Advanced performance tuning concepts

Look at key system areas and JVM tuning

Skill Level: Intermediate

Sean Walberg (sean@ertw.com)
Network Engineer
Freelance

28 Apr 2009

The performance of even the best application suffers if the underlying host is not configured properly. This article looks at the four key areas of performance tuning and identifies what to watch for in each of them. In addition, Java™-based applications bring other performance tuning requirements with them, especially the garbage collection cycle. This article also looks at what you need to know about garbage collection.

Your server has many settings that you can change to make it handle its workload better. A file server is tuned differently from a database server, and two application servers may be tuned differently, depending on the nature of the servers' loads. Tuning involves allocating the server's finite resources to different parts of the operating system and application so that the application responds as well as possible. The following are the areas to consider:

- Central processing unit (CPU)
- Memory
- Disk (both space and access speed)
- Network

These areas are often intertwined. For instance, you can allocate memory for caching, which reduces disk access, or access resources over the network rather than reading off the local disk. A particular focus of this article is on memory tuning with regards to the Java Virtual Machine (JVM). The JVM has its own memory management system that you must monitor and configure.

CPU

A server's CPU spends a lot of time waiting for things to happen. Most often, it is waiting for data to return from disk. Multitasking allows the CPU to do other things while it waits. Therefore, buying a faster CPU provides a benefit when the host is spending a lot of time in the CPU state.

The `vmstat` command gives a good real-time breakdown of where a system is spending its time, and the `sar` suite of tools is good for longer term monitoring. If these tools show that your CPU is spending most of its time in user space and the idle cycles are dwindling, it is time to look at what to do next. In this situation, you either shuffle load to another server or increase the CPU capacity.

Shuffling load may mean running batch jobs on another server or splitting the application load onto multiple servers. This last case is ideal, otherwise known as horizontal scaling. If you have to increase CPU capacity, you can either do a physical upgrade, such as adding more CPUs, or reallocate more resources if you are in a virtual environment.

Some loads do not lend themselves to parallel processing, so splitting the load across multiple servers or adding more CPUs doesn't help. In this case, you find yourself throwing faster CPUs at the problem and spending some time optimizing the underlying code to make it run in fewer cycles.

Memory

There are many items involved in memory tuning. The simplest is ensuring that you have enough RAM to hold your application without needing to use swap space. The operating system's virtual memory subsystem allows applications to allocate more memory than actually exists on the system, with the shortage being made up by temporary storage on disk. Paging a memory block out and back in from disk is significantly slower than accessing it directly from RAM, so you should generally avoid this practice.

That said, the virtual memory subsystem does need some tuning because blocks can be written to disk before the system runs out of memory, and sometimes accidents do happen and you have to dip into swap space. The main thing to check is when virtual memory is used. As the free memory on a UNIX® system is allocated, the kernel eventually decides that it has to start looking for memory pages that are candidates to be swapped out. Soon after that, the kernel starts swapping some of those pages to disk in anticipation of having to allocate memory to the process that's allocating memory. If you know you have enough memory to handle your workload, you are better off delaying these two swap activities.

In Solaris™, you do this through the tunables in /etc/system. In the IBM® AIX® operating system, use the vmo command. In Linux®, use `/etc/sysctl.conf`. These behaviors are under constant flux as operating systems evolve, so research the topic carefully before making any changes.

Finally, having extra memory on hand allows the file systems to cache files and metadata in memory. Most UNIX systems try to use free memory for caching, which is why it often looks like your system has no free memory. This saves disk activity, which can be important under some workloads such as Web servers.

Disk

Disk is slow compared to memory, so excessive disk activity is the downfall of many applications. Disk activity can be due to swapping, described earlier, or it can be due to requests from the application or operating system. Excessive logging can cause contention on a disk.

Your best tool for spotting disk bottlenecks is `iostat`. This tool tells you how many reads and writes are happening at a particular point in time and how saturated your disk controllers are. If you have several disks, splitting load onto separate spindles is an effective way of making reads and writes faster because the largest component of disk latency is the seek time. Files that grow consistently, such as log files and database journals, are on separate disks from your application's disk and database.

