The Art of Building a Good Benchmark

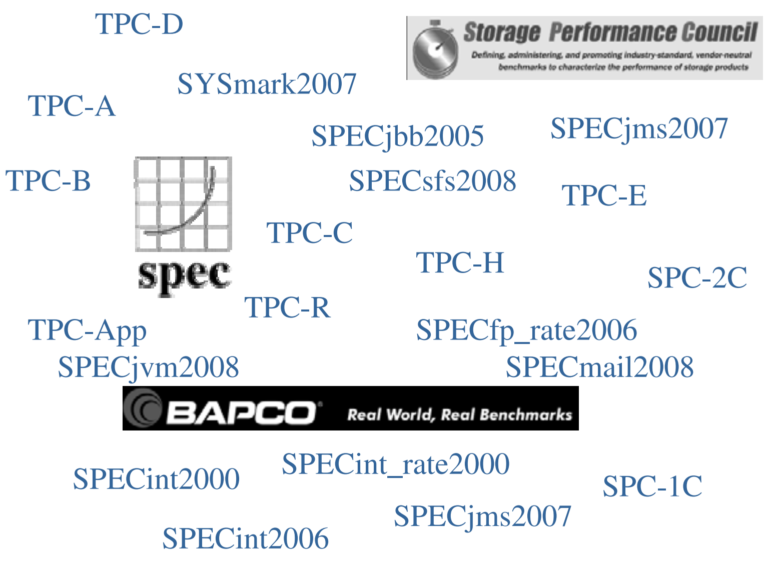
Karl Huppler

IBM Corporation
IBM MS XQK
3605 Highway 52 North
Rochester, MN 55901
huppler@us.ibm.com

Abstract. What makes a good benchmark? This is a question that has been asked often, answered often, altered often. In the past 25 years, the information processing industry has seen the creation of dozens of "industry standard" performance benchmarks – some highly successful, some less so. This paper will explore the overall requirements of a good benchmark, using existing industry standards as examples along the way.

1 Introduction – Building a Good Benchmark

Why so many benchmarks? The cynic would say "They haven't got it right, yet." The pessimist would say "They'll never get it right, but they keep on trying." The realist knows "The computing industry is so vast and changes so rapidly that new benchmarks are constantly required, just to keep up."

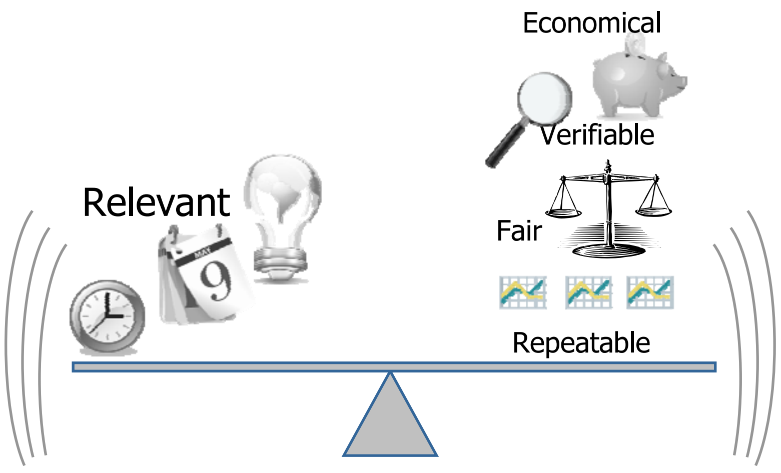


Unfortunately, just because a benchmark is "new" doesn't mean that it measures the "right stuff." The design and implementation of a good performance benchmark is a complex process – often compromising between contrasting goals.

There are five key aspects that all good benchmarks have, to some degree. It is not necessary to be perfect in each of these. In fact, it is impossible to be so. Most good benchmarks have clear strengths in one or two areas, and accommodate the others. The five characteristics are:

- Relevant – A reader of the result believes the benchmark reflects something important
- Repeatable – There is confidence that the benchmark can be run a second time with the same result
- Fair – All systems and/or software being compared can participate equally
- Verifiable – There is confidence that the documented result is real
- Economical – The test sponsors can afford to run the benchmark

Often, in order to satisfy the last four of these items, a benchmark developer must choose to give up on some of the first. This is not all bad, as long as one understands the choices being made. In fact, as we explore each of these items in greater detail, along with discussions of compromise between them, we will also look at the dangers of (believe it or not) doing too good a job in creating a benchmark.



