Java Concurrency and JVM Performance

contents

TRAINING

- Description
- Intended Audience
- Key Skills
- Prerequisites
- course contents

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

Java Concurrency and JVM Performance

course contents

TRAINING

- Common Issues with thread Day1
- Java Memory Model(JMM)
- Applied Threading techniques Day2
- Building Blocks for Highly Concurrent Design
- Highly Concurrent Data Structures-Part1
- Designing For Concurrency Day3
- Atomic Variables and Nonblocking Synchronization
- Crash course in Mordern hardware
- Designing for multi-core/processor environment Day4
- Highly Concurrent Data Structures-Part2
- Java Affinity
- Operating System Day5
- Java Virtual Machine
- Java Language Tuning

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

Description:

- Mix of Java Concurrency and JVM Tuming Topics with equal focus.Understand concurrency control issues in general. Know the instruments available in Java. Avoid common errors and pitfalls. Understand concurrency control idioms. Understanding JVM Tuning for GC and Dynamic Compiler and the rest.

Intended Audience:

- Senior Software Engineers proficient with java.

Key Skills:

- Good Knowledge of Concurrency and its usage in cluster software like Hadoop and Spark.
- Impact of Classloading on Memory, JVM and applications.
- Analyzing software design for correct Concurrency design.
- Garbage Collection Tuning with real case studies.
- Performance monitor and tuning the JVM.
- Writing Performance Benchmarks

Prerequisites:

- 5-6 years of experience with Java.
- Decent Knowledge of threads.
- Working knowledge of Java Collection.

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

Java Concurrency and JVM Performance

- ***Common Issues with thread***
  - Uncaught Exception Handler
  - problem with stop
  - Dealing with InterruptedStatus
- ***Java Memory Model(JMM)***
  - Real Meaning and effect of synchronization
  - Volatile
  - Sequential Consistency would disallow common optimizations
  - The changes in JMM
  - Final
  - Shortcomings of the original JMM
    - Finals not really final
    - Prevents effective compiler optimizations
    - Processor executes operations out of order
    - Compiler is free to reorder certain instructions
    - Cache reorders writes
    - Old JMM surprising and confusing
  - New JMM and goals of JSR-133
    - Simple,intuitive and, feasible
    - Out-of-thin-air safety
    - High performance JVM implementations across architectures
    - Minimal impact on existing code
    - Initialization safety
    - Preserve existing safety guarantees and type-safety
- ***Applied Threading techniques***
  - Safe Construction techniques
  - Thread Local Storage
  - Thread safety levels
  - UnSafe Construction techniques

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- ***Building Blocks for Highly Concurrent Design***
  - CAS
    - Wait-free Queue implementation
    - Optimistic Design
    - Wait-free Stack implementation
    - Hardware based locking
    - ABA problem
      - Markable reference
      - weakCompareAndSet
      - Stamped reference
  - Reentrant Lock
    - ReentrantReadWriteLock
    - ReentrantLock
  - Lock Striping
    - Lock Striping on LinkNodes
    - Lock Striping on table
  - Indentifying scalability bottlenecks in java.util.Collection
    - segregating them based on Thread safety levels
  - Lock Implementation
    - Multiple user conditions and wait queues
    - Lock Polling techniques
    - Based on CAS
    - Design issues with synchronization
- ***Highly Concurrent Data Structures-Part1***
  - Weakly Consistent Iterators vs Fail Fast Iterators
  - ConcurrentHashMap
    - Structure
    - remove/put/resize lock
    - Almost immutability
    - Using volatile to detect interference
    - Read does not block in common code path

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- ***Designing For Concurrency***
  - Atomicity
  - Confinement
  - Immutability
  - Visibility
  - Almost Immutability
  - Restructuring and refactoring
- ***Atomic Variables and Nonblocking Synchronization***
  - Hardware support for concurrency
    - Using "Unsafe" to access memory directly
    - CAS support in the JVM
    - Compare-and-Set
    - Performance advantage of padding
    - Nonblocking counter
    - Simulation of CAS
    - Managing conflicts with CAS
    - Compare-and-Swap (CAS)
    - Shared cache lines
    - Optimistic locking
  - Atomic variable classes
    - Optimistic locking classes
    - How do atomics work?
    - Atomic array classes
    - Performance comparisons: Locks vs atomics
    - Cost of atomic spin loops
    - Very fast when not too much contention
    - Types of atomic classes
  - Disadvantages of locking
    - Priority inversion
    - Elimination of uncontended intrinsic locks
    - Volatile vs locking performance