Both `vmstat` and `iostat` report the percentage of time a system spends in iowait, meaning the CPU is idle but the system is waiting for IO to return. A high iowait value can be an indication of a slow or overloaded disk.

Closely related to disk is the number of file descriptors that can be open. If you run out of file descriptors, attempts to open files will fail. Usually, the `ulimit` command is enough to increase the number of available file descriptors, though your operating system may have kernel limitations preventing the `ulimit` from succeeding.

Network

The network is critical to most applications because it moves data between the servers and back to the client. A slow network means the application seems to respond slowly. The first thing you should do is make sure all servers are locked down to the highest speed possible using full duplex and the switch port is set to match. Speed and duplex mismatches between the switch and the server are frequent contributors to network problems.

Your operating system allocates various buffers for network resources. For example, the operating system maintains the TCP send queue for each TCP connection. This queue holds data that the application has sent but has not been acknowledged by the remote end (and may not have even been sent on the network depending on how many packets are unacknowledged). If this queue is full, the application is not allowed to send any more data until the backlog is cleaned up.

You can look for signs of buffer congestion with `netstat -s`, which prints a list of network counters. Anything with the words "queue" or "overflow" in it is related to the TCP queues and should be monitored. These counters are generally reset only at boot, so you are more concerned with numbers that grow over time.

If `netstat -an` shows a large number of connections in a wait state, such as `CLOSE_WAIT` or `FIN_WAIT_1`, it might cause problems establishing new connections because all the system's resources are holding these stale connections. In this case, you want to reduce the timeouts that control how long the operating system holds the connections, such as with `ndd` for Solaris or `no` for AIX.

Looking deeper into Java memory

The previous sections covered four broad areas where a system needs tuning. One of those sections was memory. In a Java application environment, a server allocates memory to the Java process that is responsible for running the application code. That Java process is the JVM, which is itself responsible for allocating memory to the underlying application.

From the operating system layer, you may see a gigabyte of memory allocated to a Java process. Inside that process, the JVM manages the heap, which is where the memory for new objects comes from. As objects are created, they are placed on the heap. As they are destroyed, they stay on the heap. A process known as *garbage collection*, run by the JVM, marks all known created objects and then cleans the rest of the heap for further allocations. At this point, the heap can be expanded (if the garbage collection didn't reclaim enough memory for the new allocation) or shrunk (under certain conditions where the JVM believes the heap is too large).

From this simplistic definition of garbage collection, you can deduce that the system is not doing any application work while garbage collection is happening. The JVM is effectively paused during the garbage collection run. Therefore, much of Java tuning involves determining the optimal memory sizes for the heap and fine tuning the garbage collection process.

The rough idea behind tuning the garbage collection process is to understand how often it is run and the conditions that cause it to run, and then change the JVM settings to minimize the impact of the garbage collection run.

Collecting garbage collection information

The first step to understanding how garbage collection is affecting your application is to gather information about when and how garbage collection is being performed. Enable verbose garbage collection logging in your JVM, which will start logging garbage collection activity. In IBM WebSphere™ Application Server, you can find this setting in your administration console by navigating to **Application servers > *server name* > Process Definition > Java Virtual Machine** within the Integrated Solutions Console and selecting **Verbose Garbage Collection**.

Alternatively, start your JVM with the `-verbose:gc` parameter (which is what the Integrated Solutions Console option does anyway). Either way, your JVM's output will now include the garbage collection information.

The unfortunate part about enabling verbose garbage collection logging is that the format of the file is not consistent between vendors, or even between different versions made by the same vendor. IBM's Java Runtime Environment (JRE) 6.0, for example, logs in a verbose Extensible Markup Language (XML) file format. Sun Microsystems' HotSpot JVM, on the other hand, uses a terse one-line format that sometimes needs extra command-line parameters enabled to get the information you need.

Making sense of garbage collection data

Now that you are logging information from the garbage collector, run your application under normal load. Then, examine the garbage collection logs. You see the size of the heap grows from the initial allocation and eventually settles around a range of values. You can then use a value from this range as the initial size of the heap, which will prevent the initial delays associated with growing the heap to the steady-state value.

