Chapter 1
Benchmarking Basics

"One accurate measurement is worth a thousand expert opinions."
—*Grace Hopper (1906–1992), US Navy Rear Admiral*

"From a user's perspective, the best benchmark is the user's own application program."
—*Kaivalya M. Dixit (1942–2004), Former SPEC President*

This chapter provides a definition of the term "benchmark" followed by definitions of the major system quality attributes that are typically subject of benchmarking. After that, a classification of the different types of benchmarks is provided, followed by an overview of strategies for performance benchmarking. Finally, the quality criteria for good benchmarks are discussed in detail, and the chapter is wrapped up by a discussion of application scenarios for benchmarks.

1.1 Definition of Benchmark

The term benchmark was originally used to refer to "a mark on a workbench used to compare the lengths of pieces so as to determine whether one was longer or shorter than desired."[1] In computer science, a benchmark refers to "a test, or set of tests, designed to compare the performance of one computer system against the performance of others."[1] Performance, in this context, is typically understood as the amount of useful work accomplished by a system compared to the time and resources used. Better performance means more work accomplished in shorter time and/or using less resources. Depending on the context, high performance may involve one or more of the following: high responsiveness when using the system, high processing rate, low amount of resources used, or high availability of the system's

[1] SPEC Glossary: https://www.spec.org/spec/glossary

services. While systems benchmarking has traditionally been focused on evaluating performance in this classical sense (amount of work done vs. time and resources spent), in recent years, the scope of benchmarking has been extended to cover other properties beyond classical performance aspects (see Section 1.2). Examples of such properties include system reliability, security, or energy efficiency. Modern benchmarks can thus be seen as evaluating performance in a broader sense, that is, "the manner in which or the efficiency with which something reacts or fulfills its intended purpose."[2]

In line with this development, we use the following definition of the term benchmark in this book:

Definition 1.1 (Benchmark) A benchmark is a tool coupled with a methodology for the evaluation and comparison of systems or components with respect to specific characteristics, such as performance, reliability, or security.

We refer to the entity (i.e., system or component) that is subject of evaluation as *System Under Test (SUT)*. This definition is a variation of the definition provided by Vieira et al. (2012). A more narrow interpretation of this definition was formulated by Kistowski et al. (2015), where the competitive aspects of benchmarks are stressed (i.e., "a standard tool for the competitive evaluation and comparison of competing systems"), reflecting the fact that competitive system evaluation is the primary purpose of standardized benchmarks as developed by the Standard Performance Evaluation Corporation (SPEC) and the Transaction Processing Performance Council (TPC). To distinguish from tools for non-competitive system evaluation and comparison, such tools are often referred to as *rating tools* or *research benchmarks*. Rating tools are primarily intended as a common method of evaluation for research purposes, regulatory programs, or as part of a system improvement and development approach. Rating tools can also be standardized and should generally follow the same design and quality criteria as standard benchmarks. SPEC's Server Efficiency Rating Tool (SERT), for example, has been designed and developed using a similar process as the SPECpower_ssj2008 benchmark. The term *research benchmark* is used mostly by SPEC's Research Group[3] to refer to standard scenarios and workloads that can be used for in-depth quantitative analysis and evaluation of existing products as well as early prototypes and research results.

Each benchmark is characterized by three key aspects: *metrics*, *workloads*, and *measurement methodology*. The metrics determine what values should be derived based on measurements to produce the benchmark results. The workloads determine under which usage scenarios and conditions (e.g., executed programs, induced system load, injected failures / security attacks) measurements should be performed to derive the metrics. Finally, the measurement methodology defines the end-to-end process to execute the benchmark, collect measurements, and produce the benchmark results.

[2] Random House Webster's Unabridged Dictionary

[3] SPEC Research Group: https://research.spec.org

1.2 System Quality Attributes

As discussed above, systems benchmarking has evolved to cover properties beyond classical performance aspects, such as system reliability, security, or energy efficiency. According to the ISO/IEC 25010:2011 standard, *system quality* can be described in terms of the attributes shown in Figure 1.1. We distinguish between external and internal quality attributes. *External quality* attributes describe the view of the system users, for example, performance, reliability, and usability. *Internal quality* attributes describe the view of the system developers, typically reflected in the attribute *maintainability*, which captures the degree of effectiveness and efficiency with which the system can be modified.

