

Memory Management in the ScrumWorks® Pro Server

Introduction

This article attempts to explain how the ScrumWorks Pro server manages memory resources.

Its intended audience includes systems administrators or product evaluators in charge of running and maintaining a ScrumWorks Pro installation.

Memory Management Fundamentals

In general, the management of application memory involves supplying the memory required for a program's objects and data structures from the limited resources available, and recycling that memory for reuse when it is no longer necessary. Because application programs cannot, in general, predict how much memory they will require, they need additional code to handle their changing memory requirements. In the case of the ScrumWorks Pro server (or any Java applications for that matter), memory management is handled entirely by the Java Virtual Machine (JVM).

Application memory management combines two related tasks:

Allocation

When the ScrumWorks Pro server requests a block of memory, the JVM must allocate that block out of larger blocks it has received from the operating system. The part of the memory manager that performs this operation is known as the allocator. Dynamic Allocation is the process of allocating necessary memory during program execution. Heap Memory refers to the memory area that is managed by Dynamic Allocation.

Recycling

When memory blocks have been allocated, but the data they contain is no longer needed by the program, the blocks can be recycled for reuse. In Java, the memory manager is the Garbage Collector and it employs what is known as automatic memory management.

The JVM Garbage Collector must work within several constraints, including:

CPU Overhead: The additional time taken by the JVM garbage collector while the program is running.

Interactive Pause Times: The amount of delay a user experiences.

Memory Overhead: The amount of space wasted for administration, internal fragmentation, and poor layout.

Several JVM options are available to help negotiate these constraints.

Even with automatic memory management or garbage collection, it is possible to experience the phenomenon known as **Memory Leak**, in which the program continually allocates memory without



recycling it, thus eventually running out of memory. ScrumWorks Pro does not suffer from this anomaly and the rest of this article will attempt to explain why, at first, it might appear to be leaking memory, but, in reality, is not.

ScrumWorks Pro Memory Utilization

ScrumWorks Pro is a client/server application written in Java. In this article, we will focus on memory management at the server side. To understand why ScrumWorks does not leak memory, one must first understand how JVM garbage collection works.

A detailed explanation of how the JVM GC works is unnecessary here, but such information can be found in the reference section at the end of this document. For our purposes, it is better to use a simplified model to explain how the JVM behaves with ScrumWorks and why memory allocation growth is expected up to a predefined limit (current statistics can be found at the end of this section).

Because ScrumWorks Pro is a distributed client/server application, the demand for memory resources varies according to several factors, including the number of simultaneous users, hardware specifications, operating system specifications, and, most importantly, the size of the ScrumWorks Pro database.

The default JVM Memory configuration settings that ScrumWorks is shipped with should be sufficient for most users, but, depending on the factors above, it may be necessary to tweak these settings. (For details on how to adjust the memory settings, consult the “ScrumWorks Pro Server Performance Optimization” document.)

For the purposes of this discussion, there are four different kinds of memory:

JVM Memory (to run ScrumWorks) and OS Released Memory to run the JVM such that:

- 1) OS Released Memory = JVM Memory + Overhead Memory
- 2) JVM Memory = Heap Memory + Non-Heap Memory (PermGen)
- 3) Heap Memory = Allocated Heap Memory + Used Heap Memory
- 4) Non-Heap Memory = Allocated Non-Heap Memory + Used Non-Heap Memory

At a given instant the total amount of memory consumed by ScrumWorks Pro and released by the OS is:

OS Released Memory = Used Heap Memory + Used Non-Heap Memory + Overhead Memory.

The maximum amount of RAM memory that can be made available to ScrumWorks Pro by the OS is:

OS Released Memory = Allocated Heap Memory + Allocated Non-Heap Memory + Overhead Memory.



Allocated Heap Memory is the amount of memory that ScrumWorks Pro can utilize for dynamic allocation. From above, we can see that this value does not represent the total amount of memory consumed by the ScrumWorks Pro server, but its value limits how much memory the OS can release to the JVM running ScrumWorks Pro. The default maximum size for “allocated heap memory” for the ScrumWorks Pro server is 768Mbytes.

Used Heap Memory is the portion of the heap memory currently in use by live ScrumWorks data structures (objects). This value can grow up to the maximum total amount reserved for Allocated Heap Memory, but never exceed it.

Allocated Non-Heap Memory is used to hold meta-data about the JVM itself. Current maximum default is 64Mbytes. It represents the maximum total amount of non-heap memory that the JVM can utilize for ScrumWorks.

Used Non-Heap Memory is the portion of the allocated non-heap memory currently utilized.

Overhead Memory is memory utilized for the garbage collector’s own internal data structures, thread stacks, fragmentation, etc.

ScrumWorks Currently Configured Maximum Values:

OS Memory	Allocated Heap Memory	Allocated Non-Heap Memory	Estimated Overhead Memory
=~ 1024 MB	768MB	256MB	0.3 * 768 = 230MB

Does the ScrumWorks Pro Server leak memory?

