JBOSS DATA GRID PERFORMANCE TUNING

Divya Mehra - JBoss Data Grid Product Manager

Vijay Chintalapati - Middleware Solutions Architect

John Osborne - Middleware Solutions Architect

AGENDA

- JBoss Data Grid Introduction and Use Cases
- Performance Tuning Components
    - Configuration and Sizing
    - Tuning the JVM for a distributed system
    - Platform and Network Considerations
    - Coding for a low memory footprint
    - Persistent Stores
    - Tuning for Queries
- Benchmarking
- Roadmap

JBOSS DATA GRID INTRODUCTION

RED HAT JBOSS DATA GRID

A distributed, in-memory NoSQL datastore

HIGH PERFORMANCE AND SCALABILITY

- In-memory access to large data-sets
- High availability, easy scale out

POLYGOT

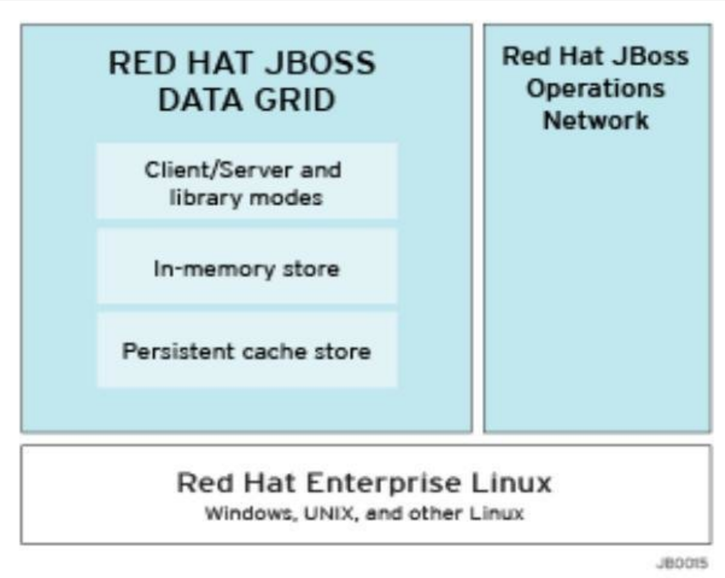
- Java, C++, .NET Hot Rod clients
- REST and memcached protocols available

CERTIFIED INTEGRATION WITH OTHER JBOSS PRODUCTS

- JBoss EAP, JBoss Fuse, JBoss Data Virtualization, JBoss Web Server

FULLY OPEN SOURCE

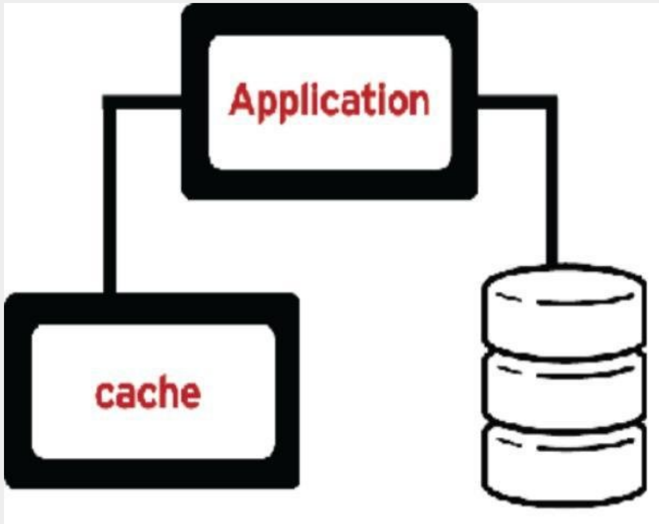
- Based on popular Infinispan project



USE CASE #1

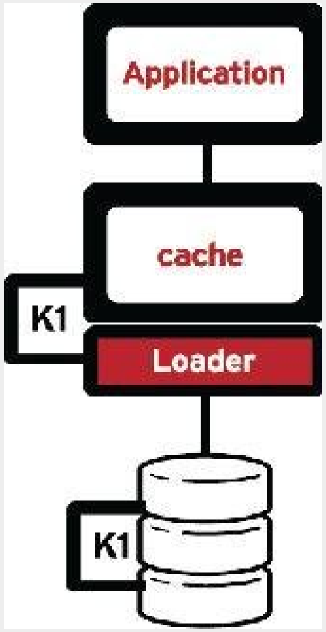
Side cache – as a secondary, high performance store

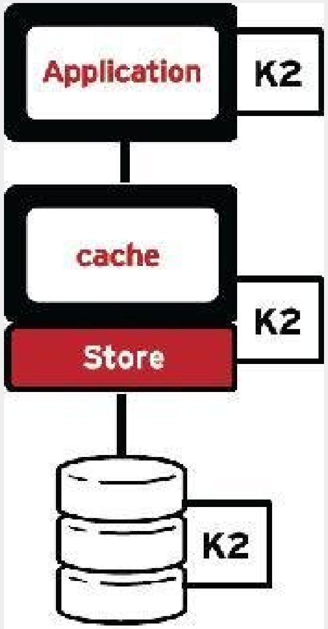
- Database is the primary store
- Distributed cache stores copy
- Application uses the distributed cache as the data source
  - Improves response time by avoiding roundtrip to database



