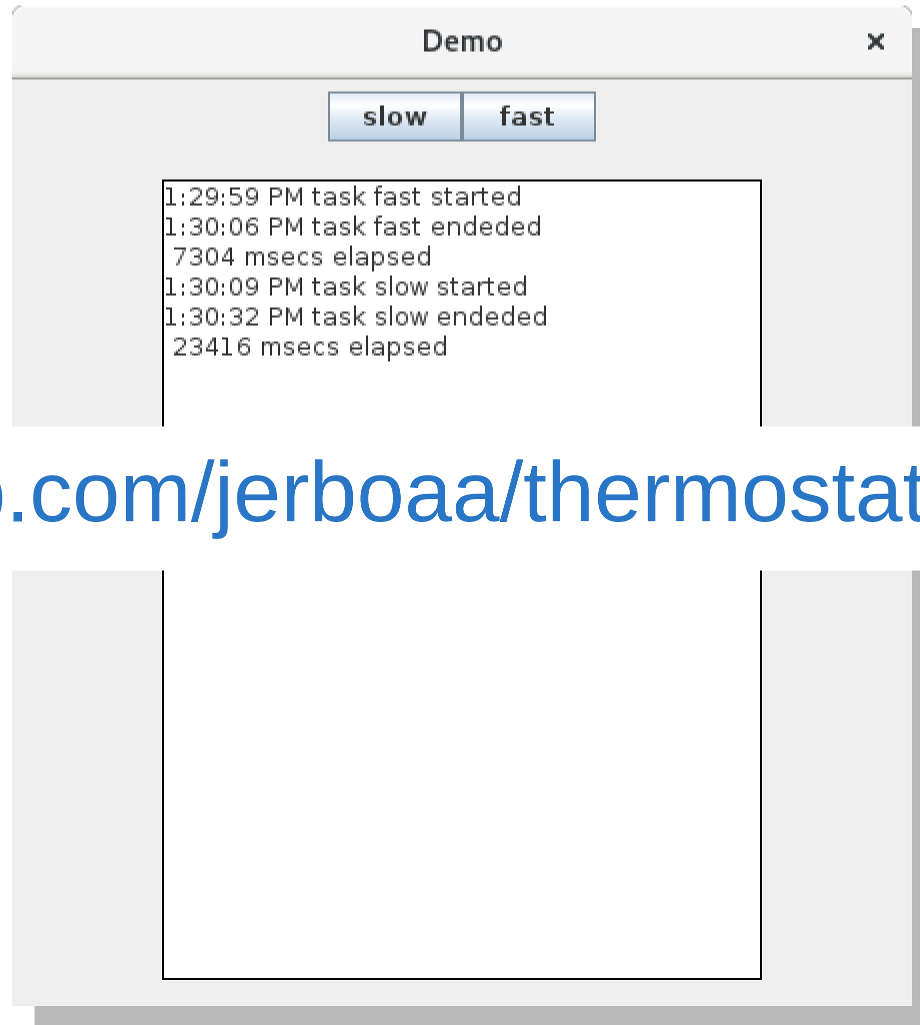
Severin Gehwolf

Senior Software Engineer, Red Hat
FOSDEM, February 4, 2017

Agenda

- Introduction of Demo Application: *“Demo”*
- *“Demo”* in Thermostat
- Byteman
- Source Code and Byteman Rules for *“Demo”*
- Thermostat Byteman Plugin Demo