If we were to take an X-ray of the JVM memory while running ScrumWorks, we would see the **Used Heap Memory** growing and shrinking while the application is utilized. Once heap memory grows close¹ to the maximum available allocated heap memory, the JVM may request to the OS more allocated heap memory, up to the maximum established, or garbage collect memory to recycle unused memory or both, so that more heap memory is made available for utilization.

To the OS, the total memory consumption is growing, because the JVM rarely gives **Allocated Heap Memory** back. The Garbage Collector frees up used heap memory and *occasionally* allocated heap memory. However, memory allocation will never grow beyond the established limit.

The “Hypersonic” distribution of ScrumWorks utilizes an in-memory database. So the used heap memory will continually grow to accommodate new objects as new data is entered into the application. For large databases, it is advisable to increase physical memory and tune the JVM settings appropriately or migrate to the “MySQL” distribution.

The information in Figure 1 was taken from ScrumWorks under relative load, utilizing a memory profiler tool. The Blue curve represents Used Heap Memory and the red curve represents Allocated Heap Memory.

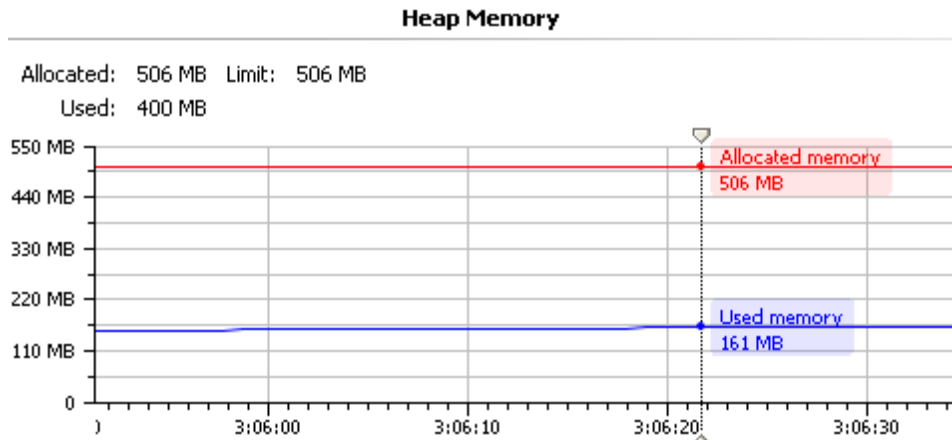


Figure 1

Figure 2 represents ScrumWorks Pro under a stress test scenario in which the database is filling up. Notice the used memory is growing.

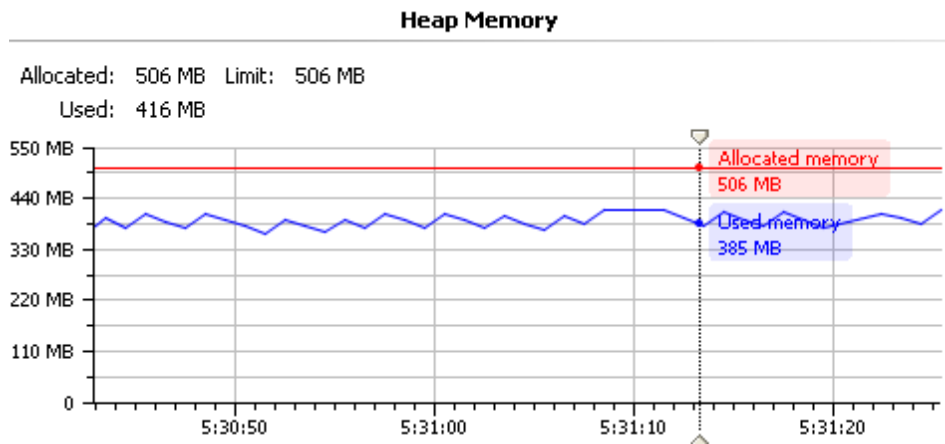


Figure 2

Figure 3 represents growing memory usage during stress test scenario.

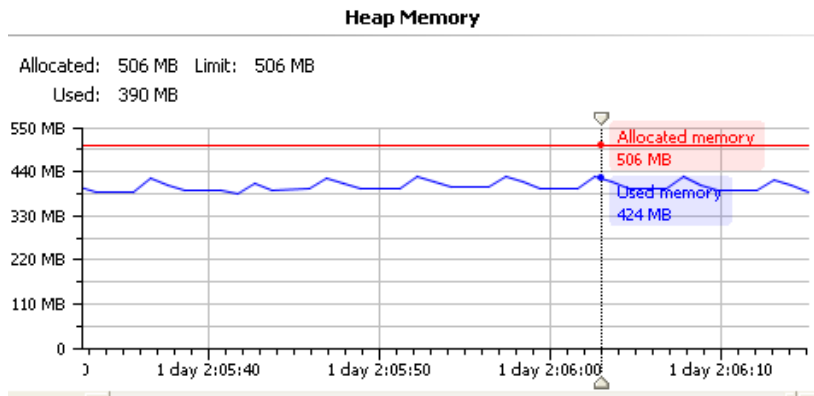


Figure 3

Figure 4 shows the Memory Usage as reported by the OS for the above Java heap Memory Report. As the blue curve grows, the Task Manager shows growing memory usage.