USE CASE #2

Inline cache – primary high-performance, scalable store





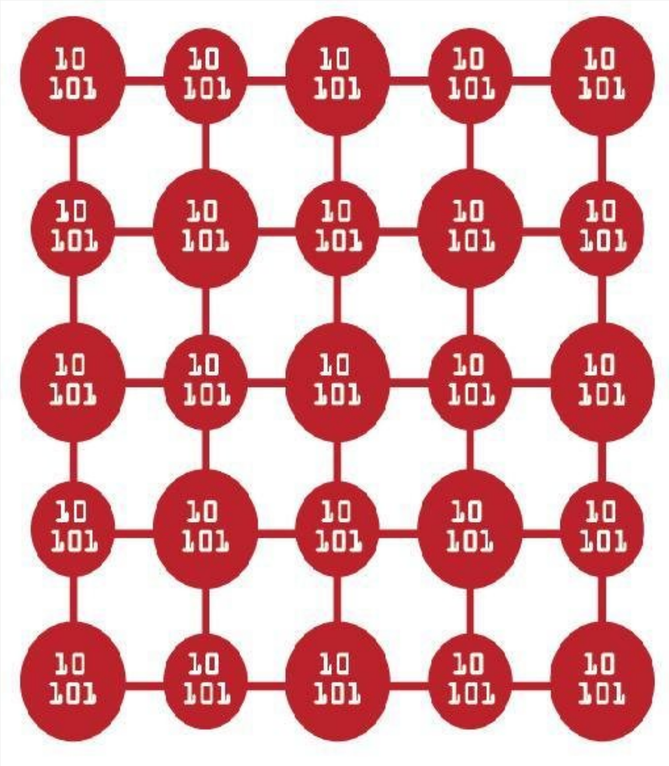
- App requests data (K1)
- If (k1,v1) not in-memory already, Cache retrieves from persistent store

- App writes data (K2)
- Cache writes to persistent store (K2)

USE CASE #3 (emerging)

In-memory compute grid

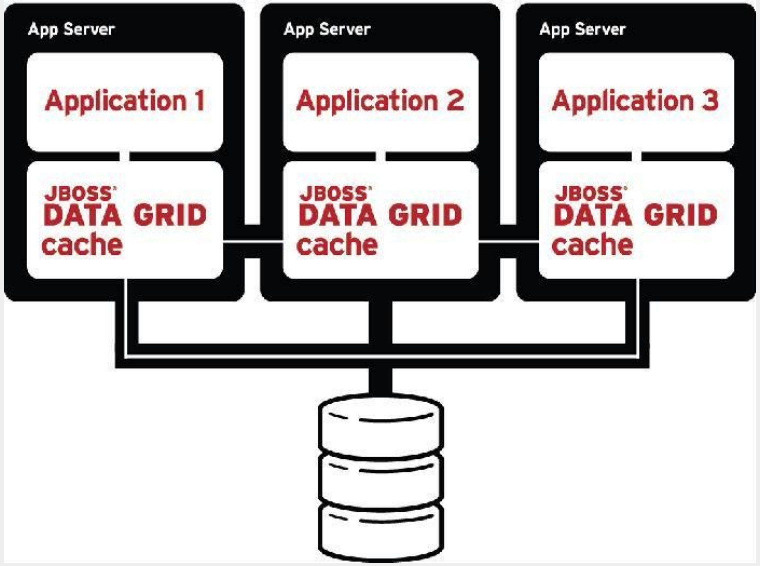
- Process TBs of data rapidly using in-memory distributed computing frameworks:
  - Distributed execution
  - Map/Reduce
- Leverage parallel computing
  - Multiple nodes of the cluster
  - Multiple cores on a machine



DEPLOYMENT MODES

Library mode – Embedded cache

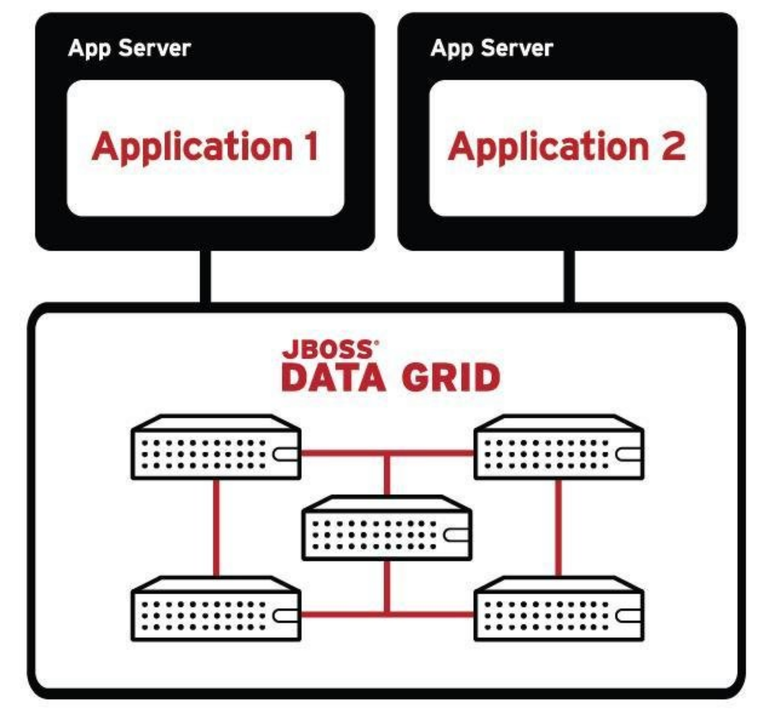
- Clustered JDG caches share heap with applications
    - Data grid scales with the application tier
- Application accesses a cache entry, regardless of whether it is present on locally or on a remote node