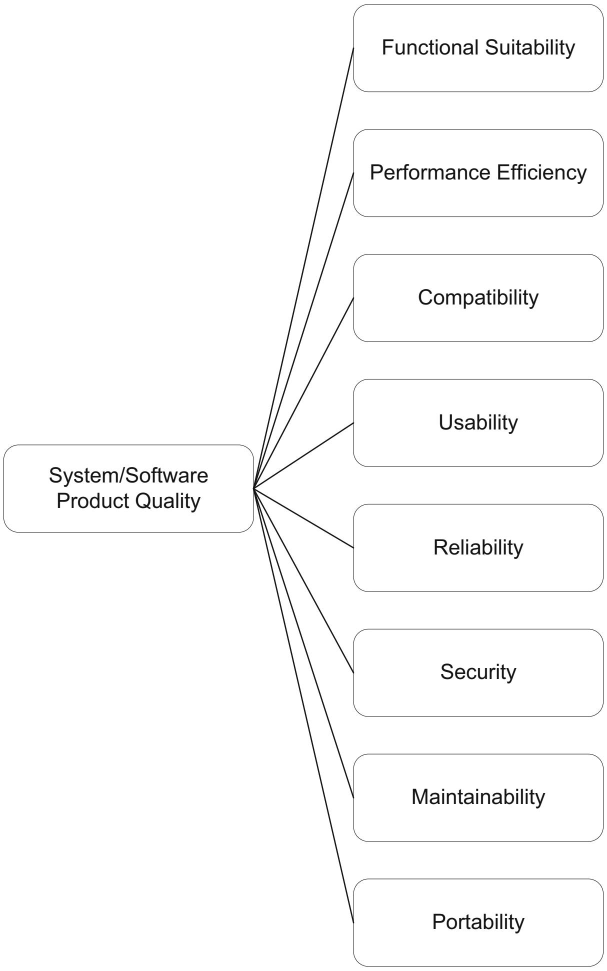

Fig. 1.1: System quality attributes according to ISO/IEC 25010:2011

In the following, we provide a brief overview of the major quality attributes targeted for evaluation by modern benchmarks.

Performance As discussed in Section 1.1, performance in its classical sense captures the amount of useful work accomplished by a system compared to the time and resources used. Typical performance metrics, which will be introduced more formally and discussed in detail in Chapter 3, include response time, throughput, and utilization. Very briefly, *response time* is the time it takes a system to react to a request providing a respective response; *throughput* captures the rate at which requests are processed by a system measured in number of completed requests (operations) per unit of time; and *utilization* is the fraction of time in which a resource (e.g., processor, network link, storage device) is used (i.e., is busy processing requests).[4]

Scalability Scalability is the ability to continue to meet performance requirements as the demand for services increases and resources are added (Smith and Williams, 2001).

Elasticity Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time, the available resources match the current demand as closely as possible (Herbst et al., 2013).

Energy Efficiency Energy efficiency is the ratio of performance over power consumption. Alternatively, energy efficiency can be defined as a ratio of work performed and energy expended for this work.

Availability Availability is the readiness for correct service (Avizienis et al., 2004). In practice, the availability of a system is characterized by the fraction of time that the system is up and available to its users (Menascé et al., 2004), that is, the probability that the system is up at a randomly chosen point in time. The two main reasons for unavailability are system failures and overload conditions.

Reliability Reliability is the continuity of correct service (Avizienis et al., 2004). In practice, the reliability of a system is characterized by the probability that the system functions properly over a specified period of time (Trivedi, 2016).

Security Security is a composite of the attributes of *confidentiality*, *integrity*, and *availability* (Avizienis et al., 2004). Confidentiality is the protection of data against its release to unauthorized parties. Integrity is the protection of data or services against modifications by unauthorized parties. Finally, availability, in the context of security, is the protection of services such that they are ready to be used when needed. Enforcing security typically requires encrypting data, which in many cases may have a significant performance overhead.

Dependability The notion of dependability and its terminology have been established by the International Federation for Information Processing (IFIP) Working Group 10.4, which defines dependability as "the trustworthiness of a computing system that allows reliance to be justifiably placed on the service it delivers." Dependability is an integrative concept that includes the following attributes (Laprie, 1995):

[4] We use the term *request* in a general sense meaning any unit of work executed by a system that has a distinct start and end time, for example, a request sent through a browser to open a web page, a database transaction, a network operation like transferring a data packet, or a batch job executed by a mainframe system.

availability (readiness for correct service), *reliability* (continuity of correct service), *safety* (absence of catastrophic consequences on the users and the environment), *confidentiality* (absence of unauthorized disclosure of information), *integrity* (absence of improper system alterations), and *maintainability* (ability to undergo modifications and repairs).

Resilience Resilience encompasses all attributes of the quality of "working well in a changing world that includes faults, failures, errors, and attacks" (Vieira et al., 2012). Resilience benchmarking merges concepts from performance, dependability, and security benchmarking. In practice, resilience benchmarking faces challenges related to the integration of these three concepts and to the adaptive characteristics of the system under test.

1.3 Types of Benchmarks

Computer benchmarks typically fall into three general categories: *specification-based*, *kit-based*, and *hybrid*. Furthermore, benchmarks can be classified into *synthetic benchmarks*, *microbenchmarks*, *kernel benchmarks*, and *application benchmarks*.

Specification-based benchmarks describe functions that must be realized, required input parameters, and expected outcomes. The implementation to achieve the specification is left to the individual running the benchmark. Kit-based benchmarks provide the implementation as a required part of official benchmark execution. Any functional differences between products that are allowed to be used for implementing the benchmark must be resolved ahead of time. The individual running the benchmark is typically not allowed to alter the execution path of the benchmark.

Specification-based benchmarks begin with a definition of a business problem and a set of specific requirements to be addressed by the benchmark. The key criteria for this definition are the relevance topics discussed in Section 1.5 and novelty. Such benchmarks have the advantage of allowing innovative software to address the business problem of the benchmark by proving that the specified requirements are satisfied by the new implementation (Huppler and Johnson, 2014). On the other hand, they require substantial development prior to running the benchmark and may have challenges proving that all requirements of the benchmark are met.

Kit-based benchmarks may appear to restrict some innovative approaches to a business problem, but have the advantage of providing near "load and go" implementations that greatly reduce the cost and time required to run the benchmark. For kit-based benchmarks, the "specification" is used as a design guide for the creation of the kit. For specification-based benchmarks, the "specification" is presented as a set of rules to be followed by a third party who will implement and run the benchmark. This allows for substantial flexibility in how the benchmark's business problem will be resolved—a principal advantage of specification-based benchmarks.

