Understanding and Designing Ultra Low Latency Systems

contents

TRAINING

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

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

Understanding and Designing Ultra Low Latency Systems

course contents

TRAINING

- CPU Day1
- Disks Day2
- Memory
- Network
- Observability Tools Day3
- Benchmarking
- Low Latency and High Performance Libraries and Classes Day4
- Java Virtual Machine

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

Description:

- Traditional models of concurrent programming have been around for some time. They work well and have matured quite a bit over the last couple of years. However, every once in a while there low latency and high throughput requirements that cannot be met by traditional models of concurrency and application design. How about handling 400-500 million operations/second per core of a system. When designing such systems one has to throw away the traditional models of application design and think different. In this seminar we discuss some of the approaches that can make this possible. This approach is hardware friendly and a re-look at data from a hardware perspective. It requires logical understanding of how modern hardware works. It also requires the knowledge of tools that can help track down a particular stall and possibly the reason behind it. This may provide pointers for a redesign if required. In the balance then, this training is about an architectures that are Hardware friendly. It also about very specialized data structures that fully exploits the underlying architecture of the processor, cache, memory, disk, filesystem, and network. The design of this training is like a series of understand-hardware/OS-concept, apply-it-to-design, measure-it-with-tool.

Intended Audience:

- Software Architects
- JVM Tuners
- Technology Officers
- Senior Software Engineers

Key Skills:

- Understand the cache coherency protocol
- Architectural understanding of disruptor.
- Processor Micro-architecture refresher
- Understand and measure the effect of thread affinity
- Understand Off heap techniques
- Understand and measure the effect of various levels of caches
- Understand how ultra low latency designs are done.
- Understand and measure the effect of prefetches

Prerequisites:

- A very good knowledge of java
- Very experienced in software design and architecture.
- A working knowledge of C/C++ will be helpful
- Knowledge of existing data structures in Java

Instructional Method:

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

- Slides with pictures to represent concepts
- Instructor Led
- Hand-on Session to gauge the effect of hardware

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

Understanding and Designing Ultra Low Latency Systems

- ***CPU***
  - Intel® Xeon™,Sandybridge™,Ivybridge™,Haswell™ Processor
  - Branch mispredictions, Wasted Work, Misprediction Penalties and UOP Flow
  - Uncore Memory Subsystem
  - Performance Analysis
  - processor Performance Events: Overview
  - Performance Analysis and the Intel® Core™ i Processor and Intel® Xeon™
  - Basic Intel® Core™ i Processor and Intel® Xeon™ Processor Architecture and
  - Core Out of Order Pipeline
  - Core Performance Monitoring Unit (PMU)
  - Core Memory Subsystem
  - Uncore Performance Monitoring Unit (PMU)
  - Core Performance Monitoring Unit (PMU)
  - CPU Internals
    - CPU Run Queues
    - Saturation
    - Software
    - Priority Inversion
    - Terminology
    - Word Size
    - Concepts
    - Instruction Pipeline
    - CPU Memory Caches
    - Microcode and Exceptions
    - Models

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

- Clock Rate
- Branch Mispredictions
- Front End Events
- Hardware
- User-Time/Kernel-Time
- Compiler Optimization
- CPU Architecture
- Utilization
- Instruction Width
- Multiprocess, Multithreading
- Preemption
- CPI, IPC
- FE Code Generation Metrics
- Architecture

- 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
- Factoring in CPU specifics into Design
  - Lock effects
  - Ordering Effects
  - Branch Prediction effects
  - Cache Line effcts
  - Cache effects
  - Thread Affinity
  - Multi-Core effects
  - Prefetcher effects

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

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

- ConcurrentSkipListMap(and Set)
  - Indexes are allowed to race
  - Iteration
  - Problems with AtomicMarkableReference
  - Probabilistic Data Structure
  - Marking and nulling
  - Different Way to mark

- ***Disks***
  - Models
  - Terminology
  - Caching Disk
  - Simple Disk
  - Controller
  - Disk Internals
    - IOPS Are Not Equal
    - Caching
    - I/O Size
    - I/O Wait
    - Concepts
    - Storage Type
    - Disk Types
    - Utilization
    - Time Scales
    - Synchronous versus Asynchronous
    - Read/Write Ratio
    - Measuring Time
    - Non-Data-Transfer Disk Commands
    - Saturation
    - Operating System Disk I/O Stack
    - Random versus Sequential I/O
  - Factoring in Disk specifics into Design
    - Scaling