DEPLOYMENT MODES

Client-server mode – Remote cache

- Applications communicate with JDG server via protocols
    - Hot Rod
    - REST
    - Memcached
- Data grid scales independent of application tier



CLIENT AND SERVER

Multiple access protocols

Protocol	Format	Client type	Smart?	Load balance and failover
REST	text	any	no	external
Memcached	text	any	no	pre-defined
Hot Rod	binary	Java, C++, C#	yes	auto/dynamic

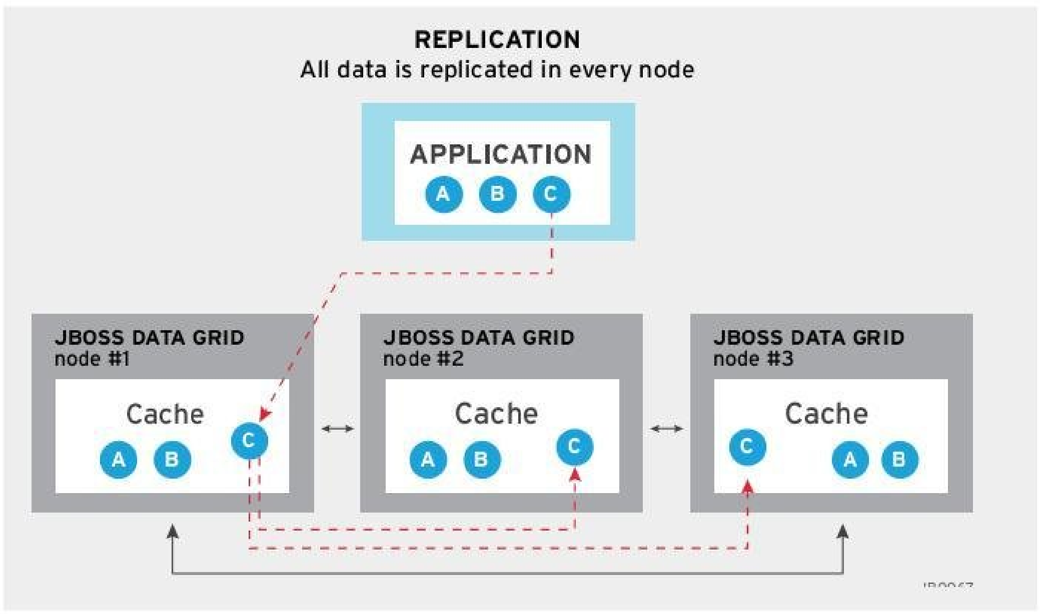
Hot Rod: Native TCP client/server protocol with rich functionality

- Hashing and topology aware
- Failover during topology changes
- Smart request routing in partitioned or distributed server clusters

ARCHITECTURE

Replicated cache

- Replicate the (key/value) entry to each node of cluster
- Local reads
- Writes become slower with increasing number of nodes
- Data limited to a single JVM heap size



IDEAL FOR

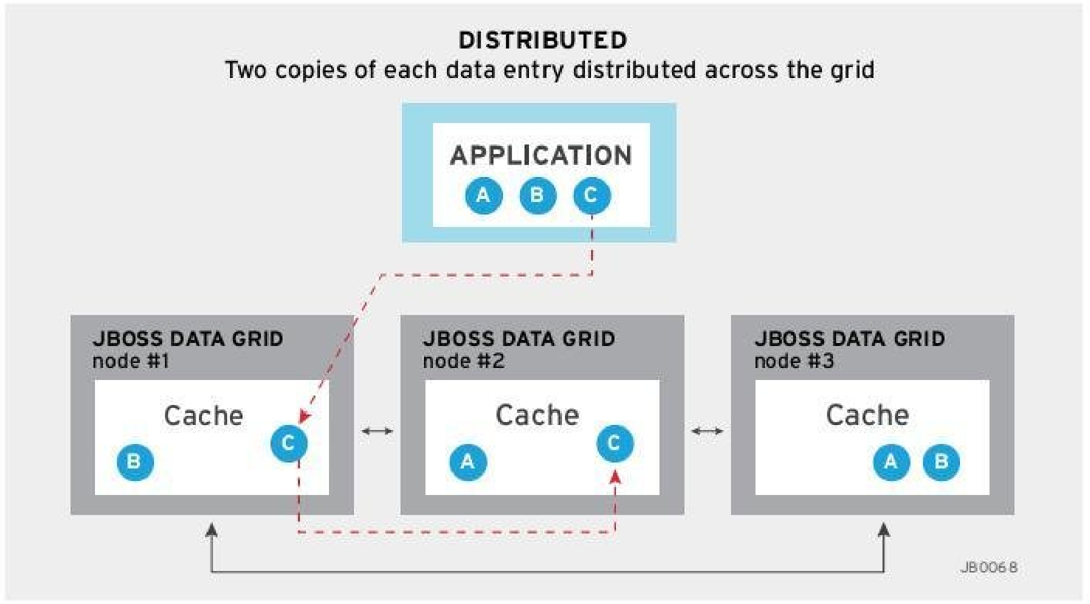
Small, fixed datasets

Highest read performance (local reads)