A hybrid of the specification-based and kit-based approaches may be necessary if the majority of the benchmark can be provided in a kit, but there is a desire to allow some functions to be implemented at the discretion of the individual running the benchmark. While both specification-based and kit-based approaches have been successful in the past, current trends favor kit-based development.

The differences between synthetic benchmarks, microbenchmarks, kernel benchmarks, and application benchmarks are discussed next based on the classification by Lilja (2000). To evaluate the performance of a system with respect to a given characteristic, the system must execute some sort of program, as defined by the benchmark *workload*. Since the user is ultimately interested in how the system will perform when executing *his* application, the best program to run is obviously the user's application itself. Unfortunately, in practice this is usually infeasible, as a significant amount of time and effort may be required to port the application to the SUT. Also, one may be interested in comparing different systems to determine which one is most suitable for developing a new application. Since, in such a case, the application will not exist yet, it cannot be used as a workload for benchmarking. Given these observations, one is often forced to rely on making measurements while executing a different program than the user's application. Depending on the type of benchmark program used, benchmarks can be classified into synthetic benchmarks, microbenchmarks, kernel benchmarks, or application benchmarks.

Synthetic benchmarks are artificial programs that are constructed to try to mimic the characteristics of a given class of applications. They normally do this by executing mixes of operations carefully chosen to elicit certain system behavior and/or to match the relative mix of operations observed in the considered class of applications. The hope is that if the induced system behavior and/or the executed operation mixes are similar, the performance observed when running the benchmark would be similar to the performance obtained when executing an actual application from the respective class. The major issue with synthetic benchmarks is that they do not capture the impact of interactions between operations caused by specific execution orderings. Furthermore, such benchmarks often fail to capture the memory-referencing patterns of real applications. Thus, in many cases, synthetic benchmarks fail to provide representative workloads exhibiting similar performance to real applications from the respective domain. However, given their flexibility, synthetic benchmarks are useful for tailored system analysis allowing one to measure the limits of a system under different conditions.

Microbenchmarks are small programs used to test some specific part of a system (e.g., a small piece of code, a system operation, or a component) independent of the rest of the system. For example, a microbenchmark may be used to evaluate the performance of the floating-point execution unit of a processor, the memory management unit, or the I/O subsystem. Microbenchmarks are often used to determine the maximum performance that would be possible if the overall system performance were limited by the performance of the respective part of the system under evaluation.

Kernel benchmarks (also called program kernels) are small programs that capture the central or essential portion of a specific type of application. A kernel benchmark typically executes the portion of program code that consumes a large fraction of the total execution time of the considered application. The hope is that since this code is executed frequently, it captures the most important operations performed by the actual application. Given their compact size, kernel benchmarks have the advantage that they are normally easy to port to different systems. On the downside, they may fail to capture important influencing factors since they may ignore important system components (e.g., operating system or middleware) and may not stress the memory hierarchy in a realistic manner.

Application benchmarks are complete real application programs designed to be representative of a particular class of applications. In contrast to kernel or synthetic benchmarks, such benchmarks do real work (i.e., they execute real, meaningful tasks) and can thus more accurately characterize how real applications are likely to behave. However, application benchmarks often use artificially small input datasets in order to reduce the time and effort required to run the benchmarks. In many cases, this limits their ability to capture the memory and I/O requirements of real applications. Nonetheless, despite this limitation, application benchmarks are usually the most effective benchmarks in capturing the behavior of real applications.

1.4 Performance Benchmarking Strategies

As discussed in the beginning of this chapter, in classical performance benchmarking, a benchmark is defined as a test, or set of tests, designed to compare the performance of one system against the performance of others. The term *performance* in this context is understood as the amount of useful work accomplished by a system compared to the time and resources used. Better performance means more work accomplished in shorter time and/or using less resources.

In classical performance benchmarking, three different benchmarking strategies can be distinguished (Lilja, 2000): (1) *fixed-work benchmarks*, which measure the time required to perform a fixed amount of work; (2) *fixed-time benchmarks*, which measure the amount of work performed in a fixed period of time; and (3) *variable-work and variable-time benchmarks*, which vary both the amount of work and the execution time.

1.4.1 Fixed-Work Benchmarks

Let W_i be the "amount of work" done by System i in a measurement interval T_i. The amount of work done can be seen as an event count, where each event represents a completion of a unit of work.

The *system speed* (execution rate) is defined as $R_i = W_i/T_i$. Assuming that we run a fixed-work benchmark on two systems, that is, $W_1 = W_2 = W$, it follows that the speedup of the second system relative to the first is given by[5]

[5] We formally introduce and discuss the metric *speedup* in more detail in Chapter 3 (Section 3.3).

$$S = \frac{R_2}{R_1} = \frac{\frac{W}{T_2}}{\frac{W}{T_1}} = \frac{T_1}{T_2}. \tag{1.1}$$

Thus, the time T_i a system needs to execute W units of work can be used to compare the performance of systems. The performance of a system typically depends on the performance of multiple system components (e.g., CPU, main memory, I/O subsystem) that are used during operation. The main issue with fixed-work benchmarks is that they introduce an intrinsic performance bottleneck limiting how much the performance can be improved by improving only a single component of the system.

