Benchmarking the Right Way

With the reduction in budgets and the complexity involved in moving to new technology, more customers are looking to running benchmarks or proof of concepts (POCs) prior to integrating new hardware into the data center. Additionally, these often involve comparing x86 and POWER architectures to determine the best place to put applications. However, performing a benchmark/POC is not as simple as it sounds and there are many things that can go wrong.

By Jaqui Lynch

Common Mistakes

The most common mistake I see is the request to do a benchmark or POC with no real understanding as to what is meant by that. This ends up being a request with no statement of work (SOW) and no properly defined scope. Without a definition of scope, benchmarks tend to drag on for some time that is very expensive in both compute and personnel time.

One of the most important things to define is what constitutes success. What is the target or point of this benchmark or POC? If it is a competitive situation, then what determines which platform wins?

Additionally, proper planning is vital to the success of a benchmark or POC. This involves determining the hardware and software, personnel time, standardized output reporting and how comparisons will be validly made between platforms. Some of the benchmarks require compiling, so a compiler may be needed along with the skills to work with makefiles. Planning also involves correctly choosing benchmarks that are valid on all platforms. As an example, a single-core benchmark is not useful on an environment such as POWER. We will discuss this later but it is important to know what kind of benchmark or POC is to be performed and what the end goal is.

What Is a Benchmark and Why Bother?

A benchmark can be many things. It can be a sizing test or study, a POC, a functionality test, a test of new or current technologies or competitive comparisons. True benchmarking is where the workload is run on the actual platform to be tested. There is also a form of benchmarking called "simulated benchmarking" where tests for a given platform are run on a different platform, as the necessary platform is not available. The discussion below relates to true benchmarking.

There are many reasons people choose to run one of the above types of benchmarks. Sizing studies are not always correct for a number of reasons but primarily because they depend on data input with respect to how the application may be used. Any deviations from that can result in an undersized study. Additionally, they rarely take into account the needs for virtualization and tend to round everything up or down. For that reason, many customers like to run a benchmark so they can determine if the server is correctly sized for the proposed workload.

Additionally, POCs are run to determine if the application, operating systems and/or hardware behave well on the new technology, and if it performs the way that is expected. These can also be pure functionality tests that are looking at whether the application has all the functionality needed and promised.

More recently there has been an upsurge in competitive bake-offs—sometimes comparing server technologies or different server vendors with each other, and sometimes comparing storage systems that are attached to a server. As an example, benchmarks comparing spinning disks, solid state disks (SSDs) and flash systems have become very popular as customers try to determine whether the performance improvements justify the price increases. And last, there are benchmarks comparing different operating systems such as AIX, Linux on POWER and Linux on x86.

Benchmark Options

There are a number of benchmarks out there that are useful to run, and many that are not. When working with POWER systems, it is important to note that POWER servers are designed for throughput, not individual core performance. What this means is that you don't really see the maximum performance until you push the box with multiple things running at the same time. No one runs their server with only one thing running at a time so it makes no sense to run a single core test on a 16 core server. It is much better to run 16 concurrent entities, or even better, to run 64 so you can test the benefits of simultaneous multithreading (SMT). That also allows you to

compare technologies and how they handle the server getting extremely busy.

There are a number of good benchmarks out there for testing performance. For the network, you can look at httperf, iperf or netperf. FTP is not a good test for performance. For general performance, there is the nstress suite for AIX from IBM. This allows you to test memory, CPU and/or disk bandwidth. For Oracle, there is Oracle SwingBench. Tests like Dhrystones are not really valid as they measure integer performance that is designed to test individual core speed, so they don't take advantage of technologies like SMT. Additionally, they have some unusual code in them that is not representative of commercial workloads; and Dhrystones also does not test instruction fetch very well. Finally, there is also the option that your application vendor may have test code you can run—as an example, SAS provides iotest.sh for UNIX to test I/O bandwidth for SAS applications.

Summary of Benchmarks That May Be Useful

httpperf is used to generate web workloads, and to measure generic web server performance. The code for this is old so I would look more to iperf.

iperf is used to test client/server networks and allows you to test maximum TCP and UDP bandwidth. It has options for tuning parameters and characteristics and reports on bandwidth, delay jotter and datagram loss. It also reports on retransmits, average CPU utilization and many other statistics.

netperf is used to test unidirectional throughput and end-to-end latency on the network.

nstress is the one I find most useful. It provides options to hammer CPU, disk and memory as well as testing shared memory, logging and file creation and deletion. It is very easy to use and very flexible. Binaries are provided for AIX, Linux on POWER and Linux on x86 that make it valuable for doing comparisons across technologies and/or operating systems.

Oracle SwingBench is used to stress test Oracle database. It has two kinds of tests—order entry and calling circle. Order entry is the most commonly used and includes logins and lets you test with various numbers of customers and orders.

Specjvm2008 is a free Java benchmark that is used to measure the performance of the Java run-time environment (RTE). It focuses on core Java functionality and stresses memory and CPU with little, to no, network and I/O activity.

There are a number of other options out there, from using multiple dd or cpio commands to purchasing benchmark suites such as TPC-H or TPC-E or any of the various Spec benchmarks.

Moving Forward

Prior to agreeing to a benchmark, there are several steps that will increase the potential of a successful benchmark. These include:

- Document the type of benchmark and all resources required
- Determine the purpose
- Determine what constitutes success and failure
- Document assumptions and expectations
- Ensure there is a clear definition of the scope
- Have a clear list of all tests to be run and how output will be reported
- Ensure the chosen benchmark is a valid test for the hardware it is being run on
- Ensure there is a valid estimate for personnel time, especially of resources such as database administrators (DBAs) are needed
- Check that there are sufficient spare licenses if the software being run requires licenses
- Make sure you understand the costs as a benchmark can take two weeks to setup, four to eight weeks to run and another

When working with POWER systems, it is important to note that POWER servers are designed for throughput, not individual core performance.

week to teardown. Additionally, time needs to be allocated to analysis and reporting of the data.

Reporting

A valid benchmark report needs to include several things. First, is the executive summary that should be at the front. Then there should be a list of what the benchmark was, why it was run and what the assumptions and expectations were. A complete list of hardware, software and setup information should be included as an appendix. For each test run, a list of what changed between tests should be included. It helps to put tables of data at the end in an appendix and it is also critical to choose graphs that make sense. For some things, tables and area graphs are useful, for others radar or spider graphs are useful. Ensure that colors are consistent between graphs so they can be compared quickly.

Summary

While benchmarks have become more common, many issues arise out of a lack of planning. Add to this a lack of understanding of what a benchmark involves and the scope-creep that ensues because of that and you can see how important it is to properly plan for a successful benchmark. Benchmarks are very useful tools if they are used properly and planned for. They can provide incredibly valuable information and help you determine the correct way to move your technology forward. All you have to do is take some control upfront to ensure success.

References

- httpperf
  http://code.google.com/p/httperf/
- iperf
  https://github.com/esnet/iperf
- netperf
  http://www.netperf.org/netperf/NetperfPage.html
- nstress
  https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/nstress
- Specjvm2008
  http://www.spec.org/jvm2008/
- Spec Benchmarks in General
  http://www.spec.org/benchmarks.html
- TPC Benchmarks
  http://www.tpc.org/ ETJ

Jaqui Lynch has more than 36 years of experience working with a variety of projects and operating systems across multiple vendor platforms, including mainframes, UNIX systems, midrange systems and personal workstations. She currently works at Forsythe Technologies as a solutions architect, focusing on POWER Systems with AIX and Linux. Additionally, she regularly presents at IBM and other conferences around the world.
Email: jaqui@circle4.com