ARCHITECTURE

Distributed cache

- High performance + high scalability
  - Typically maintain 2 or 3 copies of each entry on separate nodes
- Server hinting allows nodes on separate physical machines



PERFORMANCE TUNING

SELECTING A CONFIGURATION AND SIZING IT

Library Mode vs Client Server Mode

Use Library mode for specific use-cases such as :

- Map/Reduce, Distributed Execution, XA transactions, Advanced API etc

Use Client-Server mode when there is a need for :

- Separation and maintenance of client and JDG processes
- Choice of multiple protocols (REST, Memcached, Hotrod)
- Data access from non-Java applications
- The ability to transparently and horizontally scale for 'dist' caches
- Rolling upgrades without impacting client applications

Initial Sizing Considerations

- How much data do you think you will have in memory?
  - x = key size + value size + 200bytes metadata (library mode)
  - x = Serialized key size + Serialized value size + 200b (server mode)
  - How many entries (y)?
  - Early heap analysis may be required
- Percentage of live data in the heap? (p = 0.5)
- How many copies (n) of the data will I need?

Rule of thumb: Never fill more than half the heap with live data.

Tip: Heap Dumps are your friend

Distributed Cache Sizing Example

- JVM heap size, S = 32GB
- x = 1KB, y=64,000,000 ==> 64GB
- How many nodes to store a single copy of data?

$$m1 = \frac{x \times y}{p \times S} + 1 = \frac{1KB \times 64M}{0.5 \times 32GB} + 1 = 5$$

- Need to tolerate 2 node failures (numowners=3)

$$total = m1 \times n = 5 \times 3 = 15$$

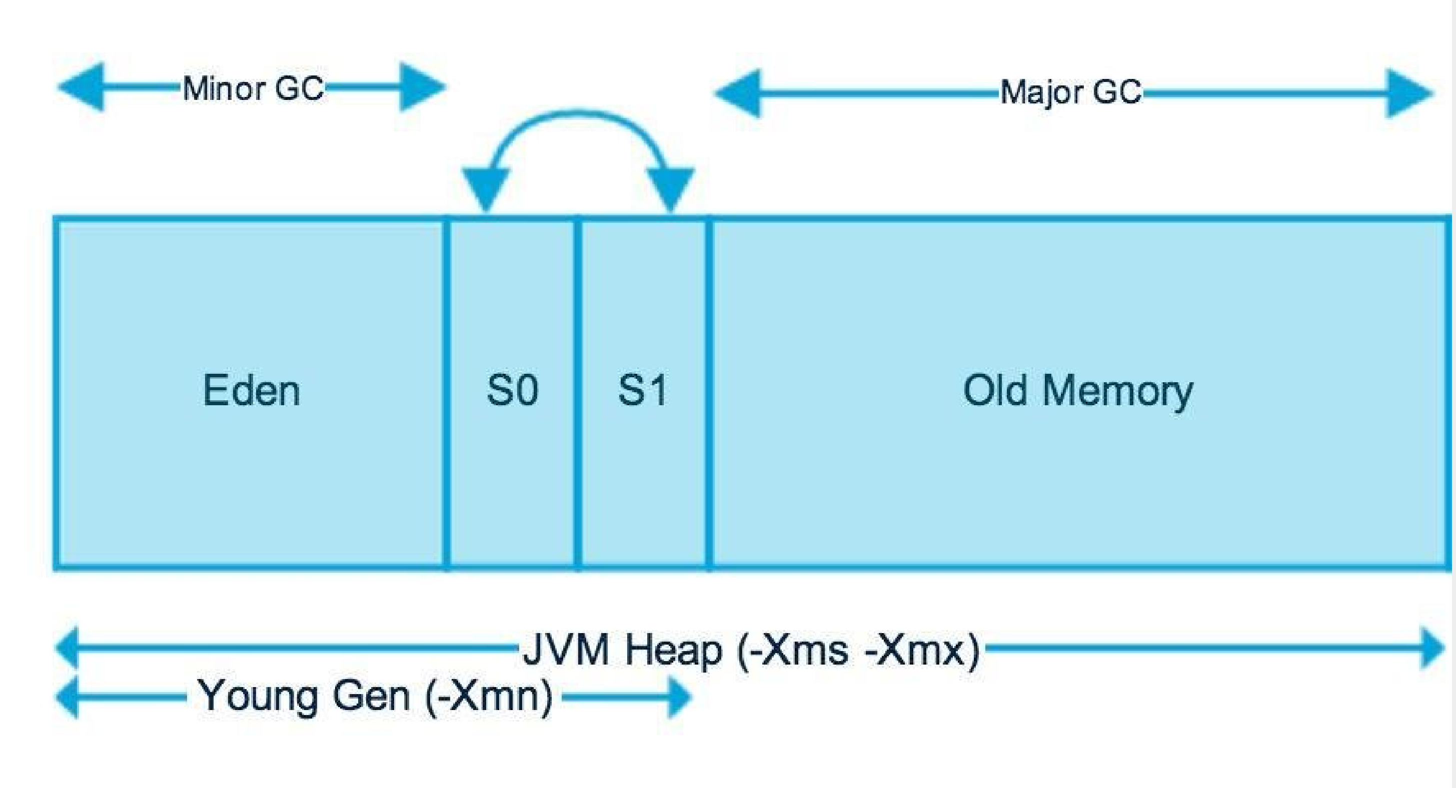
Tip: Server hinting allows multiple heaps per server

