

JVM Performance Tuning

TRAINING

contents

- ☞ Description
- ☞ Intended Audience
- ☞ Key Skills
- ☞ Prerequisites
- ☞ Instructional Method
- ☞ course contents

mobile: +91.9880951838
mailto: mohit.riverstone@gmail.com
mailto: mohit@stillwaters.ai
website: www.stillwaters.ai
blog: slowbreathing.github.io

JVM Performance Tuning

TRAINING

course contents

- ☛ Hardware Day1
- ☛ Operating System Day2
- ☛ Java Virtual Machine
- ☛ Java Language Tuning Day3

mobile: +91.9880951838
mailto: mohit.riverstone@gmail.com
mailto: mohit@stillwaters.ai
website: www.stillwaters.ai
blog: slowbreathing.github.io

Description:

- Focus of the training is to make JVM and Java performance tuning clear and simple as possible for the participants at the design, architecture and, implementation levels. This is an end-to-end training. The Training illustrates almost every concept with the help of pictures because it is much easier to understand the concept pictorially and model code. There are a lot of illustrations in the course of the training. There are worked out examples to illustrate the concepts for almost every topic. There is a detailed case study that strings together all concepts and technology. There are case studies for debugging JVM Crashes, Memory Leaks, Operating System stalls , and Hardware Bottlenecks. We learn how to associate these events with code.

Intended Audience:

- The target group is programmers who want to know foundations of concurrent programming and existing concurrent programming environments, in order, now or in future, to develop multithreaded applications for multi-core processors and shared memory multiprocessors.
- Java Consultants, developers or anyone with Java experience interested in performance testing.

Key Skills:

- Identify and debug memory leak
- Case studies of various real problems
- Identify and debug Operating System Stalls
- Identify and debug Hardware stalls
- Debug JVM Crash
- Using various hardware, OS,JVM tools and their usage

Prerequisites:

- Good Knowledge of Java.
- Workstation with JDK 1.8.0 installed
- Working knowledge of JVM
- Workstation with JDK 1.7.0 installed

Instructional Method:

- Some topics(e.g. memory analysis) are listed is more than one heading(Hardware, OS, JVM).
- This is an instructor led course provides lecture topics and the practical application of JVM tuning techniques and the underlying technologies.
- It pictorially presents most concepts and there is a detailed case study that strings together the technologies, patterns and design.
- The reason for a problem may be Hardware, OS , JVM or Application. The technique

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
mailto:mohit@stillwaters.ai
website:www.stillwaters.ai
blog:slowbreathing.github.io

and tools for problem identification may be different.

mobile: +91.9880951838
mailto: mohit.riverstone@gmail.com
mailto: mohit@stillwaters.ai
website: www.stillwaters.ai
blog: slowbreathing.github.io

JVM Performance Tuning

■ *Hardware*

■ Cache and Memory

- Measure the effects of cache on a java program
- Identifying bottlenecks with the help of these measurements
- Keeping latency low and throughput high by engaging the cache
- Measuring TLBs performance and its effects
- Associating bottlenecks with java code.
- How Hardware effects Performance of JAVA application?
- Measure the effects of other processor and hardware Counters on a java program
- Hardware counters that can be measured
- Hardware and software Prefetchers
- TLBs effect on java code.
- Crash Course in modern hardware
- How cache effects performance?
- TLBs architecture
- Cache Levels and their architecture

■ Disk

- How to measure tardy disk?
- Identifying with correct reason
- Is your disk IO slow?
- Reasons for tardy performance.
- Associating the tardy performance with java code

■ Locking and Concurrency

- Cache Coherency
- MESI
- False Sharing
- Associating False Sharing with Java Code
- Detecting False Sharing

