BigMemory for the Terracotta Server Array - FAQ

This FAQ explains what BigMemory is and how to use it.

About BigMemory

What is BigMemory?

BigMemory is a Terracotta product that allows Ehcache and the Terracotta Server Array to store data outside the JVM's object heap, in *off-heap* memory. This FAQ covers BigMemory for the Terracotta Server Array.

Today's servers can have a large amount of memory—16GB, 32GB, and more—but the long-standing problem of Java garbage collection (GC) limits the ability of all Java apps, including Terracotta server instances (L2s), to use that memory effectively. This drawback has limited the L2 to using a small Java object heap as an in-memory store, backed by a limitless but slower disk store.

BigMemory gives L2s instant, effortless access to hardware memory free of the constraints of GC.

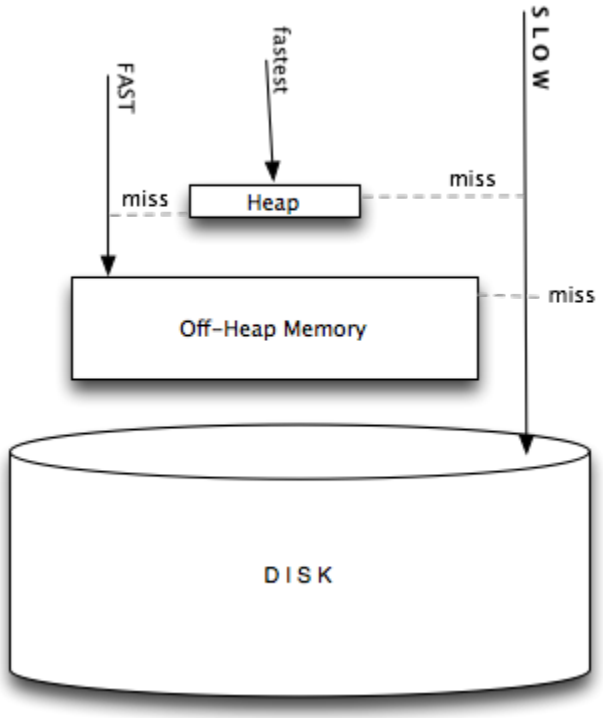
Exactly how does using BigMemory improve performance?

By reducing or eliminating the following performance bottlenecks:

- The number of Java GC operations that pause the server
- Faulting from disk on the server
- Database round trips

The resulting benefits deliver extremely low latencies for data access operations. In addition, a Terracotta cluster using BigMemory can handle the same amount of data with fewer stripes.

The following diagram illustrates how BigMemory adds a layer of off-heap memory storage that reduces faulting from the Terracotta server's disk yet remains outside of GC's domain.



How do I know if BigMemory will help me?

If you are prevented from bringing more data closer to the app because larger heap sizes cause significant performance degradation and response-time variability due to Java GC, BigMemory is the solution. Even if all the data you need is already in memory, cluster performance may be far below its potential due to the time L2 JVMs spend in GC. Additionally, BigMemory can shrink the number of L2s needed by increasing data density: Convert a large number of small-memory JVMs to a smaller number of large-memory JVMs. This in turn reduces cost and management complexity.

How do I see how much time I currently spend doing GC?

Turn on verbose Java GC logging using the option `-verbose:gc`. You can also monitor GC using tools such as JConsole, VisualVM or jstat.

How is expanding hardware memory better than expanding database or disk-based storage?

Both database and disk-based storage suffer from substantial latency overhead. And, unlike hardware memory, database license costs can balloon very quickly, while managing clustered databases can introduce unwieldy complexity. Disk-based storage may seem simple and cheap at first, but its cost and complexity can also grow in tandem with size.

Does BigMemory use JNI or non-Java code, or does it require a special version of Java or JVM?

No. BigMemory is 100% Java, and works with standard JVMs. 64-bit JVMs are recommend to make the most of the product.

Is BigMemory really just a custom garbage collector or does it require one?

No. BigMemory is compatible with all garbage collectors.

Is there documentation for BigMemory?

Yes – see the BigMemory section in the Terracotta Product Documentation.

Is BigMemory included in the open source release?

No, it is only available as a commercial offering. It is a separately licensed product available for Terracotta servers with an enterprise license. To access BigMemory, you must use the terracotta-ee kit. Trial downloads of enterprise kits with a trial license key are available for evaluation purproses.

Back to Top

Specifications

Can BigMemory be used with both standard and DSO installations of Terracotta products?

Yes.

What kind of scale does BigMemory offer?

Our tests showed that a Terracotta server instance with 8GB of off-heap memory is able to store 24 million entries—all in memory. That's up from 3 million entries with no off-heap. This 8x gain came with no degradation in throughput and delivered improved latency.

