Scaling Trading Systems with Coherence
Why multi-core processors and low latency networks are crucial

Brian Oliver
Data Grid Solutions Architect
Tangosol Oracle
boliver@tangosol.com
Agenda

• Introduction to Coherence
• Coherence as Trading System infrastructure
• Multi-core Performance
• Network Scale-out Observations
• Conclusions
Introduction to Coherence
Introduction to Coherence

- Data Management & Clustering Solution (In-Memory)
- Native libraries: Java & .NET (C++ soon)
  - UDP-Based Traffic
- #1: Provides Reliable Clustering Technology
  - Cluster applications without a container / server
- #2: Provides Automatic Partitioning of Application Data
  - Including partitioning of backups
- #3: Provides Extremely Low Latency Events
- #4: Provides Grid Agents to Process Data (Affinity)
  - Not a client / server processing model
Partitioned Topology
Partitioned Topology

• Partitioned Topology is one of many
  ‣ Others include Near, Replication, Overflow, *Extend...

• Consistent Data Access Latency
  ‣ One network operation
  ‣ Any size cluster

• Capacity Scales to Available
  ‣ RAM
  ‣ Network
Partitioned Topology

[Diagram showing a partitioned cache topology with JVMs and put() operations.]
Partitioned Topology

- **Consistent Data Update Latency**
  - Two network operations (worst case)
  - Any size cluster

- **Capacity Scales to Available**
  - RAM
  - Network
Partitioned Topology Recovery

Diagram showing the process of partitioned topology recovery in a distributed system. The diagram illustrates the interaction between JVMs, applications, and partitioned caches, with backup and primary logical entities. The process involves getting data from the primary cache, promoting a backup cache to primary, and handling failsafe measures when a primary fails.
Partitioned Topology Recovery

• Dynamic Recovery
  ‣ All cluster members have equal responsibility for Data (and recover)
  ‣ No Management Console = Unattended Health Maintenance
  ‣ No in-flight operations lost / No stop-the-world recovery
  ‣ Realtime events for Cluster membership

• More Processes = Faster Recovery
  ‣ Repartitioning occurs in parallel

• There is an opportunity for data loss
  ‣ When Cluster too Small
  ‣ Catastrophic Failure (many processes die at the same time)
  ‣ Not cluster loss. Data often is recoverable!

• Like RAID for Application Data without a Server!
Partitioned Topology Affinity

Diagram showing the relationship between Spring App, Coherence Cache, Primary, and Backup across multiple JVMs (1, 2, 3, 4). The diagram illustrates how entries are processed and how data is replicated and stored across these entities.
Partitioned Topology Affinity

• Affinity dramatically improves performance & scalability
  ‣ Data Affinity = related data co-located
  ‣ Processing Affinity = processing occurs where data is located

• Significantly reduces network operations
  ‣ eg: 14 to 4 reduction in data updates

• Reducing network increases both performance & scalability

• Types of Data Processing (in Parallel)
  ‣ Indexing
  ‣ Updating
  ‣ Aggregating
Coherence as Infrastructure
Coherence Applied to Trading

• Coherence as Infrastructure
  ‣ Reliable In-Memory Data Management
  ‣ Highly Available (even during recovery and scale-out)
  ‣ Known access and update latencies
  ‣ Dynamically Scalable & Fault Tolerant

• Natural Data Partitioning
  ‣ Instruments, Securities, Trades, Prices, Orders, Executions, Portfolios
  ‣ No longer a developer or operational concern

• Natural Data Affinity
  ‣ Instruments and Prices,
  ‣ Orders and Executions
Coherence Applied to Trading

• Natural Processing Affinity (Extreme Parallelism)
  ‣ Trades, Prices, Orders, Execution updates may occur in parallel
  ‣ Indexing of Portfolios
  ‣ Aggregation of Positions, Risk...

• Low Latency Events
  ‣ No messaging infrastructure required
  ‣ Cluster Membership and Data Updates
  ‣ Thousands of Clients (Desktops & Servers)
  ‣ Continuous Views of Data (as it changes)

• WAN and MAN support
Multi-core Comparison
Baseline Multi-core Test

• Xeon
  ‣ 2 x 3.6Ghz Single Core CPUs / Server

• Woodcrest
  ‣ 2 x 2.6Ghz Dual Core CPUs / Server

• Common
  ‣ 4 GB RAM
  ‣ 3 x Servers
  ‣ 1Gb Network
  ‣ Linux 2.6 SMP Kernel
  ‣ 3 Processes / Server

• Scale-out Test
  ‣ 100 x Servers, 10 x Processes / Server
Multi-core Load Comparison

Entries / Second

Throughput

Xeon
Woodcrest

2KB
10KB
10MB

Entries / Second

Throughput
Multi-core Load Comparison

CPU Utilisation

2KB 10KB 10MB

Xeon

Woodcrest
Network & Scale-out Observations
Network Scale-out

Aggregations / Server (1 Gb)

Entries / Second

Aggregation Latency (secs)

Servers
InfiniBand v’s 10GbE

Packet Size (bytes)

0MB/sec  237.5MB/sec  475.0MB/sec  712.5MB/sec  950.0MB/sec

1968  3968  7968  15968  31968  63968

10 GbE  IB
Conclusions

• All features are important for a solution!

• Reliable Cluster Management is important
  ‣ No time for manually manage a cluster - especially if it’s large!
  ‣ Recovery & Scale-out is automatic - great for large clusters

• Automatic Partitioning is important
  ‣ Removes developer and operational maintenance requirements (cf:DB)
  ‣ Removes hot-member bottlenecks (unless you have hot data points)
  ‣ Provides natural mechanism for scaling out

• Data and Processing Affinity is important
  ‣ Reduces network bandwidth requirements
  ‣ Increases scalability and performance
Conclusions

• Multi-core CPU
  ▸ Processing Affinity pushes CPU’s harder
  ▸ Pushing a 1Gb+ network requires a lot of CPU
  ▸ Some applications don’t use a lot of CPU = increase processes to utilize CPU and increase resilience + availability
  ▸ Higher bandwidth = more CPU load for Coherence (Traffic Management)
Conclusions

• Network
  ‣ Affinity significantly reduces Network bandwidth requirements
  ‣ ie: Less copying and context switching in an application
  ‣ But data volumes are increasing... More applications using a single cluster... Grids are becoming larger...
  ‣ Network remains the major impediment to scalability and performance
  ‣ As bandwidth increases, CPU utilization increases! (traffic management)

• InfiniBand
  ‣ Ideal for large objects
  ‣ Ideal when packet bundling (Coherence 3.3) is applicable
Thank you