To illustrate this, assume that a system is optimized by improving the performance of its most important performance-influencing component (e.g., its CPU). The system's execution time can be broken down into two parts: time spent processing at the component under optimization and time spent processing at other components unaffected by the optimization. Assume that the performance of the optimized component is boosted by a factor of q. Let T be the time the system needs to execute W units of work before the optimization is applied, and let T' be the time it needs to execute the same workload after the optimization is applied. Let α be the fraction of time in which the optimized component is executing. $(1-\alpha)$ will then correspond to the fraction of time spent on components and activities unaffected by the optimization. Figure 1.2 illustrates the impact of the optimization on the overall benchmark execution time (Lilja, 2000).

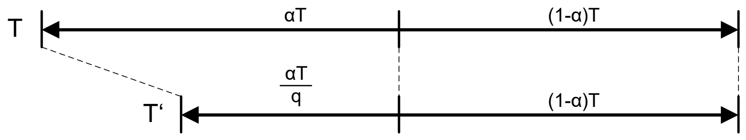

Fig. 1.2: Impact of optimizing a single system component

We observe that the overall system speedup that can be achieved through the described kind of optimization is limited:

$$S = \frac{T}{T'} = \frac{T}{\frac{\alpha T}{q} + (1-\alpha)T} = \frac{1}{1-\alpha\left(1-\frac{1}{q}\right)}, \tag{1.2}$$

$$\lim_{q\to\infty} S = \lim_{q\to\infty} \frac{1}{1-\alpha\left(1-\frac{1}{q}\right)} = \frac{1}{1-\alpha}. \tag{1.3}$$

Equation (1.3) is known as *Amdahl's law*. It introduces an upper bound on the overall performance improvement that can be achieved by improving the performance of a single component of a system. Given this upper bound, fixed-work benchmarks are not very popular in the industry since they have an intrinsic performance bottleneck limiting how much performance improvement can be achieved by optimizing a given system component.

1.4.2 Fixed-Time Benchmarks

To address the described issue, fixed-time benchmarks do not fix the amount of work W_i and instead measure the amount of work that can be processed in a fixed period of time. The amount of work that a system manages to process in the available time is used to compare the performance of systems.

This idea was first implemented in the SLALOM benchmark (Gustafson, Rover, et al., 1991), which runs an algorithm for calculating radiosity.[6] The performance metric is the accuracy of the answer that can be computed in 1 min. The faster a system executes, the more accurate the result obtained in 1 min would be. The advantage of this approach is that the problem being solved is automatically scaled to the capabilities of the system under test.

Fixed-time benchmarks address the described scalability issue of fixed-work benchmarks. Whatever part of the execution is optimized, it would lead to saving some time, which can then be used for processing further units of work improving the benchmark result.

To show this, we consider the same scenario as above but with a fixed execution time. Let T_f be the fixed time period for which the benchmark is executed. Given that the optimized part of the execution is running q times faster, if we imagine running the same amount of work without the optimization, the respective execution time would be q times longer. This is illustrated in Figure 1.3, sometimes referred to as the scaled version of Amdahl's law (Lilja, 2000). The benchmark execution time is compared against a hypothetical scenario (T_h) where the same amount of work is processed without the optimization.

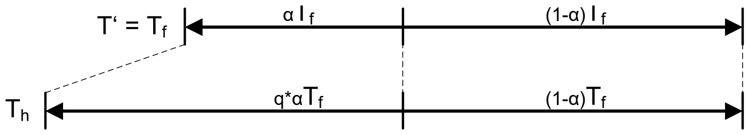

Fig. 1.3: Scaled version of Amdahl's law

We observe that unlike the case for fixed-work benchmarks, this time there is no upper bound on the speedup that can be achieved:

$$S = \frac{T_h}{T_f} = \frac{\alpha q T_f + (1-\alpha)T_f}{T_f} = \alpha q + (1-\alpha), \tag{1.4}$$

$$\lim_{q\to\infty} S = \lim_{q\to\infty} [\alpha q + (1-\alpha)] = \infty. \tag{1.5}$$

[6] Radiosity is a global illumination algorithm used in 3D computer graphics rendering.

1.4.3 Variable-Work and Variable-Time Benchmarks

Fixed-work benchmarks are most intuitive in terms of our expectation of how improvements in system performance should impact the execution time of an application. Fixed-time benchmarks, on the other hand, have the advantage that the problem solved is automatically scaled to reflect the capabilities of the system under test. Finally, there are also variable-work and variable-time benchmarks, which fix neither the available time T nor the amount of work W (Lilja, 2000). The metric is defined as a function of T and W.

An example of a benchmark that follows this strategy is the HINT benchmark (Gustafson and Snell, 1995). It defines a mathematical problem to be solved and uses the quality of the provided solution as a basis for evaluation. The assumption is that the solution quality can be improved continually if additional time for computations is available. The ratio of the provided solution quality to the time used to achieve this quality is used as a performance metric.

The specific problem considered in the HINT benchmark is to find rational upper and lower bounds for the integral

$$\int_0^1 \frac{1-x}{1+x} dx. \tag{1.6}$$

The classical interval subdivision technique is used to find the two bounds by dividing the interval $[0, 1]$ into subintervals and counting the number of squares in the area completely below the curve and those in the area including the curve. The solution quality is then defined as $1/(u-l)$, where u and l are the computed upper and lower bounds, respectively. The metric quality improvements per second (QUIPS) is then computed by dividing the quality of the solution by the execution time in seconds. By fixing neither the execution time nor the amount of work to be processed, maximum flexibility is provided to evaluate the performance of systems with different behavior and performance bottlenecks.