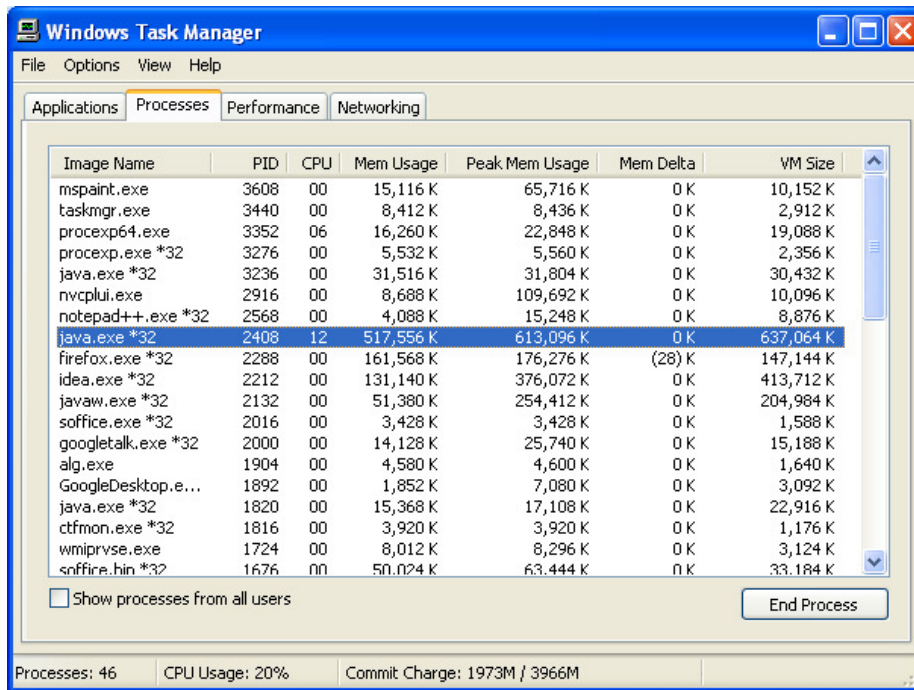


Figure 4

Figure 5 represents ScrumWorks at the end of the stress test. Notice that unused memory is now garbage-collected.

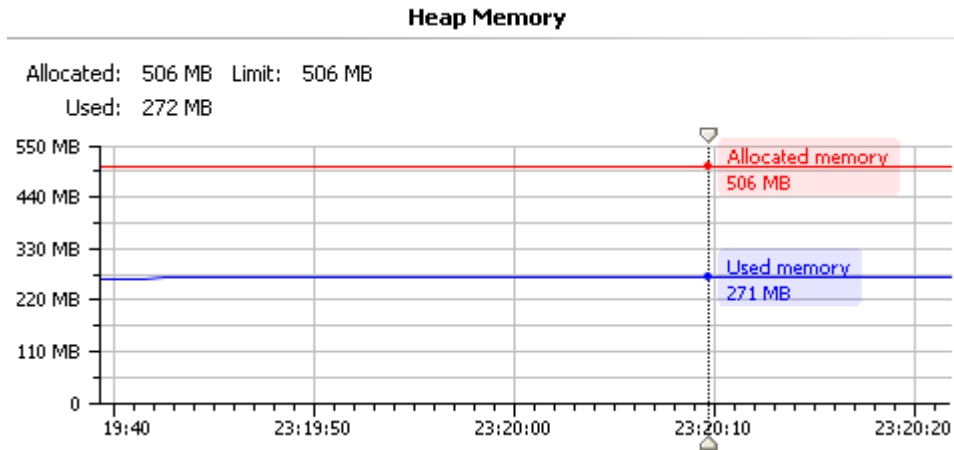


Figure 5

Note: For this test scenario, the ScrumWorks Pro server was configured with a limit of 512MB for Allocated Heap Memory and 128MB for PermGen space.

Conclusion

Memory will grow up to the currently imposed JVM limit, but never exceed it. As the ScrumWorks in-memory database grows, so does the **Used Heap Memory**. For large databases, the amount of available RAM and JVM settings requires adjustment for optimal performance.

Reference

[1] Tuning Garbage Collection with the 5.0 Java[tm] Virtual Machine:

<http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html>