Tip: Allocate enough cores to keep up with larger heaps

JVM TUNING FOR DISTRIBUTED SYSTEMS

Java 7 improvements

- JIT Compilation
  - Increase the Code Cache (-XX:ReservedCodeCacheSize=256m)
  - Enable Tiered Compilation (-server -XX:+TieredCompilation)
- Use backported Java 8 CHM -Dinfinispan.unsafe.allow_jdk8_chm=true


Minor GC
Major GC
Eden
S0
S1
Old Memory
JVM Heap (-Xms -Xmx)
Young Gen (-Xmn)

Garbage Collection Considerations

- STW pauses for large heaps can last several minutes
  - Network buffers fill up, potential data loss
  - JGroups can remove the node from the cluster
    - Potential "split-brain" problem
- In most cases you want to pick one of the concurrent collectors
  - Concurrent Mark Sweep (CMS)
  - Garbage First (G1)

Initial Tuning CMS

- Always turn off adaptive sizing (-Xms=-Xmx)
- Typical JDG data set lives longer than traditional JEE
- Manually tune for smaller young generation
  - -XX:NewSize=-XX:MaxNewSize - start 1/8 heap (2GB max)
  - CMS balancing act
    - Smaller new size => decrease throughput
    - Larger new size => Risk of STW
- Increase the Eden size -XX:SurvivorRatio=16 (or 32)
- Turn on PermGen collection (Java 7 only)
  - -XX:+CMSPermGenSweepingEnabled -XX:+CMSClassUnloadingEnabled

CMS Woes?

- Concurrent Mode Failure (gc logs)
  - Increase the heap size
  - Increase the old generation
  - Start CMS earlier
    - -XX:CMSInitiatingOccupancyFraction=60 -XX:+UseCMSInitiatingOccupancyOnly
- Insufficient heap size (> 50% live data)
- Sawtooth pattern (VisualVM)
  - Increase the NewSize

Initial G1 Tuning

- Always turn off adaptive sizing (-Xms=Xmx)
- Tune the pause time (–XX:MaxGCPauseMillis) to meet the 90th percentile for your SLA
  - Starting points: 500ms for 32GB, 1000ms for 64GB
  - Larger values increase the throughput

G1 Woes

- STW Pauses “Full GCs”, “to-space overflow/exhausted”
- Solutions
  - Increase –XX:MaxGCPauseMillis
  - Increase the heap size
  - Modify -XX:InitiatingHeapOccupancyPercent (default 45)
    - But not lower than the % of live data!
  - Increase -XX:ConcGCThreads

DECREASING YOUR MEMORY FOOTPRINT

Java Strings

- *Avg % of live heap = 25%*
- *Avg % of live heap duplicated Strings = 13.5%*
- *Average String length = 45 characters*

- How much space does the following take up on 64-bit RHEL system?
  - String str1 = “”;
  - String str2 = “Hi”
  - String str3 = “Hello”

Java Strings (cont.)

- Answers:
  - String str1 = “”; ==> 40 bytes
  - String str2 = “Hi”; ==> 48 bytes
  - String str3 = “Hello” ==> 56 bytes
- Can save > 50% space by using byte[] arrays if you are using English or other European Language based Characters in your Strings (library mode)
- str3.getBytes("UTF-8"); ==> 24 bytes

Java Strings

```
private byte[] name;
private final Charset UTF8_CHARSET = Charset.forName("UTF-8");

…

public String getName(){
    return new String(name, UTF8_CHARSET);
}
public void setName(String newName){
    name = newName.getBytes(UTF8_CHARSET);
}
```

Java Strings (cont.)

- Duplicate Strings?
  - Doesn't the JVM make sure my string constants aren't duplicated?
  - String intern?
- Are you using G1 and Java 8 (update 20+)?
  - XX:+UseG1GC -XX:+UseStringDeduplication
  - Works for JBoss Data Grid Dynamically Generated Strings
    - Preloaded String Heavy Objects
    - State transferred Strings

Other Tips

- Use smaller instance variables
- Replace wrappers with instance vars //Double is 3x double
- Be aware of object overhead
- Avoid allocating the same object multiple times
    - Modifying immutable (like a String) objects in a loop
        - If you must with Strings, use your own StringBuilder
- Use lazy instantiation for infrequently used values
- Avoid finalize()

PLATFORM AND NETWORK TUNING (YES JGROUPS)

Networking with JBoss Data Grid

- JDG leverages JGroups technology
  - Created by Bela Ban (Red Hat) during his Post-doc at Cornell
  - Subsystem in EAP and JBoss Data Grid for clustering
  - JDG 6.4 adds new default values that covers almost all use cases
- Configure the OS for the JGroups Buffer Sizes
  - sysctl -w net.core.rmem_max=26214400
  - sysctl -w net.core.wmem_max=1048576
- Enable Jumbo Frames

Huge Pages

- Enable on the JVM level -XX:+UseLargePages
- Enable at the OS level
  - RHEL Ex: 2 64GB JVMs running on a server

$$hugepages \geq \frac{(64+64)*2^{30}}{4*2^{20}} = \mathbf{32768}$$

- Verify “java -Xmx<JDG max heap size>g -XX:+UseLargePages -version”
- Disable Transparent Huge Pages

Threads

- JBoss Data Grid encourages horizontal scalability
- Typical implementations run many parallel threads
- If the threads are creating their own variables then consider giving the threads their own buffers
  - -XX:+UseTLAB
  - -XX:TLABSize
  - Eden must be > # JDG threads * TLABSize