1.5 Benchmark Quality Criteria

Benchmark designers must balance several, often conflicting, criteria in order to be successful. Several factors must be taken into consideration, and trade-offs between various design choices will influence the strengths and weaknesses of a benchmark. Since no single benchmark can be strong in all areas, there will always be a need for multiple workloads and benchmarks (Skadron et al., 2003).

It is important to understand the characteristics of a benchmark and determine whether or not it is applicable for a particular situation. When developing a new benchmark, the goals should be defined so that choices between competing design criteria can be made in accordance with those goals to achieve the desired balance.

Several researchers and industry participants have listed various desirable characteristics of benchmarks (Gustafson and Snell, 1995; Henning, 2000; Huppler, 2009; Kistowski et al., 2015; Sim et al., 2003; Skadron et al., 2003). The contents of the lists vary based on the perspective of the authors and their choice of terminology and grouping of characteristics, but most of the concepts are similar. The key characteristics can be organized in the following five groups:

1. **Relevance:** how closely the benchmark behavior correlates to behaviors that are of interest to users,
2. **Reproducibility:** producing consistent results when the benchmark is run with the same test configuration,
3. **Fairness:** allowing different test configurations to compete on their merits without artificial limitations,
4. **Verifiability:** providing confidence that a benchmark result is accurate, and
5. **Usability:** avoiding roadblocks for users to run the benchmark in their test environments.

All benchmarks are subject to these same criteria, but each category includes additional issues that are specific to the individual benchmark, depending on the benchmark's goals. In the following, the individual criteria are discussed in more detail based on Kistowski et al. (2015).

1.5.1 Relevance

"Relevance" is perhaps the most important characteristic of a benchmark. Even if the workload was perfect in every other regard, it will be of minimal use if it does not provide relevant information to its consumers. Yet relevance is also a characteristic of how the benchmark results are applied; benchmarks may be highly relevant for some scenarios and of minimal relevance for others. For the consumer of benchmark results, an assessment of a benchmark's relevance must be made in the context of the planned use of the benchmark results. For the benchmark designer, relevance means determining the intended usage scenarios of the benchmark and then designing the benchmark to be relevant for those usage scenarios (SPECpower Committee, 2014).

A general assessment of the relevance of a benchmark or workload involves two dimensions: (1) the breadth of its applicability and (2) the degree to which the workload is relevant in a given area. For example, an XML parsing benchmark may be highly relevant as a measure of XML parsing performance, somewhat relevant as a measure of enterprise server application performance, and not at all relevant for graphics performance of 3D games. Conversely, a suite of CPU benchmarks such as SPEC CPU2017 may be moderately relevant for a wide range of computing environments. Benchmarks that are designed to be highly relevant in a specific area tend to have narrow applicability, while benchmarks that attempt to be applicable to a broader spectrum of uses tend to be less meaningful for any particular scenario (Huppler, 2009).

Scalability is an important aspect of relevance, particularly for server benchmarks. Most relevant benchmarks are multiprocess and/or multithreaded in order to be able to take advantage of the full resources of the server (Skadron et al., 2003). Achieving scalability in any application is difficult; for a benchmark, the challenges are often even greater because the benchmark is expected to run on a wide variety of systems with significant differences in available resources. Benchmark designers must also strike a careful balance between avoiding artificial limits to scaling and behaving like real applications, which often have scalability issues of their own.

1.5.2 Reproducibility

Reproducibility is the capability of the benchmark to produce the same results consistently for a particular test environment. It includes both run-to-run consistency and the ability for another tester to independently reproduce the results in another but identical environment.

Ideally, a benchmark result should be a function of the hardware and software configuration, so that the benchmark is a measure of the performance of that environment; if this were the case, the benchmark would have perfect consistency. In reality, the complexity inherent in a modern computer system introduces significant variability in the performance of an application. This variability is introduced by several factors, including things such as the timing of thread scheduling, dynamic compilation, physical disk layout, network contention, and user interaction with the system during the run (Huppler, 2009). Energy-efficiency benchmarks often have additional sources of variability due to power management technologies dynamically making changes to system performance and temperature changes affecting power consumption.

Benchmarks can address this run-to-run variability by running workloads for long enough periods of time (or executing them multiple times successively) in order to include representative samples of these variable behaviors. Some benchmarks require submission of several runs with scores that are near each other as evidence of consistency. Benchmarks also tend to run in steady state[7], unlike more typical applications, which have variations in load due to factors such as the usage patterns of users.

The ability to reproduce results in another test environment is largely tied to the ability to build an identical environment. Industry-standard benchmarks require results submissions to include a description of the test environment, typically including both hardware and software components as well as configuration options. Similarly, published research that includes benchmark results generally adds a description of the test environment that produced those results. However, in both of these cases,

[7] Generally, a system or a process is considered to be in a steady state if the variables that define its behavior are unchanging in time (Gagniuc, 2017). In the context of benchmarking, typically, the variables that define the workload executed on the system under test are considered.

the description may not provide enough detail for an independent tester to be able to assemble an identical environment.