- Processor Affinity
 - Effects of processor affinity
 - Measuring effects of affinity
- CPU
 - Measuring CPI and IPC
 - CPU Performance Counters
 - Associating and CPI and IPC with performance and java code
- ***Operating System***
 - Locking and Concurrency
 - is_lock_owned
 - try_spin
 - Object layout with JOL
 - spin_pause
 - Understanding Padding
 - False Sharing
 - Designing Classes to avoid false share
 - @Contended and related annotation
 - complete_monitor_locking
 - Virtual Memory
 - Page Replacement
 - Caveats of using TLB
 - Introduction
 - How physical memory acts
 - Swap Space
 - Pages and page frames
 - Multilevel page tables
 - How the operating system sees memory
 - Optimizing Page table access
 - How virtual memory acts
 - Virtual memory and shared memory
 - Demand Paged Virtual Memory and Working Sets

- Influencing TLB performance
- JVM Tuning for Virtual Memory
 - -XX:_UseLargePages
 - /sys/kernel/mm/transparent_hugepages/enabled
 - /proc/sys/vm/nr_hugepages
 - /proc/meminfo Hugepagesize
 - Linux huge Pages
 - Linux Transparent huge pages
 - LargePages
- Setting Processor Affinity at OS
 - what is taskset
 - isolcpus
- Operating-System-Specific Tools
 - gdb
 - conky
 - vmstat
 - mpstat
 - iostat
 - system tap
 - top
- ***Java Virtual Machine***
 - Memory Analysis
 - Core/Heap dumps Analyzer
 - jdb Utility
 - Attaching to a Process
 - Attaching to a Core File on the Same Machine
 - Attaching to a Core File or a Hung Process from a Different Machine
 - JConsole Utility
 - HPROF - Heap Profiler
 - CPU Usage Sampling Profiles (cpu=samples)

- CPU Usage Times Profile (`cpu=times`)
- Heap Dump (`heap=dump`)
- Heap Allocation Profiles (`heap=sites`)
- Java VisualVM
- Serviceability Agent(SA)
 - Cache Dump
 - Stepping Through heap
 - Class Browser
 - Compute reverse pointers
 - Stepping through NON Heap
 - Deadlock detection
 - Value in code cache
 - Code Viewer
- CPU Usage Profilers
 - Solaris Studio Analyzer (Linux and Solaris)
 - stepping through assembly with source
 - `er_print` utility
 - stepping through call-stack (native and java)
 - stepping through byte codes with source
 - Associating hardware events with java code analyzer
 - Collecting processor specific hardware events
 - Collect command
 - Java Mission Control
 - Enabling JFR
 - Selecting JFR Events
 - Java Flight recorder
- Garbage Collection and Memory Architecture
 - Heap Fragmentation
 - GC Pros and Cons
 - Object Size
 - Algorithms

- Overview
- Performance
- GC Tasks
- Reachability
- Managing OutOfMemoryError
- Generational Spaces
- Measuring GC Activity
- History
 - Summary
 - Old Space
 - Young Space
 - JVM 1.4, 5, 6
- Diagnostics and Analysis
 - Native Memory Best Practices
 - Measuring Footprint
 - NIO Buffers
 - Minimizing Footprint
 - Native Memory Tracking
 - FootPrint
 - Integrating Signal and Exception Handling
 - Reducing Signal Usage
 - Console Handlers
 - Signal Chaining
 - Signal Handling on Solaris OS and Linux
 - Alternative Signals
 - Signal Handling in the HotSpot Virtual Machine
 - Reasons for Not Getting a Core File
 - Diagnosing Leaks in Native Code
 - Crash in Compiled Code
 - Tracking Memory Allocation With OS Support
 - Using libumem to Find Leaks