PERSISTENT STORES

Cache Stores

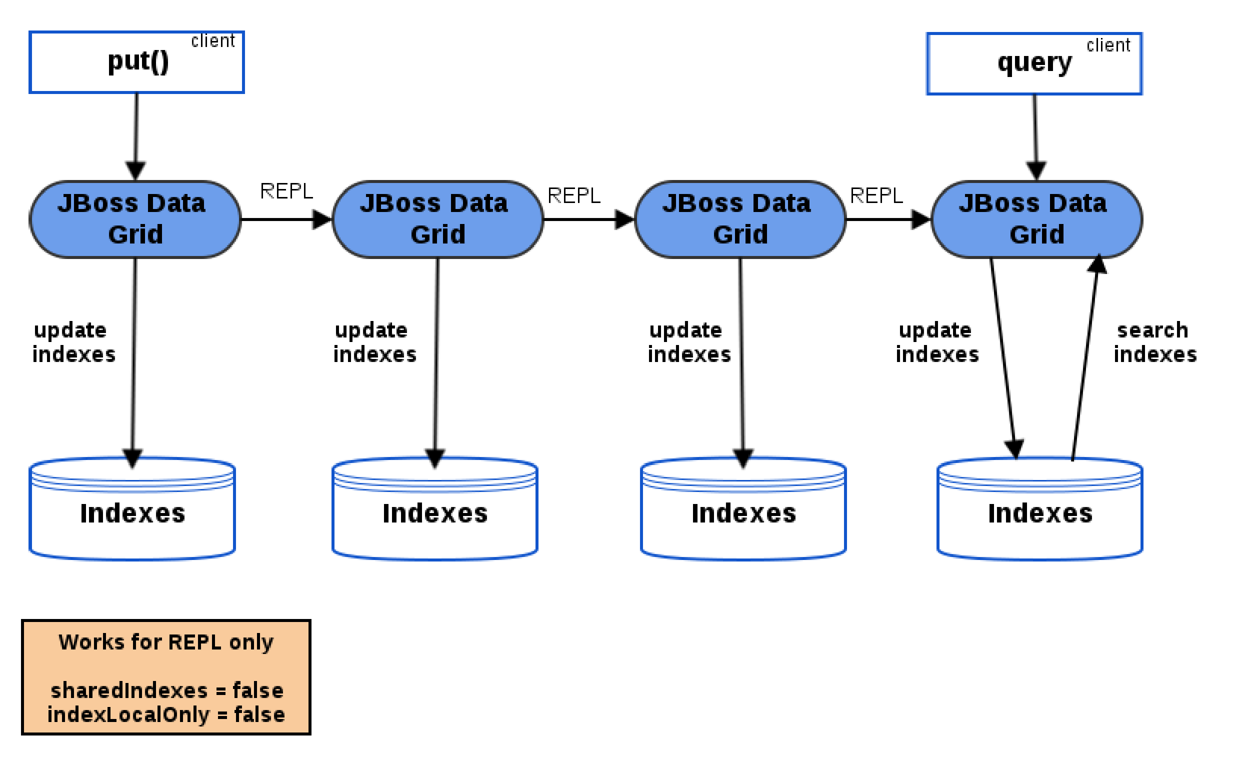
- Shared Cache Stores
  - Every node in the cluster writes to the same store
- Local Cache Stores
  - Each node has its own cache store
  - A key update will result in a write to the cache store of every owner
- Recommendations
  - Shared stores, good for small clusters, can be a bottleneck for large ones
  - LevelDB is the highest performing local cache store

Performance of the Persistent Stores

- Write-Behind (Async) speeds up write to cache
- Prevent cache misses by warming up (preloading) the cache on startup
- For a JDBC cache store:
  - Creating indexes on 'id' column can prevent a table scan.
  - Setting createOnStart on the table definition automatically takes care of defining the id column as PRIMARY KEY
  - Configure batch-size, fetch-size, etc
- JPA cache store in library mode automatically takes advantage of the primary keys

