Sela.

jvmappws

Profiling JVM Applications in Production

college@sela.co.il 03-6176666

Profiling JVM Applications in Production

jvmappws - Version: 1

1 days Course

Description:

Profiling JVM applications it not very easy in the first place. Many profilers, such as JVisualVM and jstack, will simply lie to your face about which call stacks are hottest and where your bottlenecks lie. Profiling in production environments has even more challenges, because you have to carefully manage overhead, account for other processes running on the system, and choose non-invasive tools that don't require an application restart. This is the workshop for you -- we will learn, understand, and experiment first-hand with JVM profiling tools designed for production use.

We will start from a general overview of performance goals and metrics on Linux systems, using the popular USE (Utilization, Saturation, Errors) method. We will build a simple checklist for verifying JVM application performance, and finding the area to focus on in a closer investigation. Then, we will experiment with two approaches for CPU profiling on Linux: the perf multi-tool, combined with perf-map-agent, and the async-profiler project, an innovative tool that brings perf together with traditional JVM profiling techniques. We will visualize stack traces using flame graphs, and understand where the CPU bottlenecks lie, through a series of hands-on labs.

In the second half of this workshop, we will talk about more complicated scenarios: diagnosing errors when opening files, tracing database queries, monitoring system I/O load, understanding the reasons for excessive garbage collection, figuring out why threads are blocked off-CPU, and more. Some of these tasks can be approached using perf, but others require a bleeding-edge technology -- BPF and BCC. BPF is a new Linux kernel technology that enables low-overhead, super-efficient production monitoring and tracing tools. We will use

several of the tools and understand how to build our own tools as necessary, in another series of hands-on labs.

Intended audience:

Java developers, production engineers, and operations engineers

Prerequisites:

Familiarity with the JVM: JIT compilation, GC, threads
Familiarity with Linux OS concepts: processes, threads, virtual memory
Development experience in Java is recommended, but an operations/administration background is also welcome

Objectives:

Analyzing CPU utilization and finding CPU bottlenecks
Investigating system I/O events
Diagnosing GC and allocation issues

Topics:

º Performance metrics and the USE method

º Linux and JVM performance information sources: tracepoints, {k,u}probes, USDT, JMX, JVMTI, Java SA, JFR, GC/JIT logs

º Linux JVM performance checklist: top, sar, free, iostat, pidstat, vmstat, jstack, jstat (based on hsperfdata), jcmd, jattach

º Linux perf, CPU sampling, getting stack reports, problem with symbols

º Generating perf maps with perf-map-agent, inline frames, source info

º Visualizing JVM profiles with flame graphs

º LAB: CPU profiling with perf, perf-map-agent, and flame graphs

º JVMTI profiling with an agent & jattach: GetAllStackTraces, exceptions, class loads, monitor contention etc.

º Full-stack JVM profiling with async-profiler, how it works, pros and cons

º LAB: Profiling with async-profiler (also in a container after enabling perf_event_open)

º Introduction to BPF

º BPF scenarios and the BCC toolkit, Java-specific tools, JVM USDT probes, -XX:+ExtendedDTraceProbes

º Dedicated BCC tools: fileslower, opensnoop, gethostlatency, runqlat, cpudist

º LAB: Snooping failed file opens

º General-purpose BCC tools: trace, argdist, funccount, stackcount, funclatency

º LAB: Tracing MySQL database queries using USDT probes and using uprobes

º Heap allocation profiling (based on TLAB sampling) with async-profiler vs. funccount/stackcount vs. grav

º LAB: Excessive GC and heap allocation profiling

º (If time permits) Profiling in containers: running tools on the host (perf maps, symbols, paths), BCC "just works"

º (If time permits) Container-specific issues: CPU share throttling, Java heap sizing to cgroup

º (If time permits) LAB: Identifying CPU throttling using cgroup/cpu.stat file, runqlat, cpudist

º (If time permits) LAB: Identifying OOM kill due to Java process using more than container memory limit

º (If time permits) Developing custom BCC tools

º (If time permits) LAB: Developing a contention monitor