Profilers Are Lying Hobbitses

Nitsan Wakart (@nitsanw)
Lead Performance Engineer, Azul Systems
Thanks!
I work on Zing!

- Awesome JVM
- Only on Linux/x86
- Aimed at server side systems
- Highly focused on responsiveness
  - C4 – Fully concurrent GC
  - ReadyNow! - Persisted profile data
But Also:

- Blog: http://psy-lob-saw.blogspot.com
- Open Source developer/contributor:
  - JCTools
  - Aeron/Agrona
  - Netty/Akka/RxJava/YCSB/HdrHistogram
  - Honest-Profiler/perf-map-agent
- Cape Town Java Meetup Organizer
Why profile?
IT'S SAD WHEN YOU SHOW YOUR FIANCE THE FLYING CAR YOU'VE INVENTED AND YOU HAVEN'T GOT A FIANCE AND YOU'VE GLUED AN OWL TO A RENAULT CLIO.
What are the typical root causes you most often experience? 

- 28.4% Concurrency issues
- 18.5% Slow DB
- 27.6% Memory leak
- 12.0% Slow/unreliable third party entities
- 13.3% Excessive disk IO
- 38.6% Too many database queries
- 17.9% GC pauses
- 23.4% Configuration issues*
- 10.6% Excessive memory churn
- 7.6% HTTP session bloat
- 4.4% Don't know
- 1.7% Other

54.8% Slow database queries
51.5% Inefficient application code

Answers were multiple-choice, so the numbers don't add up to 100%. Deal with it :)

Figure 1.16
Which profiler?
Which tools do you use for application profiling?

- 46.5% VisualVM
- 25.7% JProfiler
- 20.6% Custom in-house tools
- 17.1% Java Mission Control
- 15.0% YourKit
- 13.9% None
- 8.4% Don’t know
- 6.2% Other
- 9.1% NetBeans profiler
- 3.5% JProbe
- 3.3% XRebel

Figure 1.12
LIVE DEMO TIME!!!!!!
Sampling Profilers

- Sample program on interval
- Distribution of samples highlights hotspots
- **Assumption:** *Samples are 'random'*
- **Assumption:** Sample distribution approximates 'Time Spent' distribution
Sampling?
Sampling Profilers

WebServerThread.run()

Controller.doSomething()

Repo.readPerson()

new Person()

View.printHtml()

Controller.next()
Not enough samples

Solution: Switch to tracing profiler
Solution: Shorter sampling interval
Solution: Patience
Sampling interval matching application life cycle

Solution: Shorter interval
Solution: Randomized interval
Sample taking is expensive

Solution: Switch sampling method
Solution: Accept overhead
Solution: Longer interval
Sample is biased/inaccurate

Solution: Switch sampling method
Solution: Widen your scope
Problems with JVisualVM*?

- Reports all threads (running or not)
- Uses **GetStackTrace**:  
  - High overhead
  - **Safepoint** Biased

* And all other JVMTI::GetStackTrace based profilers  
** Will be explained shortly...
GetStackTrace: the official API

• Input: Thread
• Output:
  – Error code (failure IS an option)
  – List of frames (jmethodId, jlocation)

https://docs.oracle.com/javase/8/docs/platform/jvmti/jvmti.html#GetStackTrace
jlocation, where J-Lo be at?
BCI → Line of Code

- BCI – Byte Code Index
- Not every BCI has a line of code
- Find the closest...

Look in hprof for example: <OPENJDK-HOME>/demo/jvmti/hprof
GetStackTrace samples at a Safepoint
Safepoint?

LEO...JUST REMEMBER THE SAFE WORD

PINEAPPLES!!!!
Safepoint (noun.)

A JVM thread state

- Waiting/Idle/Blocked → @Safepoint
- Running Java code → !@Safepoint
- Running JNI code → @Safepoint

http://psy-lob-saw.blogspot.com/2014/03/where-is-my-safepoint.html
At a Safepoint

“...the thread's representation of it's Java machine state is well described, and can be safely manipulated and observed by other threads in the JVM”

Gil Tene, on “Mechanical Sympathy” mailing list:
https://groups.google.com/d/msg/mechanical-sympathy/GGByLdAzIPw/cF1_XW1AbpEJ
Why bring threads to Safepoint?

- Some GC phases
- Deoptimization
- Stack trace dump (and other JVMTI activities)
- Lock un-biasing
- Class redefinition
- And more!

See excellent talks:
https://vimeo.com/120533011 : “When Does the JVM JIT & Deoptimize?” by Doug Hawkins
How does a JVM bring a thread to a 'Safepoint'? 