2 Relevant

There are a number of characteristics that can make a benchmark relevant, or irrelevant. Some of them are:

- Meaningful and understandable metric
- Stresses software features in a way that is similar to customer applications
- Exercises hardware systems in a way that is similar to customer applications
- Longevity – leading edge but not bleeding edge
- Broad applicability
- Does not misrepresent itself
- Has a target audience that wants the information

First, the **metric of the benchmark must be understood by the reader** – or at least be perceived to be understood. For example, the metric of SPEC's (Standard Performance Evaluation Corporation) SPECjbb_2005 benchmark is "SPECjbb bops". It isn't difficult for the casual reader to determine that the "ops" is "operations per second", and they might guess that it is "business operations per second", there is no doubt that it is "business operations per second as measured with the SPECjbb benchmark" and one might even infer that the "jbb" stands for "java business benchmark", even though you won't find this phrase in SPEC's documentation for the benchmark. The view that this is a throughput measure of merit for server-side transactional java where bigger is better is quickly understood – and this is a strength of the benchmark.

It parallels another great benchmark, TPC Benchmark C (TPC-C), whose primary performance metric is simply "tpmC" – transactions per minute in Benchmark C – simple, yet elegant: This is a transactional benchmark, measuring throughput, where a larger value is better. That the "C" stands for the third benchmark produced by the Transaction Processing Performance Council (TPC) may be a little obscure, but this can be forgiven for the most successful transactional database benchmark in the industry. A student of the benchmark will find that "tpmC" is really a measure of "New Order Transactions per minute", where the New Order transaction is only one of 5 business transactions in TPC-C, but this is fine, since the ratios of the transaction mix are tightly controlled in the benchmark.

Two more benchmark metric examples: The TPC-H benchmark performance metric is "QphH@xxxGB" where "xxx" is a value that represents the database size that was measured. One can infer that this is also a throughput measure, one of queries per hour in Benchmark H (No, the "H" doesn't represent the 8th benchmark produced by the TPC – it stands for "ad Hoc"). If you study the benchmark, you find that the metric isn't the actual number of queries that are executed per hour, because the metric is actually the geometric mean of the throughput measure

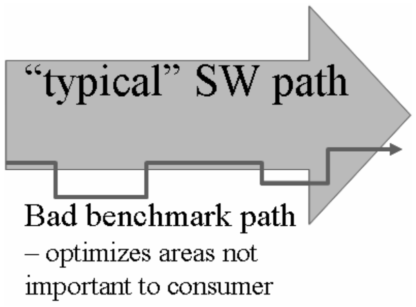
$$\sqrt{\frac{3600*SF}{\sqrt[24]{\prod_{i=1}^{i=22} QI(i,0)*\prod_{j=1}^{j=2} RI(j,0)}} * [(S*22*3600)/Ts *SF]}$$

times the database size and the database size divided by the geometric mean of the individual query times - - Confused? Sure, but for the casual reader, QphH@dbsize means it is a measure of throughput capacity in Benchmark H for queries run against a particular size of database - - - and for all intentional muddying of the formula, here, it truly does relate to that very thing!

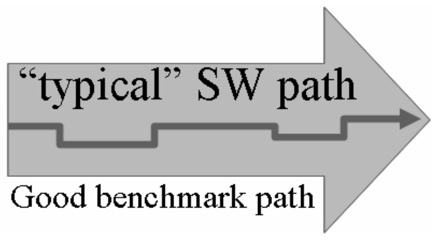
My final example is the SPECcpu2006 suite (SPECfp2006, SPECfp_rate2006, SPECint2006 and SPECint_rate2006). Here, the metric is - - - a number. There are no units, because this metric is essentially a ratio of the ability of the system under test to perform in a suite of intensive processor-oriented operations and functions in comparison to a reference point. To make matters more obscure, there are potentially 8 numbers, for "base" and "peak" measures of each of the two ways to run each of the two independent suites in the benchmark suite. One might ask "How can something that seems to measure something so esoteric be a good benchmark?" The answer is that the SPECcpu suite is so overwhelmingly strong in other aspects that it is far and above the most popular performance benchmark in the world.

The next "relevance" point is the **use of software features in a realistic way**. This can be one of the most challenging aspects of a benchmark, and one that leads to a fairly short life-span for benchmark relevance – because software is constantly evolving. As each software supplier delivers features and functions on an independent schedule, it can also run directly against the "fair" requirement for benchmarks.

