What I Learned This Month: Compressed References

Scott Chapman
American Electric Power

While 64-bit addressing is good from the perspective that it allows for addressing more memory than most of us can fathom, one issue is that a 64-bit system will generally use more memory to store the same amount of data than a 32-bit system will. This statement generally applies to any level of system, whether it is an operating system, a database system, or a Java Virtual Machine (JVM).

I ran into this recently on an IBM Java 6 JVM. Starting with WebSphere Application Server (WAS) v7, the default was changed to run the JVM for the application in 64-bit mode. In WAS 8.5's Liberty Profile[1], only a 64-bit JVM is supported.

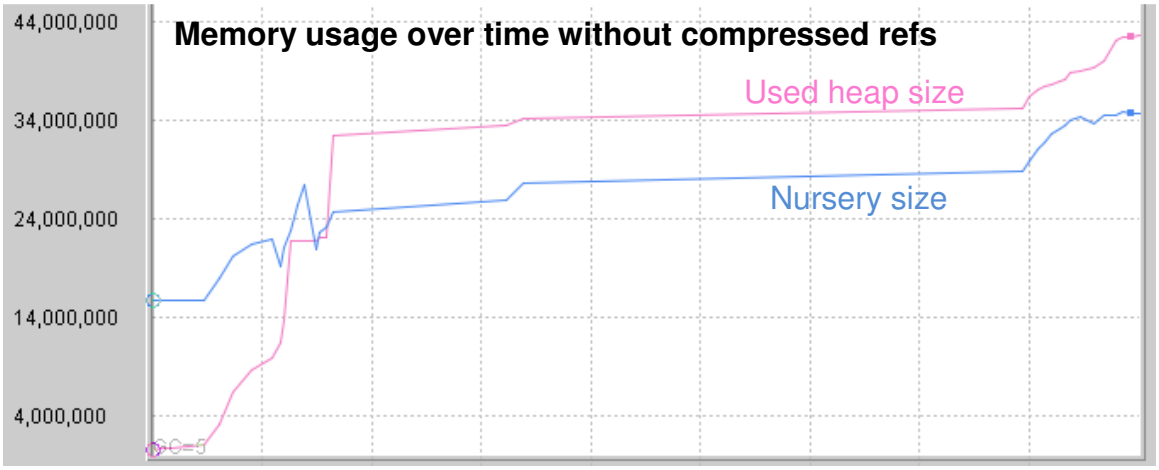
I was testing an application in the beta version of Liberty. I wasn't running a particularly large heap size, but I was nonetheless surprised when I took an out of memory error early in my testing. I didn't really expect that problem at that point in the application as it should not have been doing anything particularly memory intensive. So I simply bumped up the heap size a bit and tried again, which worked fine.