The garbage collection logs also indicate the time at which the collection appears and the length of time the garbage collection takes. If you find the length of the garbage collection runs are too long, you can tune your JVM to use a different garbage collection algorithm (the details depend on the version and vendor of your JVM). From these timestamps, you can also calculate the percent of time that the system spends in garbage collection, which you can use to compare various JVM settings.

If you find the garbage collection process is constantly growing and shrinking, you can change the ratios the JVM uses to determine when to expand or shrink, known as the `MinHeapFree` and `MaxHeapFree` values.

As the JVM evolves, so does its performance with respect to garbage collection. Your JVM's manual is the best source of current tuning parameters.

Tuning priorities

IBM has some recommendations for the areas that you should check when tuning your UNIX server for WebSphere Application Server.

First, make sure your server has the resources it needs: CPU, disk, memory, and network. These are fundamentals.

Next, understand your application's garbage collection requirements and tune the JVM accordingly. This may require that you to go back to the previous step to make sure you have enough memory to run your application the way you need.

Make sure your application server queues are such that the application server can only be served the requests it can handle. When a request comes to the Web server to be handed off to the application, it goes through a queue. If you allow too many connections in to the application, then you cause everyone to experience bad performance. Instead, queue excess connections on the Web server and keep them away from WebSphere Application Server.

Finally, there are many caches that are used, from prepared statements in the database to Enterprise JavaBean (EJB) technology and thread caches. If the caches are constantly being purged to make room for new entries, you should increase them.

Conclusion

A computer's resources fall into CPU, memory, disk, and network. Your tuning endeavors should center on measuring these resources and making adjustments in your application, application server, and servers to make sure there is no contention for resources.

The JVM manages its own heap and cleans it through a process known as garbage collection. Your tuning efforts in this area involve making sure the heap can grow to the size your application needs and tuning the garbage collection parameters to avoid the effects of heavy garbage collection.

Your primary tool in garbage collection tuning is enabling verbose garbage collection tracing, which logs each garbage collection activity. From there, you can determine how long the collection cycle is taking and the reason for the collection.

Resources

Learn

- Sun Microsystems' documentation for their garbage collection routines (v5 or v6) gives valuable information about how the various garbage collection algorithms work and how they came to be.
- The XML format for the IBM JVM's verbose garbage collection logging is buried deep within the Java Diagnostics Guide.
- See the WebSphere Application Server documentation for UNIX performance tuning.
- "Sensible Sanitation -- Understanding the IBM Java Garbage Collector, Part 1: Object allocation" (developerWorks, August 2002) is the first in a three-part series about JVM garbage collection. It's a bit dated because it doesn't include some of the latest garbage collection algorithms, but it is still relevant for its descriptions of garbage collection concepts and how memory is allocated on the heap.
- For system tuning, it is vital that you understand how to use the vmstat command.
- This presentation about AIX Performance tuning contains many commands you can use to find and fix bottlenecks on your AIX server.
- If you are running Solaris, IBM has documentation about tuning Solaris systems.
- If you are running AIX, this document about tuning AIX systems is worth a read.
- If you are working with WebSphere, you must read IBM's WebSphere Application Server V6 Scalability and Performance Handbook. It's 1100 pages of advice on how to tune WebSphere and other components, and how to write your applications with scalability in mind.
- Another IBM Redbook is Running IBM WebSphere Application Server on System p and AIX: Optimization and Best Practices, which describes horizontal and vertical scaling as it applies to a WebSphere application. The document also discusses ways to tune your application server and operating system.
- Browse the technology bookstore for books on these and other technical topics.
- The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration.
- developerWorks technical events and webcasts: Stay current with developerWorks technical events and webcasts.
- Podcasts: Tune in and catch up with IBM technical experts.

Get products and technologies

- Download the IBM Pattern Modeling and Analysis Tool for Java Garbage Collector, which helps you interpret your garbage collection logs and figure out what your JVM settings should be.

About the author

Sean Walberg
Sean Walberg has been working with Linux and UNIX systems since 1994 in academic, corporate, and Internet service provider environments. He has written extensively about systems administration over the past several years. You can contact him at sean@ertw.com.

Trademarks