The appropriate use of software features is *perhaps the most important requirement* of benchmark development, even though it is also one of the least obvious to the casual observer. It is easy enough to tag a benchmark with terms like "Database", "OLTP", "Decision Support", "Numeric Intensive", "Compute Intensive" and the like. Such terms may make a benchmark appear to be exercising relevant software paths. However, if the benchmark does not use software features in the way that a "typical" customer application will, it can prevent computer providers from delivering optimal solutions to their customers. If a benchmark becomes popular, computer providers will invest skills and money to improve the benchmark results. If the benchmark uses a very limited software path or if the benchmark uses a path that is seldom used by consumers, this investment is made *at the expense* of development that might improve real consumer applications.



On the other hand, when a benchmark exercises features realistically, it can be an absolute boon for consumers, because it gives development organizations the incentive to optimize paths that the consumer wants to take. The hallmark example of this is TPC-C. When it was delivered in 1992, it represented database transaction processing in a way that many, many consumers accomplished that function. At that time, I examined a database that IBM maintained that had performance data from thousands of AS/400 customers (running the operating system that was the predecessor to the IBM i operating system that is one of the options on IBM Power Systems, today.) The assessment showed that the overall path length of a TPC-C New Order was at approximately the 70th percentile of IBM AS/400 customer applications and exercised database and

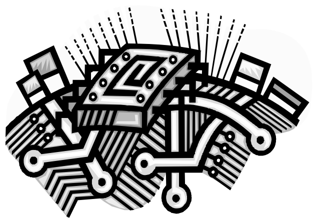


workstation features in a way that was very similar to our customer's OLTP applications. TPC-C has enabled the industry to provide customers with optimizations that are important to their applications, such as improved logging, improved serialization locking, optimal transaction processing paths, optimal transaction control, task/resource affinity, optimal interaction between customer workstations, middle-tier servers and database servers and overall improved path length for many key transaction processing functions. On several occasions, I have observed customer applications that had expanded with customer growth that would have experienced bottlenecks that would slow them down, except that our development team had already removed those bottlenecks to help optimized TPC-C. The relevant paths of the benchmark allowed us to optimize features ahead of when our customers needed them, helping them to expand without stressing the computer systems that they relied on to run their businesses.

There are other examples of similar improvements that benchmarks have provided for consumers: TPC-H provided opportunities to greatly improve parallel processing for large queries. The SPECcpu suite helps to improve compilers, arithmetic operations, string operations, and others. Versions of SPECjbb help with just-in-time (JIT) compilation for Java code. The list goes on.

Next on the "relevance list" is the **use of hardware in a manner that is similar to consumer environments.** As with software, it is important that a benchmark exercise hardware components and subsystems in a meaningful way, but it is even more important that a benchmark does not exercise hardware in ways that are not realistic. For example, a benchmark that does nothing but exercise a floating point accelerator might cause undue investment in that area, at the expense of investments in more general hardware improvements. On the other hand, a benchmark that exercises a mixture of floating point arithmetic, integer arithmetic, cache, memory, string manipulation and vector manipulation might provide a very satisfactory measure of the processor and related components in a system.

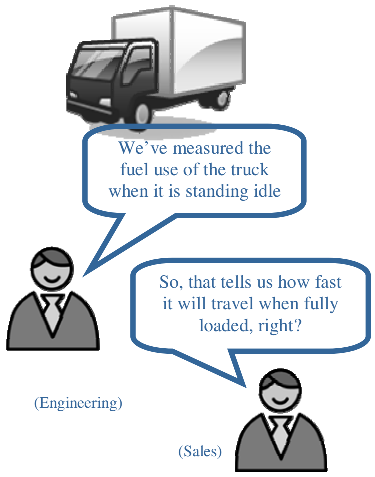
The benchmark of reference is, again, the SPECcpu suite of benchmarks. The members of the Open Systems Group CPU (SPECcpu) committee within SPEC spend a great deal of time and effort making sure that the individual test cases used within the suite stress a variety of relevant hardware and software functions within the processor nest. This is not to say that the benchmark tests that make up the SPECcpu suite are the end-all measure of hardware functions. In fact, these benchmarks do not exercise all hardware functions – by design. This leads, briefly, into a discussion of **appropriate representation.** A strength of the SPECcpu suite is that it says what it does and it does what it says. There is no implication that superior performance in SPECint_rate2006, for example, will translate to superior performance in an environment that requires massive numbers of user-tasks simultaneously competing for processor, memory, cache and I/O resources on the system – but there is a strong indication that it will work well for the portions of the processing that require substantial time to be spent manipulating integers in a variety of ways.