Hardware must be described in sufficient detail for another person to obtain identical hardware. Software versions must be stated so that it is possible to use the same versions when reproducing the result. Tuning and configuration options must be documented for firmware, operating system, and application software so that the same options can be used when rerunning the test. Unfortunately, much of this information cannot be automatically obtained in a reliable way, so it is largely up to the tester to provide complete and accurate details.

TPC benchmarks require a certified auditor to audit results and ensure compliance with reporting requirements. SPEC uses a combination of automatic validation and committee reviews to establish compliance.

1.5.3 Fairness

Fairness ensures that systems can compete on their merits without artificial constraints. Because benchmarks always have some degree of artificiality, it is often necessary to place some constraints on test environments in order to avoid unrealistic configurations that take advantage of the simplistic nature of the benchmark.

Benchmark development requires compromises among multiple design goals; a benchmark developed by a consensus of experts is generally perceived as being more fair than a benchmark designed by a single company. While “design by committee” may not be the most efficient way to develop a benchmark, it does require that compromises are made in such a way that multiple interested parties are able to agree that the final benchmark is fair. As a result, benchmarks produced by organizations such as SPEC and TPC (both of which comprise members from industry as well as academic institutions and other interested parties) are generally regarded as fair measures of performance.

Benchmarks require a variety of hardware and software components to provide an environment suitable for running them. It is often necessary to place restrictions on what components may be used. Careful attention must be placed on these restrictions to ensure that the benchmark remains fair. Some restrictions must be made for technical reasons. For example, a benchmark implemented in Java requires a Java Virtual Machine (JVM) and an operating system and hardware that supports it. A benchmark that performs heavy disk I/O may effectively require a certain number of disks to achieve acceptable I/O rates, which would therefore limit the benchmark to hardware capable of supporting that number of disks.

Benchmark *run rules*, which specify the requirements that have to be fulfilled in order to produce a compliant benchmark result, often require the used hardware and software to meet some level of support or availability. While this restricts what components may be used, it is actually intended to promote fairness. Because benchmarks are by nature simplified applications, it is often possible to use simplified software to run them; this software may be quite fast because it lacks features that may be required by real applications. For example, enterprise servers typically require certain security features in their software which may not be directly exercised by benchmark applications; software that omits these features may run faster than software that includes them, but this simplified software may not be usable for the customer base to which the benchmark is targeted. Rules regarding software support can be a particular challenge when using open source software, which is often supported primarily by the developer community rather than commercial support mechanisms.

Both of these situations require a careful balance. Placing too many or inappropriate limits on the configuration may disallow results that are relevant to some legitimate situations. Placing too few restrictions can pollute the pool of published results and, in some cases, reduce the number of relevant results because vendors cannot compete with the "inappropriate" submissions.

Portability is an important aspect of fairness. Some benchmarks, such as TPC-C, provide only a specification and not an implementation of the benchmark, allowing vendors to implement the specification using whatever technologies are appropriate for their environment (as long as the implementation is compliant with the specification and other run rules). Other benchmarks, such as those from SPEC, provide an implementation that must be used. Achieving portability with benchmarks written in Java is relatively simple; for C and C++, it can be more difficult (Henning, 2000).

If the benchmark allows code to be recompiled, rules must be defined to state what compilation flags are allowed. SPEC CPU2017 defines *base results* (with minimal allowed compilation flags) and *peak results* (allowing the tester to use whatever compilation flags they would like). Similarly, Java benchmarks may put limits on what JVM command line options may be used.

In some cases, multiple implementations may be required to support different technologies. In this case, it may be necessary (as with SPECweb2009) for results with different implementations to be assigned to different categories so they cannot be compared with each other.

Benchmark run rules often include stipulations on how results may be used. These requirements are intended to promote fairness when results are published and compared, and they often include provisions that require certain basic information to be included any time that results are given. For example, SPECpower_ssj2008 requires that if a comparison is made for the power consumption of two systems at the 50% target load level, the performance of each system at the 50% load level as well as the overall `ssj_ops/watt` value must also be stated.

SPEC has perhaps the most comprehensive fair use policy, which further illustrates the types of fair use issues that benchmarks should consider when creating their run rules.[8]

[8] SPEC fair use rules: https://www.spec.org/fairuse.html

1.5.4 Verifiability

Within the industry, benchmarks are typically run by vendors who have a vested interest in the results. In academia, results are subjected to peer review and interesting results will be repeated and built upon by other researchers. In both cases, it is important that benchmark results are verifiable so that the results can be deemed trustworthy.

Good benchmarks perform some amount of self-validation to ensure that the workload is running as expected and that run rules are being followed. While a workload might include configuration options intended to allow researchers to change the behavior of the workload, standard benchmarks typically limit these options to some set of compliant values, which can be verified at run time. Benchmarks may also perform some functional verification that the output of the test is correct; these tests could detect some cases where optimizations (e.g., experimental compiler options) are producing incorrect results.

Verifiability is simplified when configuration options are controlled by the benchmark or when these details can be read by the benchmark. In this case, the benchmark can include the details with the results. Configuration details that must be documented by the user are less trustworthy since they could have been entered incorrectly.

One way to improve verifiability is to include more details in the results than are strictly necessary to produce the benchmark's metrics. Inconsistencies in this data could raise questions about the validity of the data. For example, a benchmark with a throughput metric might include response time information in addition to the transaction counts and elapsed time.

1.5.5 Usability

Most users of benchmarks are technically sophisticated, making ease of use less of a concern than it is for more consumer-focused applications; however, there are several reasons why ease of use is important. One of the most important ease of use features for a benchmark is self-validation. This was already discussed in terms of making the benchmark verifiable. Self-validating workloads give the tester confidence that the workload is running properly.