Demo Application



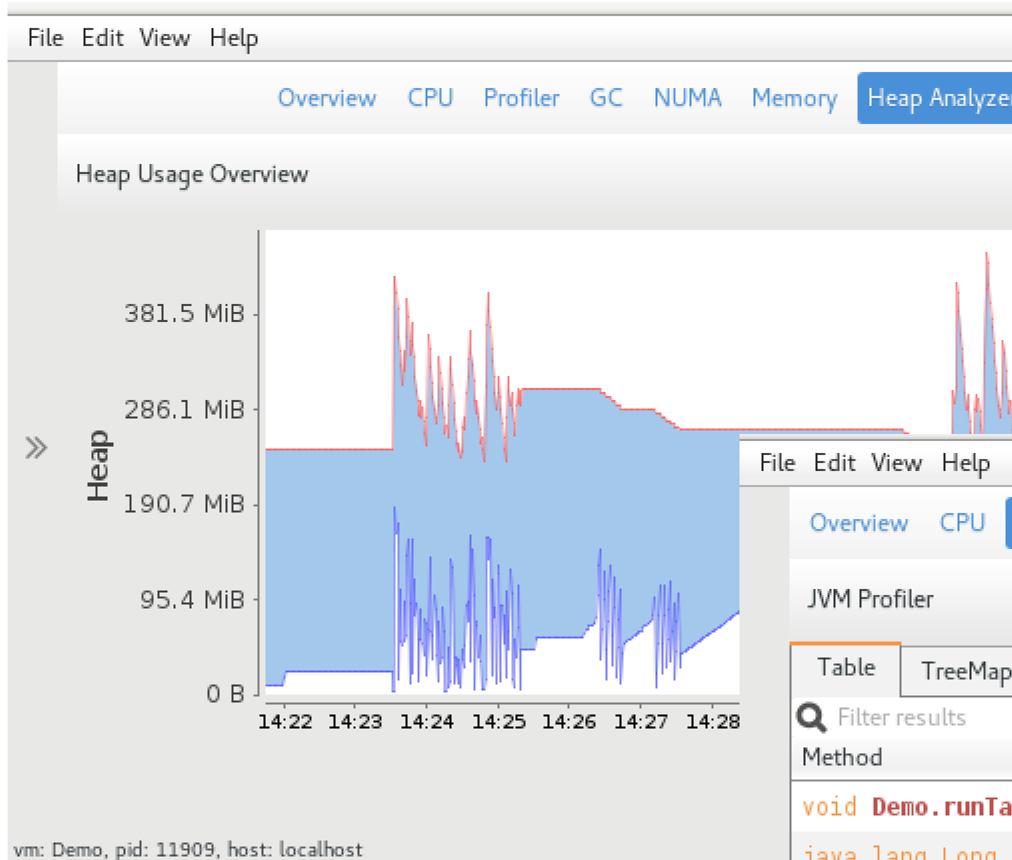
http://github.com/jerboaa/thermostat-byteman-demo

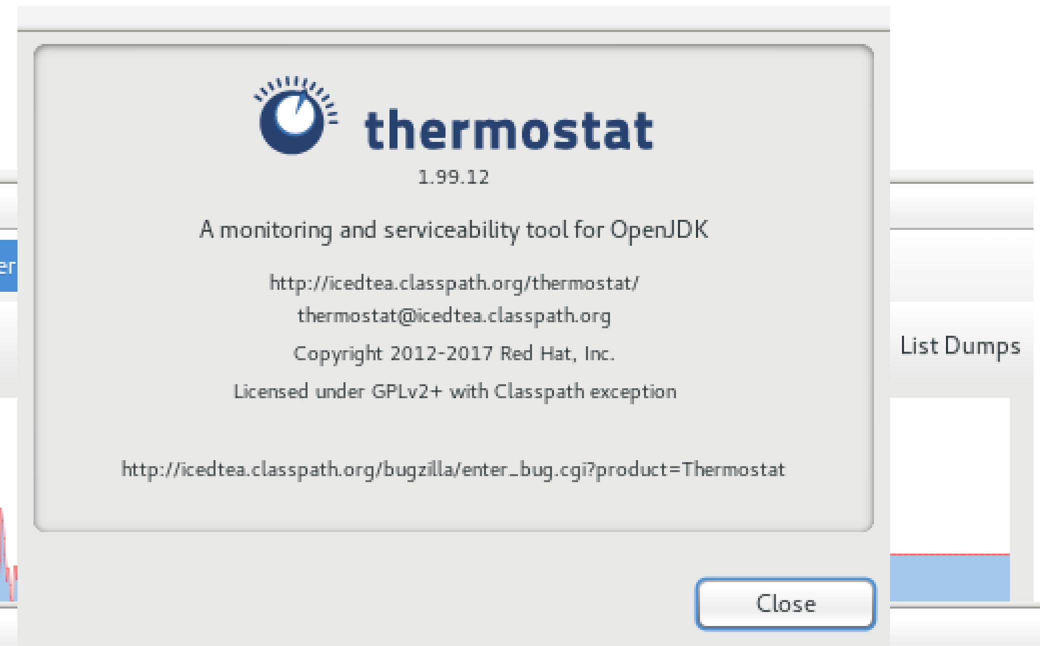
The Problem

- Task execution times vary greatly
- There are “fast” and “slow” tasks
- Profilers in analysis inconclusive
  - Only show aggregate results
  - Byteman can help determining where calls come from
- Example is based on real customer case (simplified)

What if we look at “Demo” with Thermostat?