For focus on a broader spectrum of hardware components in an environment with massive numbers of competing tasks that exercise processor, memory, cache, NUMA

characteristics, network I/O and storage I/O, the benchmark of choice has, for years, been TPC-C. One could argue that the sands of time have eroded the software relevance of TPC-C. Applications of today are far more complex than those developed in 1992, when TPC-C was first introduced. However, TPC-C continues to be a premier engineering tool for ensuring that an overall hardware design (and the associated firmware and OS kernels that run on it) is capable of supporting robust multi-user, multi-tasking environments. In this regard, the TPC's two transaction processing benchmarks compliment each other, with TPC-C enabling and encouraging strong affinity and non-uniform allocation of system resources and TPC-E requiring a more uniform allocation of resources across the entire system with less focus on affinity. Both environments are important to consumers, and a combination of the two benchmarks can lead to strong innovation in processor technology and associated hardware components.

Another aspect of **appropriate representation** is taking the steps necessary to ensure that the benchmark is not misused to represent something that was not intended. This can be a challenge, since one of the strengths of a benchmark is to deliver a metric and exercise software and hardware in ways that are meaningful. The natural inclination of a user of the benchmark is to generalize this to assume that the benchmark represents *everything* associated with the environment that it emulates.


We've measured the fuel use of the truck when it is standing idle
So, that tells us how fast it will travel when fully loaded, right?
(Engineering)
(Sales)

I recently had an experience with SPECjbb2005 that highlighted this. The benchmark is "server-side java" and "transactional", with a metric that includes "operations per second". The inclination is to assume that it can be used to represent *all* transactional java environments that run on a server with multiple users – even though the benchmark intentionally does not include network I/O, storage I/O, database or a user interface. In the situation I encountered, someone was attempting to use SPECjbb2005 to examine power management routines when the system is not running at full speed. The way to reduce system utilization with SPECjbb2005 is to run fewer jobs than there are logical processors – which focused some jobs on processors running at nearly 100% utilization while other processors sat idle. Clearly, this is not the way that a real transactional environment would work at moderate system utilization, and the result of the experiment were not what would be expected in a real environment. I should note that SPEC's SPECpower committee addressed this very point when creating SPECpower_ssj2008. This committee used the SPECjbb2005 application as a base for the SPECpower_ssj2008 benchmark, but altered it to more appropriately distribute work across the entire system at lower utilization points.

The next item on my “relevance” list is **longevity.** A benchmark whose usefulness lasts only one year is not a benchmark – it is a workload for a white paper. To a large degree, longevity is accomplished by creating a successful benchmark with other qualities described in this paper. In addition to satisfying these requirements “today”, however, there needs to be a perception that the benchmark will satisfy them “tomorrow”. In order to build a base of comparative performance information, a benchmark needs to be relevant for several years. This means that the software concepts that are exercised must be modern enough that they will still seem current 5 years hence, but not so modern that they will go through rapid change as they mature – The benchmark must be leading edge, but not bleeding edge. It also means that benchmark development must be accomplished in a reasonable time. Innovations in computing technology will likely stay current for five years and may stay current for ten, but if it takes seven years to develop a benchmark, chances are the opportunity for the benchmark to remain relevant over time is very limited.

There is another way to look at **longevity** – that being the longevity of the benchmark suite. Both the SPEC and TPC organizations have recognized that as technology changes, benchmarks may need to change with it. SPEC, in particular, has done an excellent job of initiating discussions for the next version of a benchmark almost as soon as a new version is released. Thus, while the results from the SPECcpu95 suite are not comparable with those of SPECcpu2000 or SPECcpu2006, the concept of what the benchmark is trying to achieve has been retained, maintaining longevity while upgrading the currency of the benchmark suite. The TPC has done this to some degree, too, with changes to the pricing and storage rules for TPC-C and the growth of TPC-D into TPC-R and TPC-H, although one could argue that the next change is overdue.

There are two items left in the “relevant” list: **broad applicability** and **having a strong target audience.** Both seem simple and straightforward, but both create challenges.

