Hi Harish,

We have the following questions:

IPP
Native heap – DRAM0
JVM heap - DRAM1
Strings/Bitmaps - ? DRAM0

IPN
Native heap – DRAM0
JVM heap - DRAM1
Strings/Bitmaps – DRAM1

Questions:
1) Strings&Bitmaps and Native heap memory are not same? - No, not the same. Also from what I understand bitmaps/strings are pinned and not moved so garbage collection and compaction can not move these items to recover memory.
2) We are using 66M for normal heap and 20M for native heap, is there any base for setup this limitations, what could be the percentage of RAM memory, we can use for Heap and native usage. It is appreciable, if you could tell us the percentage of memory that needs to be used for other processes in the box. These settings were specified by NSN/Myrio. There is a minimum amount of memory required by the Linux os after which oom-killer will kill the siege process. Normally in a healthy system I see 7 to 8 meg of memory available when checking using vmstat.

Native heap allocation in both the IPN and IPP is coming out of DRAM0 which is also used for system memory.
In the IPP box the JVM heap is allocated from DRAM1 which allowed us to free ~66Mb in DRAM0 (system memory)
In the IPN box the JVM heap and Native heap are allocated out of DRAM0 along with system memory. Strings/Bitmaps are first allocated out of DRAM1 and will allocate out of DRAM0 once DRAM1 available space is exhausted.

You can increase the max limit for native heap but this will take away from system memory. On the IPN box this is very limited and would lead to OOM-killer.

---

1. What happens on an overflow of the native heap memory? Will this be allocated in the normal heap?
2. Can we to avoid overflow increase native heap without further consequences?
3. The GC's shown in the log files, are they independent on the kind of memory ie are they both showed for
4. Normal heap and native heap?

Native heap overflow - From what I understand you can not over commit native heap, once you've used up the allocated memory for native heap siege does not try to pull from system memory or from the JVM heap to fulfill a memory request in the native space. That is why GC is kicking in trying to free up memory to honor the request. I've copied Skelmir in case I am wrong.

Increasing Native heap size - There is always a trade-off. You can make native heap larger but do you want to steal the memory from the system or the JVM heap? I don't believe there to be much head-room and to take more could push us into out-of-memory condition in which the OOM-killer would kick in. Wouldn't it be better if the client cleaned up what isn't needed to reduce the amount of heap used, does so much really need to cache?