Tools: Thermostat





File Edit View Help

Overview CPU Profiler NUMA GC Memory Heap Analyzer IO Threads Compiler

JVM Profiler — Start Profiling — List Sessions

Table | TreeMap

Filter results

Method	Percentage	Total Time (ms)
void Demo.runTask(Task)	50.00139146567718	107803
java.lang.Long Task.computeIntensive(int)	49.30287569573284	106297
void Task.ioWait()	0.6957328385899815	1500
void Demo.runFastTask()	0.0	0
void Task.doWork(int)	0.0	0
void Demo$1.actionPerformed(java.awt.event.Actior	0.0	0
Task Demo.getFastTask()	0.0	0

Thermostat: Host CPU vs. JVM CPU time

Thermostat

File Edit View Help

Overview | Processor | Memory | NUMA | Notes

Processor

Processor Model Intel(R) Core(TM) i7-4900MQ CPU @ 2.80GHz

Processor Count 8

Fast *Slow* *Fast*

CPU Usage (%)

100 90 80 70 60 50 40 30 20 10 0

14:19:30 14:20:00 14:20:30 14:21:00 14:

Time

Display the most recent 3 MINUTES

Host

Thermostat

File Edit View Help

Overview | CPU | Profiler | NUMA | GC | Memory | Heap Analyzer | IO

CPU Usage by Virtual Machine

CPU Usage (%)

13 12 11 10 9 8 7 6 5 4 3 2 1 0

14:19 14:20 14:21 14:22

Time

JVM

Thermostat: GC

- Short-lived Objects
- No GC in Old

Thermostat

File Edit View Help

Overview CPU Profiler NUMA GC Memory Heap Analyzer IO

Garbage Collection in the VM

Perform GC

Configured Garbage Collector: Parallel Collector

Collector PSParallelCompact running on old

GC Time (ms)

1.00 0.75 0.50 0.25 0.00

14:19 14:20 14:21 14:22 14:23

Display the most recent 5 MINUTES

Collector PSScavenge running on new

GC Time (ms)

3.0 2.5 2.0 1.5 1.0 0.5 0.0

14:19 14:20 14:21 14:22 14:23

Display the most recent 5 MINUTES 0.935

vm: Demo, pid: 10290, host: localhost

Thermostat: Threads

- Tasks are Threads
- 2 x fast, 1 x slow

Thermostat

File Edit View Help

Overview CPU Profiler NUMA GC Memory Heap Analyzer IO

Thread Control Panel — Monitor Thread — List Recordings

Thread Count | Timeline | Locks | Stack Trace Profiler

Live Threads: 12 Daemon Threads: 9

Living Threads vs. Daemon Threads

threads: 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4

time: 14:20, 14:21, 14:22, 14:23, 14:24

— Living Threads — Daemon Threads

Table | Details | Deadlock Detection

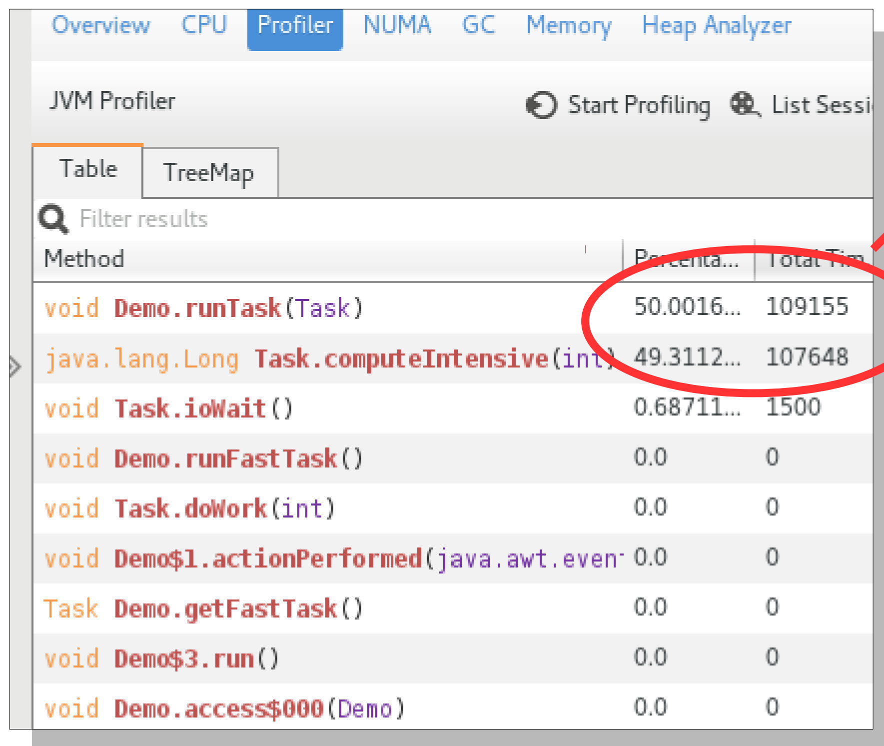
N...	...	First Sa...	Last Sa...	Wait ...	Blocked...	Runni...	Waiti...	Sleep...	Moni...