- Nonblocking algorithms
  - Scalability problems with lock-based algorithms
  - Atomic field updaters
  - Doing speculative work
  - AtomicStampedReference
  - Nonblocking stack
  - Definition of nonblocking and lock-free
  - Highly scalable hash table
  - The ABA problem
  - Using sun.misc.Unsafe
    - Dangers
    - Reasons why we need it
- ***Crash course in Mordern hardware***
  - Amdahl's Law
  - Cache
    - cache controller
    - write
    - Direct mapped
    - read
    - Address mapping in cache
  - Memory Architectures
    - NUMA
    - UMA
- ***Designing for multi-core/processor environment***
  - Concurrent Stack
  - Harsh Realities of parallelism
  - Parallel Programming
  - Concurrent Objects
    - Sequential Consistency
    - Linearizability
    - Concurrency and Correctness

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- Progress Conditions
- Quiescent Consistency

- Concurrency Patterns
  - Lazy Synchronization
  - Lock free Synchronization
  - Optimistic Synchronization
  - Fine grained Synchronization
- Priority Queues
  - Heap Based Unbounded Priority Queue
  - Skiplist based Unbounded priority Queue
  - Array Based bounded Priority Queue
  - Tree based Bounded Priority Queue
- Lists
  - Coarse Grained Synchronization
  - Lazy Synchronization
  - Optimistic Synchronization
  - Non Blocking Synchronization
  - Fine Grained Synchronization
- Skiplists
- Spinlocks
  - Lock suitable for NUMA systems
- Concurrent Queues
  - Unbounded lock-free Queue
  - Bounded Partial Queue
  - Unbounded Total Queue
- Concurrent Hashing
  - Open Address Hashing
  - Closed Address Hashing
  - Lock Free Hashing

- ***Highly Concurrent Data Structures-Part2***
  - CopyOnWriteArray(List/Set)

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- Queue interfaces
  - Queue
  - BlockingQueue
  - Deque
  - BlockingDeque
- Queue Implementations
  - ArrayDeque and ArrayBlockingDeque
    - WorkStealing using Deques
  - LinkedBlockingQueue
  - LinkedBlockingDeque
  - ConcurrentLinkedQueue
    - GC unlinking
    - Michael and Scott algorithm
    - Tails and heads are allowed to lag
    - Support for interior removals
    - Relaxed writes
  - ConcurrentLinkedDeque
    - Same as ConcurrentLinkedQueue except bidirectional pointers
  - LinkedTransferQueue
    - Internal removes handled differently
    - Heuristics based spinning/blocking on number of processors
    - Behavior differs based on method calls
    - Usual ConcurrentLinkedQueue optimizations
    - Normal and Dual Queue
- Skiplist
  - Lock free Skiplist
  - Sequential Skiplist
  - Lock based Concurrent Skiplist
  - ConcurrentSkipListMap(and Set)
    - Indexes are allowed to race
    - Iteration

- Problems with AtomicMarkableReference
- Probabilistic Data Structure
- Marking and nulling
- Different Way to mark

- ***Java Affinity***
  - Affinity Thread Factory
  - isolcpus
  - using perf and likwid to measure L1, L2 and, L3 cache performance
  - PosixJNA Affinity
  - Write Buffer
  - Lock Inventory
  - How much does thread Affinity Matters
  - using likwid to measure prefetchers
  - Non Forging Affinity Lock
  - Affinity Strategies
  - Affinity Support
  - Cache Architecture
  - using mpstat to measure and verify
  - Read Buffers
  - CPU Layout
- ***Operating System***
  - Locking and Concurrency
    - is_lock_owned
    - try_spin
    - Object layout with JOL
    - spin_pause
    - Undestanding 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
  - JVMTuning for VirtualMemory
    - -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

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com

- 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
      - Intrinsics
        - 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

mobile:+91.9880951838
mailto:mohit.riverstone@gmail.com
website:www.movaatechnologies.com