Certainly a benchmark application that focuses on the electronic examination of dental x-rays would not be considered to have a broad interest base, and yet if it does not include some of the functions that are important for this, the target audience may not include dentists who are looking to upgrade their information technology. On the other hand, a benchmark that makes use of a variety of imaging techniques could build a target audience that includes dentists, physicians, x-ray specialists, meteorologists, seismologists, geologists, natural resource engineers, crime investigators, and security specialists. The key is to retain sufficient specific use of hardware and software functions and features to stay “real”, while broadening the application to be appropriate for a wide number of uses.

A couple more points on the identification of a **strong target audience:** The target audience must be interested in receiving the information. Suppose the key selection criteria for a computer solution for the groups listed above center around software functionality, hardware stability and customer service, with the assumption that the application design and hardware capacity are capable of handling the required workload. If the target audience doesn't need the information to help with their purchase decisions, the benchmark is of little use.

Finally, I must note that the "target audience" does not need to be "customers". Taking advantage of the many strengths that are listed throughout this paper, the SPECcpu suite has developed a huge audience – in the very people who run the benchmark – engineers, programmers, scientists, academics. Because the benchmark does not require sophisticated software support, it is also an outstanding tool for early processor development. While the benchmarks within the suite are most certainly used to help sell systems, this is almost an afterthought, once the real audience for the benchmark results completes its study.

3 The Other Side of the Coin

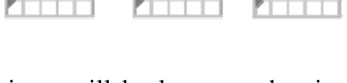
Thus far, I have spent a good deal of time on the need for performance benchmarks to be relevant. Indeed, without relevance, the benchmark will be worthless, at best, and at the worst will cause damage by forcing bad investments. However, just being relevant is insufficient to label a benchmark as "a good benchmark."

An often used phrase is "The best benchmark is the customer's application". This may be true as long as one accepts a target audience of one, but it may not be true, even then. The other four main criteria also enter in. If the benchmark results cannot be **repeated** again and again, the value of the measurement information is in question. Often a customer environment has data that change in a nonuniform way, making it difficult to run the benchmark over and over without doing a full system save/restore operation. If the benchmark cannot be run on different systems with different software packages driving it, it cannot be used to **fairly** evaluate different solutions. If there is no way to **verify** that the results are accurate and the benchmark was run correctly, the confidence in the result is questionable. If the benchmark cannot be run **economically,** without making a massive investment, there is little incentive to run it.

It is well worth discussing these four criteria further, including some examples of how successful benchmarks have implemented them.

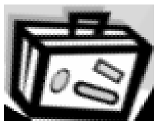
3.1 Repeatable

It sounds so simple – You run the same code on the same system, so you should get the same answer, right? In most cases, this is not so. Database applications grow (or shrink) data and consequently grow and change indices, which means "identical" queries have different paths and process different numbers of rows. Java applications can JIT repeatedly, causing the identical "code" to perform more effectively over time, but they also build up garbage in the java heap that must be cleaned out. Even physical entities are not immune: rotating disk can become less efficient when filled, because seek times will be longer and writes to newly formatted solid state storage are typically faster than over-writes of space that has been previously used to store information.



Benchmark designers must trade some aspects of "reality" to ensure repeatability and consistency – both from run to run and from minute to minute. One of these trade-offs is the creation of a steady-state period within the benchmark. Real applications are hardly steady in the way that they generate work on the system, but a benchmark where results will be compared requires either that the application and associated performance does not change over a period of time (such as with TPC-C, SPECjbb_2005 and SPECweb2005) or that the exact same (or nearly exact same) "dynamic" work flow runs for each iteration of the benchmark (such as with the SPECcpu suite and TPC-H) In TPC-H, updates are made to tables, but in key ranges that do not affect the queries that are the main focus of the benchmark, and in a way that lock contention from the inserts does not affect the read-only queries. In TPC-C, although the History, Order and Order_Line tables continue to grow throughout the benchmark run, empirical data demonstrates that they do not grow so much as to affect the processing of the benchmark application. And, while the TPC-C New_Order table is increased at the same rate that rows are removed by the Delivery transaction, care is taken to reset the database at least after every 12 hours of benchmark execution, because that is the point when the Delivery transaction will begin to process new orders that were created during the benchmark run, rather than the nicely compressed and ordered information that comes in the prebuilt database.

3.2 Fair/Portable



This is another requirement that seems blatantly obvious, but is truly a challenge to accomplish. Portability is less of an issue today that it was two decades ago when the primary benchmark consortias were formed. The use of standard C, C++ and Java languages and the use of standard SQL data access methods allows benchmark applications to be run on a wide variety of platforms. However, being "portable" does not mean that the benchmark is automatically "fair".