Another aspect of ease of use is being able to build practical configurations for running the benchmark. For example, one of the top TPC-C results has a system under test with over 100 distinct servers, over 700 disk drives, 11,000 SSD modules (with a total capacity of 1.76 petabytes), and a system cost of over $30 million. Of the 18 non-historical accepted TPC-C results published between January 1, 2010 and August 24, 2013, the median total system cost was $776.627. Such configurations are not economical for most potential users (Huppler, 2009).

Accurate descriptions of the system hardware and software configuration are critical for reproducibility but can be a challenge due to the complexity of these descriptions. Benchmarks can improve ease of use by providing tools to automatically extract the relevant information required for generating the descriptions.

1.6 Application Scenarios for Benchmarks

While benchmarking has traditionally been focused on competitive system evaluation and comparison, over the past couple of decades the scope of benchmarking has evolved to cover many other application scenarios. In the following, a brief overview of the main application scenarios for modern benchmarks is given. Benchmarks are considered in a broad sense including tools for non-competitive system evaluation (i.e., rating tools or research benchmarks).

As discussed in Section 1.1, systems benchmarking has traditionally been focused on evaluating performance in a classical sense (amount of work done vs. time and resources spent); however, in recent years, the scope of systems benchmarking has been extended to cover other properties beyond classical performance aspects, such as system reliability, security, or energy efficiency.

Generally, benchmarks are used to evaluate systems with respect to certain quality attributes of interest. In Section 1.2, we provided an overview of the major system quality attributes that may be subject of evaluation using benchmarks.

We refer to the entity (e.g., end user or vendor) that employs a benchmark to evaluate a system under test (SUT) as *benchmarker*. The SUT could be a hardware or software product (or service), or it could be an end-to-end application comprising multiple hardware and software components. Figure 1.4 shows the different scenarios of what the benchmarker could be with respect to his goals and intentions.

In the first scenario, the benchmarker is a customer interested to buy the SUT who uses the benchmark to compare and rank competing products offered by different vendors or to determine the size and capacity of a specific system to be purchased.

In the second scenario, the benchmarker is a vendor who sells the SUT to customers and is using the benchmark to showcase its quality for marketing purposes. Industry-standard benchmarks and standardization bodies provide means to evaluate and showcase a product's quality on a level playing field. The benchmarker might also be interested to receive an official certificate issued by a certification agency, attesting a given quality level (e.g., energy-efficiency standard).

In the next scenario, the benchmarker is in the process of developing, deploying, or operating the SUT. In this scenario, one might be using the benchmark for stress or regression testing, for system optimization or performance tuning, for system sizing and capacity planning, or for validating a given hardware and software configuration. In this scenario, the benchmark may also be used as a blueprint demonstrating programming best practices and design patterns as a guidance for the development of a new system architecture.

Finally, the benchmarker may be a researcher using the benchmark as a representative application to evaluate novel system architectures or approaches to system development and operation.

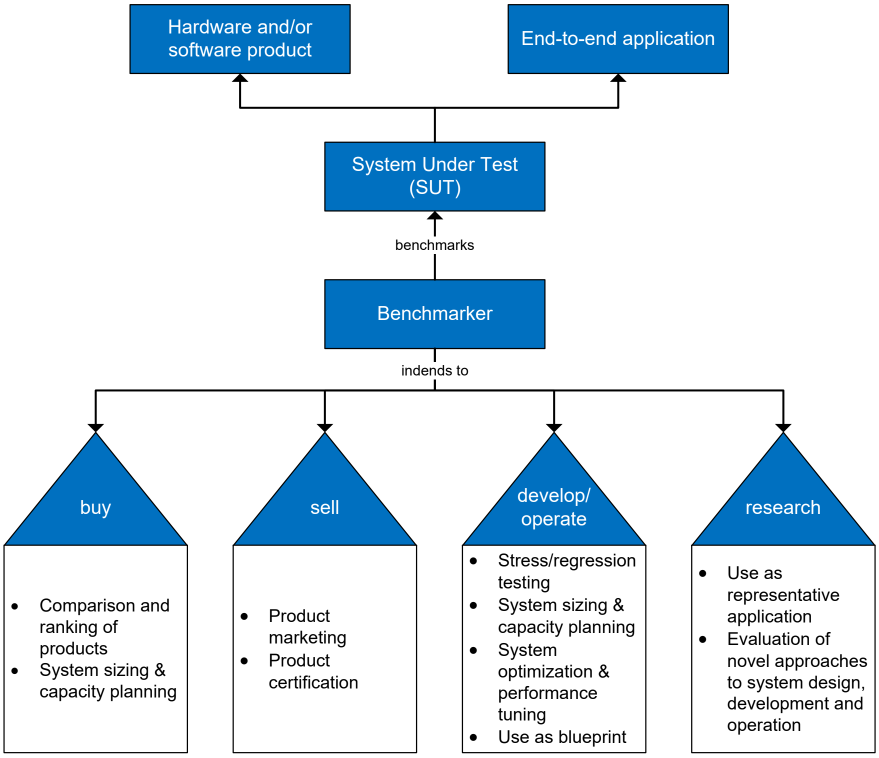
Hardware and/or software product
End-to-end application
System Under Test (SUT)
benchmarks
Benchmarker
indends to
buy
sell
develop/ operate
research
• Comparison and ranking of products
• System sizing & capacity planning
• Product marketing
• Product certification
• Stress/regression testing
• System sizing & capacity planning
• System optimization & performance tuning
• Use as blueprint
• Use as representative application
• Evaluation of novel approaches to system design, development and operation

