WHITE PAPER

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

and other application performance considerations

Application performance can impact revenues, cus-tomer 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 need to know that you are getting what you paid for.

Automobiles have been around a lot longer than computers and there are many standards in place for testing the power and efficiency of an engine. Unfortunately, when you buy a piece of software, or develop it in house, there are few benchmarks and standards for the performance and stability of that software. Most development includes a Quality Assurance (QA) phase, but usually it's functional QA; i.e., do 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 these stressful conditions. Engine durability or reliability is also an important test. Only once these set of tests are passed 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 the formal process of performance testing. In very large companies, it's more common for enterprise applications to be performance tested as this capability may exist in the IT group. In small and mid-size enterprises however, it's rare to have this kind of expertise in house. So some organizations take advantage of qualified third party providers for these services.

Performance testing typically is done after QA has gone through a couple of rounds of functionality testing. It's important that the software is func-tionally stable before performance testing. If an engine is being tested for durability but has a design flaw that causes it to overheat, then the du-rability test will not be able to run for the intended amount of time. In the same way, software reliability testing should be done on a fairly stable build so that functional issues do not get in the way of the tests.

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

There are various kinds of performance tests.

Reliability Testing – covers software durability. This involves tests run with 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 to make sure it fails gracefully. When an auto engine fails, it should not blow up and harm the occupants in the vehicle. When software fails it should not cause undue damage to users or systems.

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

If enterprise applications were put through the same performance testing rigors as an automobile engine, they would hold up much better than they currently do.

Capacity Planning

Another important aspect of performance testing is capacity planning. This is the art of making sure 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 your data center isn't wasting lots of electricity running software that only utilizes a small portion of the hardware allocated. Hardware efficiency also reduces licensing costs on software and hardware as well as reducing administrative costs.

So how does one go about figuring the most efficient configuration for an application? The first step is to figure out what your customer base looks like and what their habits are. For example:

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

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

Once you have figured out what your peak times are, you should calculate the maximum number of users that you can expect to access your site at any given time during those periods. These are called concurrent users in a system. Your configuration needs to support the maximum number of concurrent users you can expect. 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 you can accommo-date peak load periods without having excess capacity beyond that, you have an efficient configuration.

There are various calculations and spreadsheets that can be used to model the right configuration, but the data needs to be based on the product behavior of the system in question. This data can only be deter-mined by running a small-scale production system in a lab and under-standing the characteristics of the software:

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

These answers will be different for each application and it's critical that some testing is done in a small scale environment to understand these characteristics. If the deployment application server tier will have 32 CPUs, the smaller environment for understanding the application footprint under load can be 4 CPUs. There are various formulas for extrapolating from 4 CPUs to 32. One of the key characteristics looked for during initial testing is if the software scales linearly. This means if the load is doubled from 1 CPU to 2 CPUs, 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. Linear performance is not infinite so one should be careful extrapolating from smaller numbers to very high numbers. At some point, the linearity will tail off, and its part art to figure that out. However, the science of capacity planning can assure a responsive web site that does not consume more resources than it needs.

Performance Monitoring

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

Monitoring a system is also important to application performance and it can provide a lot of useful runtime data. Monitoring is different from profiling a system. It can be done in a way that does not incur any overhead and can be non-intrusive so as not to cause any issues due to the process of monitoring itself. 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. Monitoring also does not require any changes to the code itself. 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 differ but 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. Profiling 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. Another drawback to profiling is that, frequently, attaching a profiler can cause a system to crash under load due to the overhead imposed by the profiler. Monitoring tools, though less informative, often provide a better way to diagnose an application problem under load.

Linux provides a variety of system monitoring tools. As opposed to profilers that need to be purchased separately, these tools are operating system commands and come as part of the Linux operating system. The Linux monitoring commands are text-based so the most useful way to look at the data is to graph it through a spreadsheet tool like Excel. Vi-sual representation of the data can show memory or CPU issues with an application. Below are some ways to gather CPU, memory and disk I/O information about a system. These commands can help identify system level problems, but 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 sys-tem 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 display if 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 any business and IT organization is uptime. Application performance can impact revenues, 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 you ensure the health and the performance of your enterprise applications both before you launch and while in production. Organizations of all sizes should consider these services when application uptime is critical to business productivity.

About Perforic

Perforic provides flexible, high quality and cost-effective enterprise application porting and performance testing services for software product companies and internal development organizations. Our team consists of seasoned veterans with vast experience in performance and porting issues. We have worked with companies of all sizes – from startups to Fortune 500 companies - and have deep expertise in J2EE, .NET and database technologies in highly scalable, three-tier enterprise application environments. The company is based in Cambridge, MA and has 24 x 7 operations with locations and resources in the US and Asia. For more information, please visit http://www.perforic.com/index.html.

PERFORIC
performance is the key

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