Consider the wide variety of database products that exist in today's market – from traditional row-oriented structures, to newer columnar organization, to in-memory environments, to database accelerator appliances – each with specific strengths and potential weaknesses. How, then, can any single application fairly represent the ability of each of these products to perform in a more general environment? The answer is, of course, "It can't." However, benchmark implementers can make compromises that help the situation.

At the extreme, these compromises can take a benchmark to a "lowest common denominator" situation, where they include only tried and true functions that almost all products have had a chance to optimize already. This can be self-defeating, making the benchmark old before it is even introduced. The key is to select functions that are viewed as important in the environment that the benchmark attempts to emulate and to assume that, for products that are weak in some areas, the benchmark can be used to help optimize those products for the general benefit of their customers. The phrase on application currency, "leading edge but not bleeding edge," also applies to the creation of fair benchmarks.

Another aspect of fairness comes, not with the specific benchmark design, but with the designers. If a benchmark is developed and prototyped only on one operating environment, it will naturally tend to be optimized for that environment, at the expense of others. This has been true for some benchmarks from SPEC's Java/ClientServer committee in the past, which focused initially on UNIX-related environments and the TPC's TPC-App benchmark, where development was focused on Windows environments. These benchmarks naturally flowed toward the environments of choice for the benchmark developers, and were not necessarily "fair" to the other environments – even though part of this is because of the choice of the specific vendors involved to simply not participate.

Some compromises can be avoided by not relying on a single benchmark, but instead using multiple benchmarks that may appear to operate in the same space. As previously mentioned, TPC-C is structured to stress features that can take advantage of partitioning and strong affinity between processes and the data they manipulate, whereas TPC-E is structured to reflect applications that are not as easily divided. TPC-E uses standard SQL with portions of the application logic being dictated by the benchmark, much like a business management software package might run, whereas TPC-C allows a broader range of data access methods and complete control over the transaction application code, much like a custom "roll your own" application would use. TPC-H focuses on ad-hoc queries, while its prior sister benchmark, TPC-R, focused on the kind of report generation that can be achieved with pre-defined materialized views that are formed with prior knowledge of the kind of queries that will execute. (TPC-R was retired by the TPC, not because it was poorly implemented, but because it did not generate a sufficient target audience to warrant active continuation of the benchmark.) This is also one of the reasons there is a SPECint2006 and a SPECfp2006 instead of a SPECcpu benchmark.

3.3 Verifiable

A benchmark result is not very useful if there is not a high degree of confidence that it represents the actual performance of the system under test. Simple benchmarks can be self-verifying, providing high confidence as soon as the result is delivered. More complex "system level" benchmarks have greater requirements for verification because there are more things that can change. One possible answer is to take the route that the TPC has taken, requiring benchmark results to be reviewed by a TPC-certified auditor who is very familiar with the benchmark and can identify when an implementation does not follow the benchmark requirements.

SPEC's approach is to simplify benchmarks when possible, to provide automatic verification routines when possible, and to assign final verification to the committee that created the benchmark and is charged with considering revisions in the future.

Both approaches are designed to deliver confidence to the receiver of benchmark results and both have merit. The TPC could learn from SPEC in the creation of self verification routines and the simplification of benchmarks when complexity is not required. As SPEC works toward more complex environments, such as Service Oriented Architecture and Virtualization, they may find that volunteer reviews of results are insufficient without the benefit of the dedicated scrutiny of an independent professional.

3.4 Economical

This is the final item in my list of primary criteria. It is too often overlooked during initial benchmark development, because the initial phases of development are focused on emulating reality to provide the necessary relevance for the benchmark. Indeed, to be relevant, one might expect a benchmark to be realistic; and to be realistic often means to be complex; and to be complex invariably means to be expensive. This is clearly another opportunity for compromise, if one wants to create a successful benchmark.

The term, “economical”, does not mean “cheap”, but rather “worth the investment”. Consider IBM’s leading TPC-C result (6,085,166 tpmC, $2.81USD/tpmC, available December 10, 2008) which employed the use of 11,000 disk drives and 128 middle-tier client systems. Clearly, the return on the investment was worth it. The benchmark was implemented and the result published, after all. On the other hand, it isn’t something one wants to do every week! In fact, as systems become more and more powerful, the cost of supporting equipment in the TPC-C benchmark has been one of the contributing factors in a decline in benchmark publishes.