- Micro-Benchmarking
- Event Tracing
- Latency Analysis
- Cache Tuning
- Resource Controls
- Static Performance Tuning

- ***Memory***
  - Virtual Memory
  - Concepts
  - Terminology
  - Memory Internals
    - File System Cache Usage
    - Swapping
    - Utilization and Saturation
    - Overcommit
    - Allocators
    - Process Address Space
    - Demand Paging
    - Paging
    - 9 Word Size
  - JVM Memory
    - Ordering fields of DataValueClasses
    - Write with Direct Reference
    - Off-Heap Data Structures
    - Off-Heap Queues
    - Write with Direct Instance
    - Off-Heap Maps
    - Read with Direct Reference
- ***Network***
  - Network Interface
  - Terminology
  - Models

- Factoring in Network specifics into Design
  - Replication How it works
  - Zero copy/Send file
  - TCP/IP Throttling
  - Multiple Processes on the same server with Replication
  - How to setup UDP Replication
  - TCP / UDP Background
  - TCP / UDP Replication
  - Identifier for Replication
- Network Internals
  - Latency
  - Software
  - Controller
  - Encapsulation
  - Protocols
  - Buffering
  - Packet Size
  - Protocol Stack
  - Connection Backlog
  - Networks and Routing
  - Hardware
  - Interface Negotiation
  - Utilization
- ***Observability Tools***
  - DTrace
  - Perf
  - Profiling
  - Observability Sources
  - Solaris Analyzer
  - /sys
  - Ftrace
  - kstat

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

- Tracing
- SystemTap
- /proc
- JMH
- Tool Types
- Delay Accounting
- Microstate Accounting
- Counters
- Monitoring

- ***Benchmarking***
  - Passive Benchmarking
  - Sanity Check
  - Ramping Load
  - Activities
  - Replay
  - Active Benchmarking
  - Workload Characterization
  - Micro-Benchmarking
  - Background
  - Statistical Analysis
  - Benchmarking Types
  - Custom Benchmarks
  - Methodology
  - Effective Benchmarking
  - CPU Profiling
  - Industry Standards
  - Simulation
  - Benchmarking Sins
- ***Low Latency and High Performance Libraries and Classes***
  - OpenHFT Architecture
    - Compiler
      - Generating Off heap classes for on heap interfaces

- 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
- Chronicle
  - Consumer insensitive
  - How does it collect garbage
  - Messaging between processes via shared memory
  - Synchronous text logging
  - High throughput trading systems
  - Messaging across systems
  - Low latency, high frequency trading
  - Supports recording micro-second timestamps across the systems
  - Synchronous binary data logging
  - Cache friendly
  - Functionality is simple and low level by design
  - Very fast embedded persistence for Java.
  - Replay for production data in test

- Introduction to Chronicle
  - Chronicle:Modes of use
    - Lock-less
    - Supports thread affinity
    - Shared memory
    - Text or binary
    - GC free
    - Replicated over TCP
  - Huge Collection
    - Advanced Off Heap IPC in Java
    - Low latency, high throughput software
    - OpenHFT Chronicle, low latency logging, event store and IPC. (record / log everything)
    - Micro-second latency.
    - OpenHFT Collections, cross process embedded persisted data stores. (only need the latest)
    - Millions of operations per second.
    - HugeHashMap
    - SharedHashMap
    - How is off heap memory used?
      - Around 8x faster than System V IPC.
      - Memory mapped files
      - Durable on application restart
      - One copy in memory.
      - Can be used without serialization / deserialization.
      - Thread safe operations across processes
  - Lang
    - serializable and deserialization of data
    - writing and reading enumerable types with object pooling
    - writing and read primitive types in binary and text without any garbage.
    - random access to memory in native space (off heap)

- provide the low level functionality used by Java Chronicle
- writing and reading String without creating an object (if it has been pooled).
- Small messages serialization and deserialization in under a micro-second.

- The Disruptor Architecture
  - Data structure and work flow with no contention.
  - Very fast message passing
  - Overview of the Disruptor
  - Allows you to go truly parallel
  - Create your own
- Hardware aware data structures in java
  - Magical ring buffers
  - Single writer principle
  - DoubleAdder
  - DoubleAccumulator
  - LongAdder
  - Measuring incremental tangible benefits of hardware aware structures
  - ConcurrentAutoTable
  - LongAccumulator

- ***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
    - 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

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