By delaying or preventing the costly swap-to-disk operations triggered when the L2 heap is low on memory, BigMemory can substantially improve performance. This allows each L2 to bear a heavier load, in turn allowing Terracotta clusters to manage the same amount of data with fewer L2s.

Does BigMemory require a specific OS or OS version, or OS patches?

No, although 64-bit operating systems are recommended to take full advantage of the product.

Which JVMs are supported?

Currently the Sun JVM is supported.

Where can BigMemory operate?

On L2s in a Terracotta cluster. BigMemory allows you to allocate off-heap memory for each L2 in the cluster.

How is off-heap memory allocated?

Before off-heap memory can be allocated, direct memory space, also called direct (memory) buffers, must be allocated. In most popular JVMs, direct memory space is allocated using the Java property `-XX:MaxDirectMemorySize`. This memory space, which is part of the Java process heap, is separate from the object heap allocated by -Xmx. The value allocated by `-XX:MaxDirectMemorySize` must not exceed physical RAM, and is likely to be less than total available RAM due to other memory requirements.

The off-heap memory you allocate to the Terracotta server is in configuration and must be at least 32MB less than the total allotted by `-XX:MaxDirectMemorySize`.

Do JVMs have a default value for `MaxDirectMemorySize`?

Sun HotSpot has a default equal to maximum heap size (-Xmx value), although some early versions may default to a particular value.

What are the maximum heap and off-heap sizes I can allocate for my JVM?

The maximum heap size of a Java application is limited by some key factors: the process data model (32-bit or 64-bit) and the associated operating system limitations, the amount of virtual memory available on the system, and the amount of physical memory available on the system.

Can I use this with a 32-bit JVM?

Yes, though the amount of heap-offload you can achieve is limited by the addressable memory. For a 32-bit process model, the maximum virtual address size of the process is typically 4 GB, though most 32-bit operating systems have a 2GB limit. The maximum heap size available to Java is lower still due to particular OS limitations, other operations that may run on the machine (such as mmap operations used by certain APIs), and various JVM requirements for loading shared libraries and other code.

A useful rule to observe is to allocate no more to off-heap memory than what is left over after `-Xmx` is set. For example, if you set `-Xmx3G`, then off-heap should be no more than 1GB. Breaking this rule may not cause an OOME on startup, but one is likely to occur at some point during the JVM's life.

Back to Top

Configuration and Tuning

How do I start using BigMemory?

See the documentation for BigMemory.

How much of my total available hardware RAM should I make available to the direct memory space using `-XX:MaxDirectMemorySize`?

Committing too much of a system's physical memory is likely to result in paging of virtual memory to disk, quite likely during garbage collection operations, leading to significant performance issues. On systems with multiple Java processes, or multiple processes in general, the sum of the Java heaps and off-heap stores for those processes should also not exceed the the size of the physical RAM in the system.

Note that `MaxDirectMemorySize` sets an upper limit for the JVM to enforce, but does not actually allocate the specified memory. Overallocation of direct memory (or buffer) space is therefore possible, and could lead to paging or even memory-related errors. The limit on direct buffer space set by `MaxDirectMemorySize` should take into account the total physical memory available, the amount of memory that is allotted to the JVM object heap, and the portion of direct buffer space that other Java processes may consume.

Note also that there could be other users of direct buffers (such as NIO and certain frameworks and containers). Consider allocating additional direct buffer memory to account for that additional usage.

What kind of tuning should be considered to ensure optimal memory performance?

Linux, Microsoft Windows, and Solaris users should review their configuration and usage of swappiness and hugepages. See the following for more information:

- http://www.pythian.com/news/1326/performance-tuning-hugepages-in-linux/
- http://unixfoo.blogspot.com/2007/11/linux-performance-tuning.html
- http://andrigoss.blogspot.com/2008/02/jvm-performance-tuning.html
- http://blogs.sun.com/dagastine/entry/java_se_tuning_tip_large

For 64-bit JVMs running Java 6 Update 14 or higher, consider enabling compressed references to improve overall performance. For heaps up to 32GB, this feature causes references to be stored at half the size, as if the JVM is running in 32-bit mode, freeing substantial amounts of heap for memory-intensive applications. The JVM, however, remains in 64-bit mode, retaining the advantages of that mode.

For the Sun HotSpot, compressed references are enabled using the option `-XX:+UseCompressedOops`. See the following for more information on compressed references:

- http://wikis.sun.com/display/HotSpotInternals/CompressedOops
- http://blog.juma.me.uk/2008/10/14/32-bit-or-64-bit-jvm-how-about-a-hybrid/

Back to Top