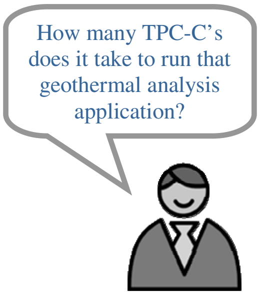
Other benchmarks, like TPC-E, TPC-H, SPECjAppServer2004, SPECweb2005 and some SPEC and TPC benchmarks that are currently under development require robust system configurations that will require investments to run them. However, as with TPC-C, the existence of storage, memory and networking components is key to the business model for these benchmarks, so the trade-off must be the degree to which the business model is satisfied.

In contrast, SPECjbb2005 and SPECfp2006/SPECint2006 enjoy large numbers of benchmark publishes – in part because it is not necessary to establish a massive data center to support them. College students can run these benchmarks on their laptops. They might not want to play too many video games while they wait for SPECfp2006 to complete, but the point is that the benchmarks are very affordable. Both benchmarks make conscious trade-off decisions – They select only a slice of the computing industry’s “total reality”, in return for the appeal of being inexpensive to run, easy to run and easy to verify. As long as they are not used out of the context of their intent, they also meet the requirements for relevance, fairness and repeatability.

4 You Don’t Want All Items Satisfied

Can a benchmark be too perfect? I think so. When TPC-C was introduced in 1992, it satisfied a hunger for a meaningful, robust benchmark that was representative of the kind of database transaction processing that existed in the industry. It had (and still has) a business model that was easily understood. It used software and hardware in a representative way. It was (and is) verifiable. It was (and is) repeatable. At the time, it was relatively economical (The first benchmark results topped out at 33.81 tpmC and 54.14 tpmC, requiring somewhat fewer resources than the results of today.) The target audience was - - - Everyone! Many companies do different things with their computing technology, but ALL businesses must do some kind of database transaction processing to run their business. TPC-C grew to be the premier benchmark of the industry. Marketing teams and customers asked for results in TPC-C first and considered other benchmarks as an afterthought.

TPC-C became such a force in performance benchmarks that it was extraordinarily difficult to change or introduce new, "competing" benchmarks. It became an almost generic measure of computer power, regardless of whether a target application was similar to the TPC-C business model or not. The TPC had several development efforts that would have built on the strengths of TPC-C, while upgrading the characteristics of the benchmark to keep pace with the times. Of these, the newest TPC benchmark, TPC-E, was the only successful one, and although the rate of publishes of TPC-E has now exceeded those of TPC-C, one could argue that they continue to be slowed by the continued strength of the TPC-C benchmark.


How many TPC-C's does it take to run that geothermal analysis application?

In contrast, while SPEC benchmarks were far from obscure, these benchmarks have not been viewed under the brightest of spotlights that was, for a time, reserved for TPC-C, and the engineers who created them have enjoyed the freedom to maintain currency by reviewing and revising them.

5 In Summary

What can we learn from all of this? The first point is that benchmark developers must keep these five primary criteria in mind from the beginning of the development process. Benchmarks must have some component of relevance, repeatability, fairness, verifiability and economy. Perhaps more important is the reality that all of these should not (and likely cannot) be totally satisfied. It is more important to understand the compromises made to enable one strength over another than it is to satisfy every possible criterion.

It is equally important to ensure that the consumers of benchmark information understand the strengths and limitations of each benchmark. It may be better to spend 2 years developing a benchmark that stresses a single subsystem than it is to spend 6 years developing a total system benchmark, but not if the subsystem benchmark is used to represent the "total system."

The industry continues to move rapidly, which implies that new benchmarks are needed and old ones should be considered for retirement. There will likely be some mainstays – Linpack, for one, TPC-C for another, but there is also a need for new tools to evaluate and optimize the features and functions that are growing in importance in today's environment.

Finally, we need to learn from each other. The TPC has an outstanding reputation for building robust, full system benchmarks. As SPEC moves in that direction (particularly with their efforts in virtualization), they could learn a few things from the TPC. SPEC has an outstanding reputation for "rapid" (still measured in years) development and enhancement of benchmarks, and for making conscious compromises I recommend to make benchmarks more manageable in scope and therefore more readily accepted by those who are interested in using them to measure computer systems. The TPC could well learn from this example.