PERFORIC

performance is the key

Why Software Performance Testing Should be Like Dyno Testing an Automobile Engine

and other application performance considerations

March 2010

Application performance can impact revenue, customer satisfaction, employee productivity, data center efficiency and software and hardware licensing costs.

Software performance tuning is a lot like dyno tuning a car engine: every engine is tested on a dyno during the later phases of development to check for power and endurance. If you buy a car advertising a certain amount of horsepower, you want to be sure that you are getting what you paid for. In the world of IT, we would like the same to be true of software.

Automobiles have been around a long time and, as a result, there are many standards in place for testing the power and efficiency of an engine. Unfortunately, the same cannot be said of software: when you buy a piece of software or develop it in house there are few benchmarks and standards available to test the performance and stability of that software. Most development includes a Quality Assurance (QA) phase, but usually it's functional QA designed to ensure that all the buttons and widgets function as designed. In automotive parlance, this would be like driving a car at 5 mph on a smooth surface to make sure the wheels don't come off.

Most real world driving includes going much faster than 5 mph, emergency braking, the occasional bumpy ride and, at times, overloading the car past its maximum number of occupants. This is known as stress testing. It ensures nothing catastrophic happens under both normal and less than ideal conditions. Of course, engine durability and reliability are also tested. Only once it passes these and other tests does an engine go into production.

Dyno Testing for Software

Unfortunately, most software QA testing today does not have this level of rigor. The gap between base functionality testing and certified product testing can be addressed through formal performance testing. In very large companies, it's more common for enterprise applications to be performance tested as the necessary skill sets may exist within the IT group. In small and mid-size enterprises, however, it's rare to find this kind of expertise in house. As a result, some organizations take advantage of qualified third party providers for these services.

Performance testing typically is done after QA has already gone through a few rounds of functionality testing. After all, it's important to ensure that the software is functionally stable before testing. If an engine is being tested for durability, but has a design flaw that causes it to overheat, then

When software fails, it should not cause undue damage to users, systems or data.

the durability test will not run as intended. Similarly, software reliability testing should be done on a fairly stable build so that functional issues do not interfere with testing.

Performance testing can take on a variety of forms, depending on the goals of the test and the software's intended use.

Reliability Testing – covers software durability. This involves tests that run a low load for an extended period of time—usually 24 to 48 hours. Things like memory and database connections are monitored to reveal any memory and connection leak issues.

Stress Testing - identifies the breaking point of an application and ensures that it fails gracefully. When an auto engine fails, it should not blow up and harm the occupants in the vehicle. Similarly, when software fails, it should not cause undue damage to users, systems or data.

Scalability Testing - shows how the application performs as the load is gradually increased. This is like stepping on the gas to see how fast a car accelerates.

If enterprise applications were put through performance tests as rigorous as those experienced by an automobile engine, they would hold up much better than they currently do.

Capacity Planning

Capacity planning is another important aspect of the performance testing process. This is the art of ensuring that the hardware allocated for a production system is both adequate and efficient. With the increasing cost of energy and today's green mentality, it's imperative that the data center isn't wasting electricity running software that only utilizes a small portion of the hardware allocated. Hardware efficiency also reduces licensing costs for software and hardware in addition to reducing administrative costs.

So how does one go about determining the most efficient configuration for an application? Take, for example, a consumer-facing web application: the first step is to figure out what your customer base looks like as well as their habits. For example:

The science of capacity planning can assure a responsive web site that does not consume more resources than it needs.

- A telecommunications company might have customers checking their bills on the 1st and the 15th of the month, depending on how many billing cycles you have.
- A credit card company might have more traffic on the weekends, when people typically use their credit cards more and want to check their balances.

Once peak times have been determined, the next step is to calculate the maximum number of users that can be expected to access the site at any given time during those periods. These are called concurrent users in a system. The configuration needs to support the maximum number of expected concurrent users. Super efficiency can be achieved by things like smart grid computing which can add and remove hardware based on need, but that is still not a common solution and can lead to stability issues. For most purposes, as long as peak load periods can be accommodated without having excess capacity beyond that, the configuration can be considered efficient.

A variety of calculations and spreadsheets can be used to model the right configuration, but the data needs to be based on the behavior of the system in question. This can only be determined by running a small-scale production system in a lab and understanding the software's characteristics:

- Is the bottleneck in the database or the application tier?
- What is the ratio of application to database servers needed?
- Is there a disk I/O bottleneck?