Fig. 1.4: Application scenarios for benchmarks

1.7 Concluding Remarks

In this chapter, we introduced the fundamental concepts and terminology used in systems benchmarking. We provided a definition of the term “benchmark” followed by definitions of the major system quality attributes that are typically subject of benchmarking. We distinguished between external and internal system quality attributes. After that, we presented a classification of the different types of benchmarks, followed by an overview of strategies for performance benchmarking. Finally, the quality criteria for good benchmarks, such as relevance, reproducibility, fairness, verifiability, and usability, were discussed in detail. While benchmarking has traditionally been focused on competitive system evaluation and comparison, over the past couple of decades, the scope of benchmarking has evolved to cover many other application scenarios. In the final section of this chapter, we provided an overview of the application scenarios for benchmarks discussing both their use in industry and their use in the research and academic community.

References

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). “Basic Concepts and Taxonomy of Dependable and Secure Computing”. *IEEE Transactions on Dependable and Secure Computing*, 1(1). IEEE Computer Society: Los Alamitos, CA, USA, pp. 11–33 (cited on p. 6).

Gagniuc, P. A. (2017). *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons: Hoboken, New Jersey, USA (cited on p. 14).

Gustafson, J., Rover, D., Elbert, S., and Carter, M. (1991). “The Design of a Scalable, Fixed-Time Computer Benchmark”. *Journal of Parallel and Distributed Computing*, 12(4). Academic Press, Inc.: Orlando, FL, USA, pp. 388–401 (cited on p. 11).

Gustafson, J. and Snell, Q. (1995). “HINT: A New Way To Measure Computer Performance”. In: *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*. (Wailea, Hawaii, USA). Vol. 2. IEEE, pp. 392–401 (cited on pp. 12, 13).

Henning, J. L. (2000). “SPEC CPU2000: Measuring CPU Performance in the New Millennium”. *Computer*, 33(7). IEEE: New Jersey, USA, pp. 28–35 (cited on pp. 13, 16).

Herbst, N. R., Kounev, S., and Reussner, R. (2013). “Elasticity in Cloud Computing: What it is, and What it is Not”. In: *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013)*. (San Jose, CA, USA). USENIX, pp. 23–27 (cited on p. 6).

Huppler, K. (2009). “The Art of Building a Good Benchmark”. In: *Performance Evaluation and Benchmarking—First TPC Technology Conference (TPCTC 2009), Revised Selected Papers*. Ed. by R. O. Nambiar and M. Poess. Vol. 5895. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 18–30 (cited on pp. 13, 14, 17).

Huppler, K. and Johnson, D. (2014). “TPC Express - A New Path for TPC Benchmarks”. In: *Performance Characterization and Benchmarking—5th TPC Technology Conference (TPCTC 2013), Revised Selected Papers*. Ed. by R. O. Nambiar and M. Poess. Vol. 8391. Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, pp. 48–60 (cited on p. 7).

Kistowski, J. von, Arnold, J. A., Huppler, K., Lange, K.-D., Henning, J. L., and Cao, P. (2015). “How to Build a Benchmark”. In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)*. (Austin, TX, USA). ACM: New York, NY, USA, pp. 333–336 (cited on pp. 4, 13).

Laprie, J.-C. (1995). “Dependable Computing: Concepts, Limits, Challenges”. In: *Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS 1995)*. (Pasadena, CA, USA). IEEE Computer Society: Washington, DC, USA, pp. 42–54 (cited on p. 6).

Lilja, D. J. (2000). *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press: Cambridge, UK (cited on pp. 8–12).

Menascé, D. A., Almeida, V. A., and Dowdy, L. W. (2004). *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall: Upper Saddle River, NJ, USA (cited on p. 6).

Sim, S. E., Easterbrook, S., and Holt, R. C. (2003). "Using Benchmarking to Advance Research: A Challenge to Software Engineering". In: *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*. (Portland, Oregon). IEEE Computer Society: Washington, DC, USA, pp. 74–83 (cited on p. 13).

Skadron, K., Martonosi, M., August, D. I., Hill, M. D., Lilja, D. J., and Pai, V. S. (2003). "Challenges in Computer Architecture Evaluation". *Computer*, 36(8). IEEE Computer Society: Los Alamitos, CA, USA, pp. 30–36 (cited on pp. 12–14).

Smith, C. U. and Williams, L. G. (2001). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley Professional Computing Series. Addison-Wesley: Boston, USA (cited on p. 6).

SPECpower Committee (2014). *Power and Performance Benchmark Methodology V2.2*. Gainesville, VA, USA: Standard Performance Evaluation Corporation (SPEC) (cited on p. 13).

Trivedi, K. S. (2016). *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Second Edition. John Wiley and Sons Ltd.: Hoboken, New Jersey (cited on p. 6).

Vieira, M., Madeira, H., Sachs, K., and Kounev, S. (2012). "Resilience Benchmarking". In: *Resilience Assessment and Evaluation of Computing Systems*. Ed. by K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel. Springer-Verlag: Berlin, Heidelberg, pp. 283–301 (cited on pp. 4, 7).