TUNING FOR QUERIES

Local Replicated Index



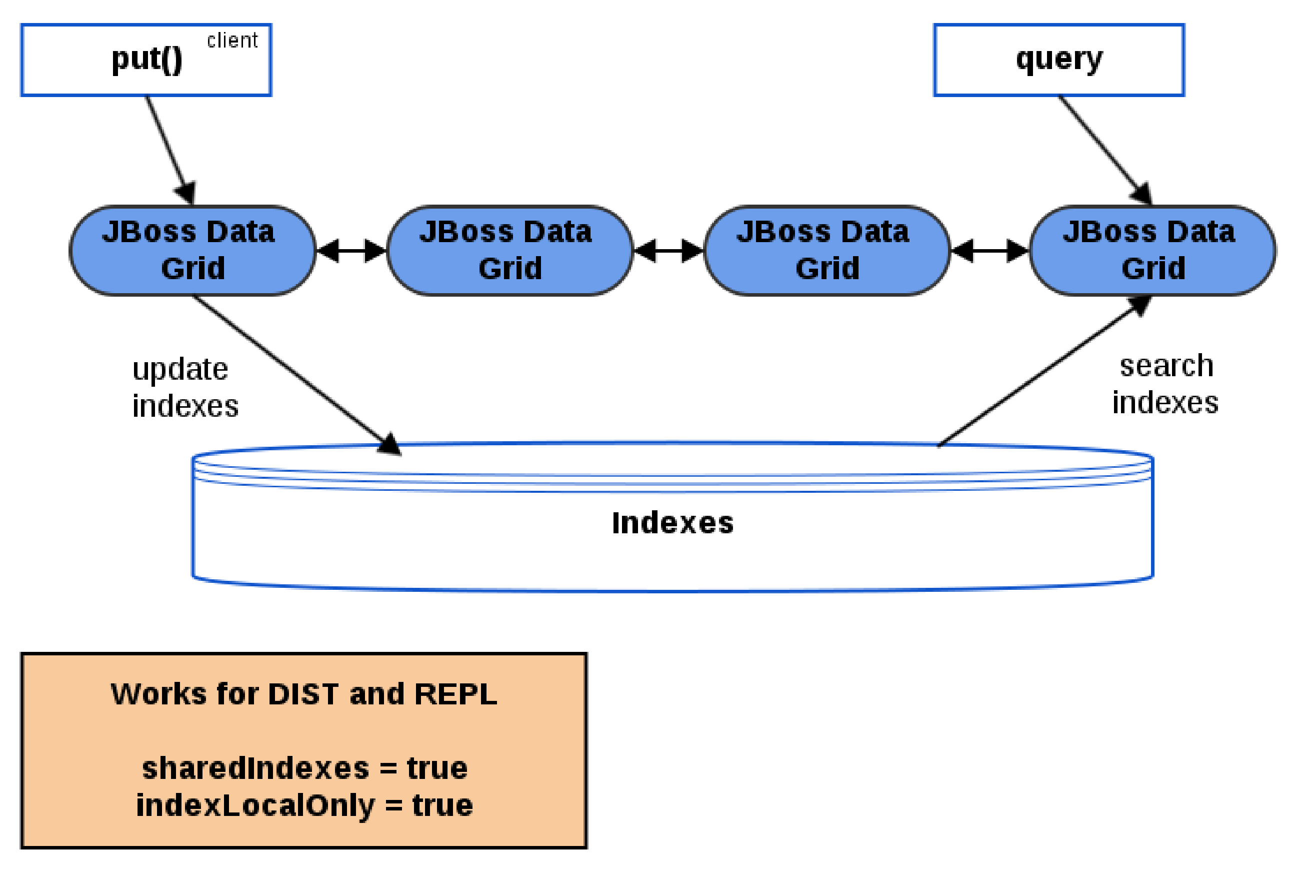
- Use Filesystem Directory Provider
- Increase the chunk size

```
<property name="hibernate.search.infinispan.chunk_size">4096</property>
```

- Use Near-Real-Time Indexing

```
<property name="default.indexmanager" value="near-real-time" />
```

Shared Index



- Lucene Infinispan Directory uses three caches to store the index:
  - Data cache, Metadata cache, Locking cache

General Query Performance Tuning

- Pagination

```
CacheQuery cacheQuery =
Search.getSearchManager(cache).getQuery(luceneQuery, Customer.class);
cacheQuery.firstResult(15); //start from the 15th element
cacheQuery.maxResults(10); //return 10 elements
```

- Avoid Storing Fields in the Index unless using projections

```
@ Indexed
public class Person implements Serializable {
@ Field(store = Store.NO) // Store.NO is the default option
```

- Filter the Search Results by Entity Type

```
CacheQuery cacheQuery =
Search.getSearchManager(cache).getQuery(luceneQuery, Customer.class);
```

- Asynchronous Indexing

```
<property name="default.worker.execution">async</property>
```

BENCHMARKING

Radar Gun

- Radargun is an open-source IMDG benchmarking tool
- Easily benchmark and compare JBoss Data Grid, EHCache, Coherence, etc.
- M/S model; Master runs stages in parallel on all slave nodes

Stage includes

- load-data
- check-cache-data
- cluster-validation
- basic-operations-test
- bulk-operations-test
- jvm-monitor-start / stop