1) Raise **Safepoint** ‘flag’
2) Wait for thread to **poll** Safepoint 'flag'
3) Thread transitions to **Safepoint** state
Where do we see a Safepoint poll?

- Between every 2 bytecodes (interpreter)
- Backedge of non-'counted' loops (C1/C2)
- Method exit (C1/C2)
- JNI call exit
public void foo(Bar bar) {
    int nogCount = 0;
    for (int i = 0; i < 10; i++) {
        if (bar.getZog(i).isNog()) nogCount++;
    }
    while (nightIsYoung) {
        nogCount += hit(bar);
    }
    if (nogCount > MAX_NOG)
        throw new NogOverflowError();
}
public void foo(Bar bar) {
    int nogCount = 0;
    for (int i = 0; i < 10; i++) {
        if (bar.getZog(i).isNog()) nogCount++;
    }
    while (nightIsYoung) {
        nogCount += hit(bar);
        // Safepoint poll
    }
    if (nogCount > MAX_NOG)
        throw new NogOverflowError();
    // Safepoint poll
}
Safepoint Bias

WebServerThread.run()

Controller.doSomething()

Repo.readPerson()

new Person()

View.printHtml()

Controller.next()

???
It's just a harmless lil' safepoint they said
GetStackTrace Overheads

- `setSafepoint(true)`
- `resumeAll()`

Actual GetStackTrace work
GetStackTrace overhead (OpenJDK)

- Stop ALL Java threads
- Collect single/all thread call traces
- Resume ALL stopped threads

Use -XX:+PrintGCAplicationStoppedTime to log pause times
GetStackTrace overhead (Zing)

- Stop sampled Java thread
- Collect single thread call trace
- Resume stopped thread
LIVE DEMO TIME!!!!!
GetStackTrace demo points

- Use -XX:+PrintGCApplicationStoppedTime
- Safepoint location is 'arbitrary'
- Overhead scales with number of threads
- Widen scope up the call tree?
I will not buy this RECORD, it is SCRATCHED!!!!
AsyncGetCallTrace: unofficial API

- **Input:** signal context and JNI env
  - Context will provide PC/FP/SP
- **Output:**
  - Error code (failure IS an option)
  - List of frames (jmethodId, lineno)
  - lineno == BCI
Why Use AsyncGetCallTrace?

- Built for sampling in **signal handler**
- Does not require a safepoint
- Samples the **interrupted** thread
- Interrupted thread need not be at safepoint

http://jeremymanson.blogspot.co.za/2013/07/lightweight-asynchronous-sampling.html
AsyncGetCallTrace sequence

1. TIMER
2. signal()
3. J1@J
4. OS interrupt
5. Interrupt Handler
6. AsyncGetStackTrace
7. Serialize
8. J2@J
9. signal()
10. J4@JNI
11. OS interrupt
12. Interrupt Handler
13. AsyncGetStackTrace
14. Serialize
15. J5@JNI
Who Uses AsyncGetCallTrace?

- Solaris Studio (but not only AGCT...)
- Java Flight Recorder
- Lightweight-Java-Profiler
- Honest-Profiler
LIVE DEMO TIME!!!!!
AGCT demo points

- **Use:** `-XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints`
- Only Java stack is covered
- Only on **CPU** is sampled
- Lookout for failed samples
Oh?
You want the truth?
YOU CAN'T HANDLE THE TRUTH!!!
Reality is complex...

• There is no Line Of Code
• There's no BCI
• Only instructions
• And more than just Java
Stack Frame → Call Trace Frame

• Stack frame:
  – PC – program counter
  – FP – frame pointer (optional)
  – SP – stack pointer

• Call trace frame:
  – jmethodid
  – BCI
PC → BCI

- PC points to the 'current' instruction
- Not every instruction has a BCI
- Find the closest...
Funny Thing About PCs...

“> I think Andi mentioned this to me last year --
> that instruction profiling was no longer reliable.

It never was.”

http://permalink.gmane.org/gmane.linux.kernel.perf.user/1948
Exchange between Brenden Gregg and Andi Kleen
Skid

- Super Scalar CPU
- Speculative execution
- Signal latency

The blamed instruction is often shortly after where the big cost lies
PC → BCI → Line of Code

- This is as good as it gets
- Mostly it's good enough
- Look for other suspects nearby

60% OF THE TIME

IT WORKS, EVERY TIME
Nearby? Nearby where?
LIVE DEMO TIME!!!!!
Perf-map-agent demo points

- **Use:** `-XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints`
- No LOC info (fixable)
- Only **CPU** is sampled
- Opportunity to differentiate virtual/real frames
Take Aways

- Know your profiler
- There's no perfect profiler
- Try an 'unbiased' profiler, give honest-profiler/perf-map-agent a go!