The answers to these questions will be different for each application, which is why it is so critical that some testing is done in a small scale environment. If the deployment application server tier will have 32 CPUs, the test environment for understanding the application footprint under load can consist of as few as 4 CPUs. There are various formulas for extrapolating from 4 CPUs to 32. One of the key characteristics looked for during initial testing is linear scalability. This means if the load is doubled, the response time remains the same. In other words, if the response time is 2 seconds for 100 users on 1 CPU, it remains 2 seconds for 200 users on 2 CPUs and for 400 users on 4 CPUs. Keep in mind that performance cannot scale in a linear fashion indefinitely; take care when analyzing the results taken from a test environment. At some point, scalability will level off; determining when that will happen is as much an art as it is a science. However, the science of capacity planning can assure a responsive web site that does not consume more resources than it needs.

Monitoring tools, though less informative, often provide a better way to diagnose an application problem under load.

Performance Monitoring

Monitoring a system is also important to application performance and it can provide a lot of useful runtime data. It differs from profiling a system and can be done in a way that does not incur any overhead and is non-intrusive so as not to cause any issues. Monitoring is a useful way to gather data on a production or live system that, due to its nature, cannot be impacted as a result of the monitoring process. It also does not require any changes to the code itself because it happens at a lower, system level.

Profiling, on the other hand, is typically more invasive and can add overhead to a running application. Profiling tools can add anywhere from a 5 to 35% overhead due to the profiler itself. Frequently, profiling tools require changes to the code or the way an application is started. These tools are very useful during the development phase and can uncover bottlenecks and other code issues; however, they should be run on a test system and not in production. Frequently, attaching a profiler can cause a system to crash under load imposed by the profiler itself. Monitoring tools, though less informative, often provide a better way to diagnose an application problem under load.

A variety of system monitoring tools are offered via Linux. As opposed to profilers that need to be purchased separately, these tools are operating system commands that come as part of the Linux operating system. The commands are text-based, so the most useful way to look at the data is to graph it through a spreadsheet tool like Excel. Visual representation of the data can show memory or CPU issues with an application.

The following are some ways to gather CPU, memory and disk I/O information about a system. These commands can help identify system level problems, but in order to isolate the code causing the problem, they should be used in conjunction with a code profiler.

CPU

Command: vmstat –n 2 > output.txt

Description: vmstat displays memory and CPU information, among other things. **–n** tells the command to only display the header once. This makes it easier to graph the data. '**2**' is to display statistics every two seconds. The output is piped to **output.txt**. The output is fairly small, so this technique can be used to gather data for extended periods of time without worrying about disk usage.

Graphing data: To display percent CPU time spent on running user or non-kernel code, graph the 'us' column in a spreadsheet. For percent CPU time spent on kernel or system level code, graph the 'sy' column. The 'id' column contains percent CPU time spent idle. Idle time is a good way to gather total time (kernel + non-kernel). Since the values are displayed in percents, 'percent total busy time spent by system' = 100 – % idle time. Please note that the stats collected here are system-wide and not per-process. You can visually inspect a running system with a command like "top" to identify the top offending processes.

Memory

Command: vmstat –n 2 > output.txt

Description: as above

Graphing data: The 'free' memory column displays free RAM (in kilobytes) on the system. Graphing this column can uncover any memory leaks in the application. If the free memory on the system continues to decrease over time, it's a sign of a potential memory leak. Linux memory management can make this a bit tricky as Linux grabs unused memory for its own housekeeping and caching, but if your application or system runs out of memory, this graph should help identify the problem. (Linux will release memory from its own cache when an application needs it).

Disk I/O

Command: iostat –c 2 > output.txt

Description: iostat collects disk level I/O information, among other things. **–c** displays an abbreviated report that includes disk input output. '**2**' is to collect the stats every two seconds. The output is redirected to **output.txt**.

Graphing data: The % I/O wait column should be graphed to indicate whether or not the disk is a bottleneck. On a system where disk I/O is not an issue, this value should be less than 5%. If this value starts to increase, it's a sign that the disk subsystem is not able to keep up with the requests made on it. Either a faster disk should be considered or the application should be redesigned to reduce the amount of disk I/O by using techniques like caching data files.

Summary

One of the biggest concerns for both businesses and their IT organizations is uptime: application performance can impact revenue, customer satisfaction, employee productivity, data center efficiency and software and hardware licensing costs. There are several ways to manage application performance including performance testing, benchmarking, capacity planning and performance monitoring. These services are available both on-premise and as hosted service offerings and can help ensure the health and performance of enterprise applications, both before launch and while in production. Organizations of all sizes should consider these services whenever application uptime is critical to business productivity.

PERFORIC
performance is the key

Perforic LLC
955 Massachusetts Avenue, #309
Cambridge, MA 02139
info@perforic.com