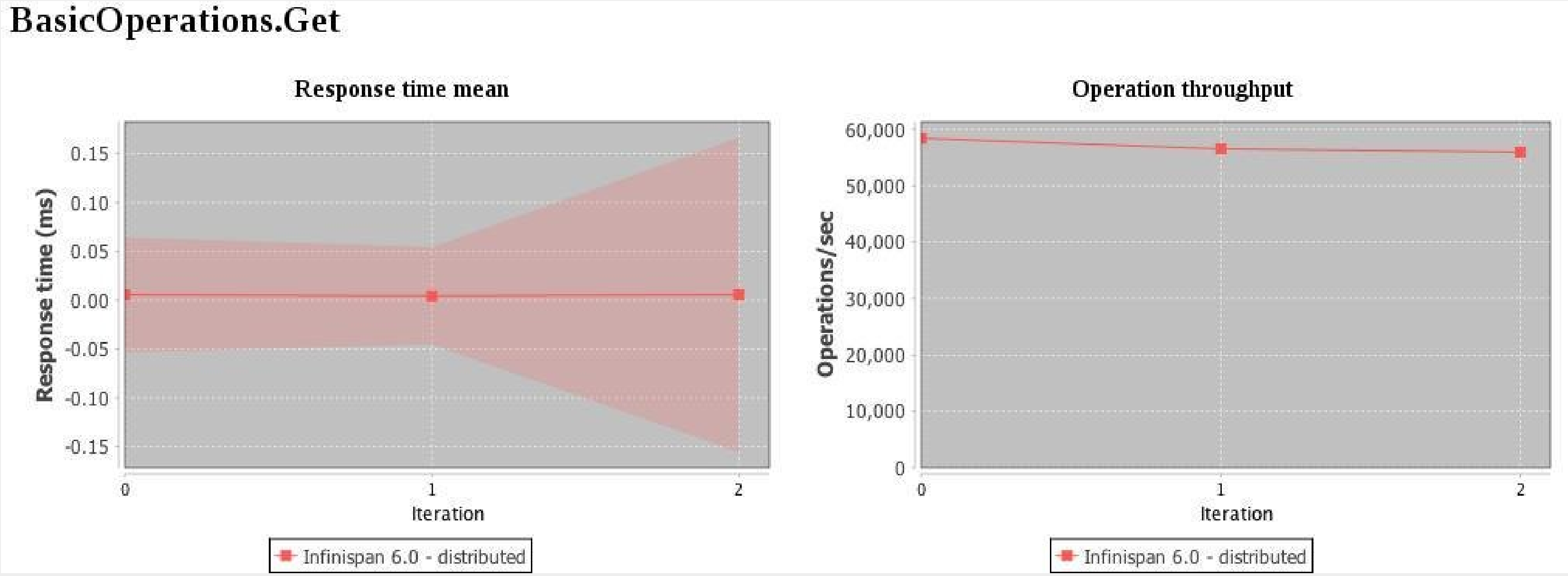
Easily Define

- total # of such operations
- Percentages for each CRUD op
- entry sizes, key generator, value generator
- num of entries
- num of threads per node
- duration of the entire test etc.

Radargun Scenario

- MonitorStart
- LoadData
  - **num-entries = 5000** show definition
- BasicOperationsTest
  - **key-selector = ConcurrentKeysSelector {numEntriesPerThread=0, totalEntries=5000 }** show definition
  - **num-requests = 100000** show definition
  - **num-threads-per-node = 5** show definition
  - **test-name = warmup** show definition
- ClearCache
- LoadData
  - **num-entries = 10000** show definition
- RepeatBegin
  - **from = 10** show definition
  - **inc = 10** show definition
  - **to = 30** show definition
- BasicOperationsTest
  - **amend-test = true** show definition
  - **duration = 1 mins 0 secs** show definition
  - **key-selector = ConcurrentKeysSelector {numEntriesPerThread=0, totalEntries=10000 }** show definition
  - **num-threads-per-node = 10** show definition
  - **test-name = stress-test** show definition

Radargun Results



iteration 0					iteration 1				
requests	errors	mean	std.dev	operation throughput	requests	errors	mean	std.dev	operation throughput
3504329	0	5.49 us	58.93 us	58405 reqs/s	3417207	0	4.41 us	49.89 us	56536 reqs/s

ROADMAP

Roadmap

- The roadmap slides have been removed from this presentation to avoid potentially stale content being broadly circulated. If you would like a briefing on the current roadmap - please contact your local Red Hat sales team

Questions?

TROUBLESHOOTING

Heap Dumps are your friend

- Snapshot of memory of a java process
- What it gets you
  - All objects and references
  - All classes, class loaders, static fields, super classes
  - Thread stacks and local variables
- What it does not do
  - Allocation information
  - Who created the object
  - Where was it created

Acquiring the dreaded heap dump

- There are several easy ways to acquire the heap dump:
  - jcmd <pid> GC.heap_dump /path/file.hprof
  - jmap -dump:live,file=my_jdg_stack.bin <pid>
  - Automatically through runtime flags
    - -XX:+HeapDumpAfterFullGC
    - -XX:+HeapDumpOnOutOfMemoryError
  - Invoke the MBean operation (i.e. jconsole, visualvm)
- Red Hat recommends the Memory Analyzer Tool for heap dump analysis in JBoss Developer Studio or Eclipse

Analyzing the Heap Dump (cont.)

Overview | Histogram

Class Name	Objects	Shallow Heap	Retained Heap
.*com.redhat.*	<Numeric>	<Numeric>	<Numeric>
com.redhat.summit.Customer	1,000,000	56,000,000	684,722,000
com.redhat.summit.PrintNamesServlet	1	24	272
com.redhat.summit.Resources	0	0	152
com.redhat.summit.InjectCustomersServlet	0	0	224
Σ Total: 4 entries (11,800 filtered)	1,000,001	56,000,024	

i Overview | Histogram | list_objects [selection of 'Customer']

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
com.redhat.summit.Customer @ 0x635614810	56	680
<class> class com.redhat.summit.Customer @ 0x60f8a2c50	8	8
notes java.lang.String @ 0x60e8d88c0 Here is some random notes on this customer	24	128
ssn java.lang.String @ 0x635614848 547668	24	56
firstname java.lang.String @ 0x635614880 Noah	24	48
lastname java.lang.String @ 0x6356148b0 Moore	24	56
address java.lang.String @ 0x6356148e8 31 chambers hill rd	24	80
city java.lang.String @ 0x635614938 North Falmouth	24	72
state java.lang.String @ 0x635614980 MA	24	48
zipcode java.lang.String @ 0x6356149b0 02556	24	56
email java.lang.String @ 0x6356149e8 josborne@redhat.com	24	80
Σ **Total: 10 entries**		