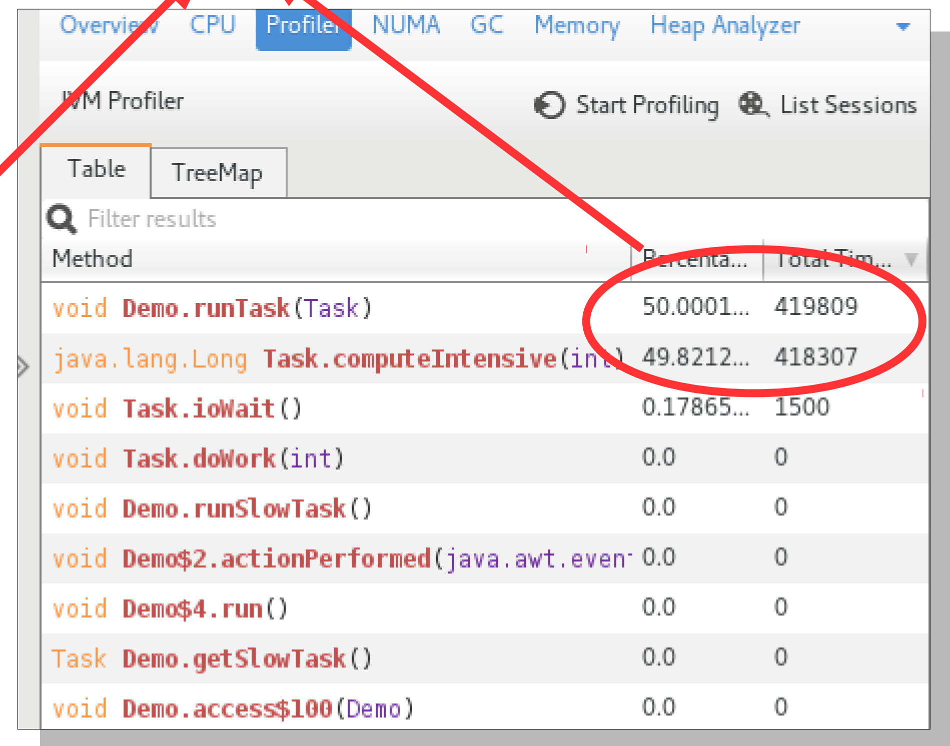
vm: Demo, pid: 10290, host: localhost

Thermostat Profiler

- Find cause of performance problem
- Instrumenting profiler: Method level

Absolute times differ between fast and slow





Thermostat Profiler

- Find cause of performance problem
- Instrumenting profiler: Method level

Inconclusive

Abs between fast and slow

Overview CPU Profiler NUMA GC Memory Heap Analyzer

JVM Profiler — Start Profiling — List Sessi

Table | TreeMap

Filter results

Method	Percenta...	Total Tim
void Demo.runTask(Task)	50.0016...	109155
java.lang.Long Task.computeIntensive(int)	49.3112...	107648
void Task.ioWait()	0.68711...	1500
void Demo.runFastTask()	0.0	0
void Task.doWork(int)	0.0	0
void Demo$1.actionPerformed(java.awt.even	0.0	0
Task Demo.getFastTask()	0.0	0
void Demo$3.run()	0.0	0
void Demo.access$000(Demo)	0.0	0