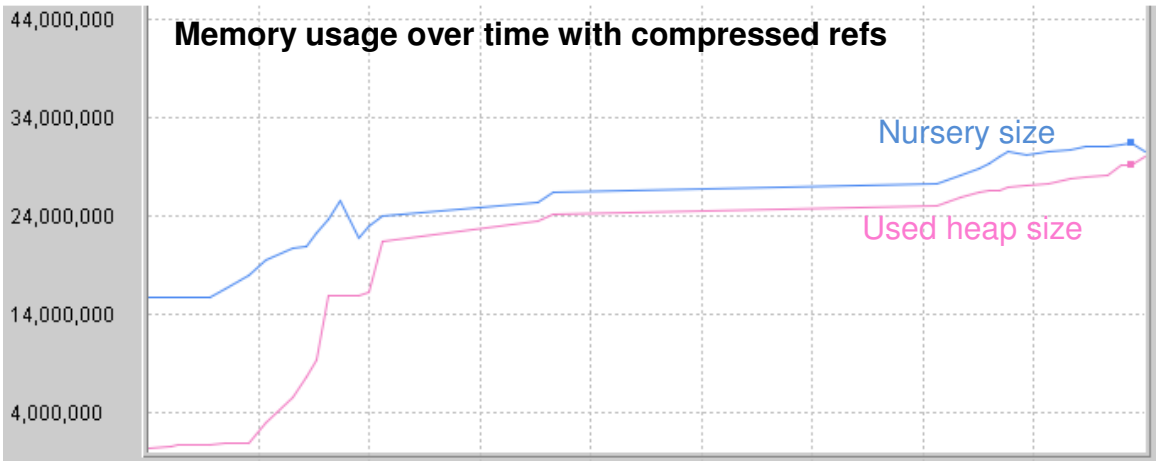
However, later I was thinking about it and mulling over the surprising need for that large of a heap for a relatively simple function when I remembered that the JVM in Liberty is 64-bit. Because the JVM manages objects on the heap, every object has to have a "reference" number so the JVM can locate it. What that number represents is, I believe, undefined in the Java specification. But it can be thought of as a pointer to a location in the heap, or perhaps an index into a list of objects. For very large heaps, you may have a very large number of objects and so the default for 64-bit JVMs is to have 64-bit reference numbers. But there is a JVM option to enable "compressed references", which change the object references from 64-bit numbers to 32-bit numbers. This necessarily limits the number of objects that the heap can contain but can greatly reduce the heap usage for applications that do not need scores of GBs for their heap. The IBM JVM option is -Xcompressedrefs. Oracle's JVM has a similar option.

IBM has been saying for a few years now that you should use compressed refs for 64-bit JVMs.[2] The performance improvements that they've measured by using compressed references are pretty significant. I'm a bit confused as to why that is not the default for WAS. Interestingly, I did find IBM APAR PK75174[3] that apparently made compressed refs the default for WAS 7 on all platforms other than z/OS. This, of course, doesn't help those of us running WAS on z/OS.

Since the z/OS default hasn't changed, one might wonder if it's really that big of a deal, despite numerous IBM presentations that strongly suggest that you use compressed references. Since I had an easy test case and changing it in Liberty was pretty simple, I captured the garbage collection (GC) trace for two executions of the initial bit of the application: once with compress refs on and once with it off.

I started the server and then ran the same function in both tests. Note that this was not meant to be a heap stress test; it simply was the initialization function of the application, a relatively simple test case. Once the function was done, I captured the verbose GC outputs and analyzed them in PMAT[4]. Those trace results are shown below. The used tenure heap curves (pink) are a somewhat similar shape, but clearly the one in the second test that used compressed references uses significantly less memory, finishing the test case at about 30MB vs. about 43MB. Similarly, the nursery size (blue) is about 10% smaller when compressed references were used.





Frankly, I was a bit surprised that compressed references reduced the heap size by almost a third in this simple test case. The results will vary by application, but we're certainly changing all our 64-bit WAS JVMs to use compressed references!

If you're curious whether your IBM JVM is already using compressed references or not, simply look for the "compressedRefs" attribute in the verbose GC output.

As always if you think I got it all wrong or have questions or comments, you can reach me via email at sachapman@aep.com.

---

[1] Liberty Profile is a new "feature" of WAS that allows you to run a limited-function application server to support applications that don't need the full J2EE support. On the mainframe that

means 1 address space per LPAR + 1 address space per server instead of the multitude of address spaces that traditional WAS seems to spawn.

[2] See for example: ftp://public.dhe.ibm.com/software/webserver/appserv/was/WAS_V7_64-bit_performance.pdf

[3] See: http://www-01.ibm.com/support/docview.wss?uid=swg1PK75174

[4] IBM Pattern Modeling and Analysis Tool for Java Garbage Collector. This is a free tool that helps you visualize and analyze the garbage collection activity in the IBM, Sun/Oracle, and HP JVMs. You can find it at: https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=22d56091-3a7b-4497-b36e-634b51838e11