- Tracking Memory Allocation in a JNI Library
- Tracking All Memory Allocation and Free Calls
- Sample Crashes
- Crash in Native Code
- Crash in VMThread
- Determining Where the Crash Occurred
- Crash due to Stack Overflow
- Using dbx to Find Leaks
- Crash in the HotSpot Compiler Thread
- Troubleshooting System Crashes
- Diagnosing Leaks in Java Language Code
 - Obtaining a Heap Histogram on a Running Process
 - -XX:+HeapDumpOnOutOfMemoryError Command-line Option
 - jmap Utility
 - Using the jhat Utility
 - JConsole Utility
 - Monitoring the Number of Objects Pending Finalization
 - Obtaining a Heap Histogram at OutOfMemoryError
 - HPROF Profiler
 - NetBeans Profiler
 - Creating a Heap Dump
- Developing Diagnostic Tools
 - Java Platform Debugger Architecture
 - java.lang.management Package
 - Java Virtual Machine Tools Interface
- Troubleshooting Hanging or Looping Processes
 - Diagnosing a Looping Process
 - Deadlock Detected
 - No Thread Dump
 - Deadlock Not Detected
 - Diagnosing a Hung Process

- Forcing a Crash Dump
- Troubleshooting Memory Leaks
 - Crash Instead of OutOfMemoryError
 - Meaning of OutOfMemoryError
 - Detail Message: <reason> <stack trace> (Native method)
 - Detail Message: Java heap space
 - Detail Message: request <size> bytes for <reason> Out of swap space?
 - Detail Message: PermGen space
 - Detail Message: Requested array size exceeds VM limit
- Finding a Workaround
 - Crash During Garbage Collection
 - Class Data Sharing
 - Crash in HotSpot Compiler Thread or Compiled Code
- Garbage Collection-Advanced Tuning Scenarios
 - Advance Tuning Scenarios-Part2
 - JDK 5,6,7 defaults
 - Default Flags
 - Garbage Collection Data of Interest
 - Tuning GC For Throughput and Latency
 - Latency
 - Old(Parallel)
 - Perm
 - Young (Parallel)
 - Pset Configuration
 - Old(CMS)
 - Tenuring Distribution
 - Initiating Occupancy
 - Common Scenarios
 - Survivor Ratio
 - Tenuring threshold

- **Througput**
 - (Parallel GC)
 - CondCardmark
 - Adaptive Sizing
 - Tlabs
 - Large Pages
 - Numa
 - Pset Configuration
- **CMS**
 - Concurrent Mode Failure
- **Monitoring GC**
 - Par New
 - Parallel GC
 - Safe Pointing
 - Time Stamps
 - Date Stamps
 - System.GC
- **Advance Tuning Scenarios-Part1**
 - Monitoring the GC
 - Conclusions
- **GC Tuning**
 - Tuning Parallel GC
 - Tuning CMS
 - Tuning the young generation
- **GC Tuning Methodology**
 - Deployment Model
 - Choosing Runtime
 - General GuideLines
 - Data Model
- **Heap Sizing**
 - Factor Controlling Heap Sizing

- Advanced JVM Architecture
 - Tuning inlining
 - MaxInlineSize
 - InlineSmallCode
 - MaxInline
 - MaxRecursiveInline
 - FreqInlineSize
 - Monitoring JIT
 - Deoptimizations
 - Backing Off
 - PrintCompilation
 - OSR
 - Log Compilations
 - Optimizations
 - PrintInlining
 - Intrinsic
 - Common intrinsics
 - Understanding and Controlling JVM Options
 - DoEscapeAnalysis
 - AggressiveOpts
 - CallSites
 - Polymorphic
 - BiMorphic
 - MegaMorphic
 - MonoMorphic
 - HotSpot
 - Client
 - Server
 - Tiered
 - Advanced JVM Architecture Part 1
 - NUMA

- Inline caching
- Virtual method calls Details
- Virtual Machine Design
- Dynamic Compilation
- Large Pages
- Biased Locking
- Lock Coarsening
- Standard Compiler Optimizations
- Speculative Optimizations
- Escape Analysis
- Scalar Replacements
- Inlining Details
- VM Philosophy
- **Advanced JVM Architecture-Part 2**
 - JIT
 - Mixed mode
 - Golden Rule
 - Profiling
 - Optimizations
- ***Java Language Tuning***
 - **Class-loading**
 - Common Class loading Issues
 - Changing the rules of default class visibility
 - Class Loading Basics
 - Introduction
 - Diagnosing and resolving class loading problems
 - Custom Class Loaders
 - Class visibility