Overview CPU Profiler Memory Heap Analyzer

JVM Profiler — Start Profiling — List Sessions

Table | TreeMap

Filter results

Method	Percenta...	Total Tim...
void Demo.runTask(Task)	50.0001...	419809
java.lang.Long Task.computeIntensive(int)	49.8212...	418307
void Task.ioWait()	0.17865...	1500
void Task.doWork(int)	0.0	0
void Demo.runSlowTask()	0.0	0
void Demo$2.actionPerformed(java.awt.even	0.0	0
void Demo$4.run()	0.0	0
Task Demo.getSlowTask()	0.0	0
void Demo.access$100(Demo)	0.0	0

Conclusion: Thermostat

- Thermostat useful for ...
  - Understanding the problem better
  - Getting some Evidence
  - Narrowing down the culprit
- But …
  - Evidence might be too coarse grained
  - Need a tool to drill down

Could Byteman help?

Tools: Byteman

- General purpose tool for instrumenting Java code
- JVMTI agent, uses DSL for rules
- Dynamic loading and unloading of rules

```
RULE work started
CLASS Task
METHOD doWork(int)
AT ENTRY
HELPER org.jboss.byteman.thermostat.helper.ThermostatHelper
BIND count = incrementCounter($0);
     id = "work" + $0.getName() + count;
     input = $1
IF TRUE
DO resetTimer($this);
   send("work", new Object[] { "transition", "call",
                               "input", input,
                               "id", id });
ENDRULE
```

Simplify Java tracing, monitoring and testing with Byteman

Tools: Byteman

- Sample Usage:

```
$ java -javaagent:${BH}/lib/byteman.jar=script:thread.btm \
     -Dorg.jboss.byteman.transform.all \
     org.my.AppMain2 foo bar baz
```

- Inject into running JVM:

```
$ bminstall.sh <PID_OF_JVM>
$ bmsubmit.sh -l thread.btm
```

Could Thermostat help?

Tools: Thermostat + Byteman

- Combine Thermostat and Byteman to drill down on the problem
  - Use Byteman for ad-hoc metrics retrieval
  - Use Thermostat to:
    - Drive Byteman
    - Visualize Metrics
  - Implemented as Thermostat plug-in

Thermostat Byteman Demo (almost!)

Source Code and Byteman Rules Deep-Dive

Example Java Application (1)

Class Demo

```
public Task getFastTask() {
    return new Task("fast") {
        public void run() {
              doWork(39);
              ioWait();
              doWork(40);
              ioWait();
              doWork(41);
              ioWait();
        }
    };
}
```

```
public Task getSlowTask() {
    return new Task("slow") {
        public void run() {
              doWork(35);
              ioWait();
              doWork(40);
              ioWait();
              doWork(45);
              ioWait();
        }
    };
}
```

Example Java Application (2)

Class Task

```
class Task extends Thread {
    public void doWork(int i) {
        computeIntensive(i);
    }

    public void ioWait() { … }

    public void computeIntensive(int i) { … }
}
```

Byteman Rules For Java App (1)

```
RULE work started
CLASS Task
METHOD doWork(int)
AT ENTRY
HELPER org.jboss.byteman.thermostat.helper.ThermostatHelper
BIND count = incrementCounter($0);
     id = "work" + $0.getName() + count;
     input = $1
IF TRUE
DO resetTimer($this);
   send("work", new Object[] { "transition", "call",
                               "input", input,
                               "id", id });
ENDRULE
```

Byteman Rules For Java App (2)

```
RULE work ended
CLASS Task
METHOD doWork(int)
AT EXIT
HELPER org.jboss.byteman.thermostat.helper.ThermostatHelper
BIND count = readCounter($0);
     id = "work" + $0.getName() + count;
     elapsed = getElapsedTimeFromTimer($0);
IF TRUE
DO send("work", new Object[] {"transition", "return",
                              "elapsed", elapsed,
                              "id", id});
ENDRULE
```

Thermostat Byteman Demo

Thanks!

Questions?

References

Thermostat: http://icedtea.classpath.org/thermostat/

Byteman: http://byteman.jboss.org

Demo Code:
  http://github.com/jerboaa/thermostat-byteman-demo

Slides: http://bit.ly/2klBpFv