# Benchmarking robust performance

Goetz Graefe

Hewlett-Packard Laboratories

## Abstract

It’s great if a system enables great performance and scalability, but it’s even better if the system promises at least good performance and scalability upon every invocation. In other words, benchmarks are required not only for performance and scalability but also for robustness of performance and scalability.

## 1 Introduction

Benchmarks are employed in a wide variety of engineering and business functions to quantify performance and a variety of aspects of performance, e.g., latency and bandwidth, scalability and efficiency, reliability and availability, and more. One aspect is particularly relevant for planning and decision making yet it is particularly difficult to measure, namely robustness of performance. In fact, it even is difficult to define robustness: for example, “absence of surprises” is merely an unsatisfactory attempt to describe, let alone define, robustness of performance.

Nonetheless, “absence of surprises” captures the essence fairly well. A user or administrator will perceive a system and its performance as robust if it reliably performs in the same or similar ways. For example, an automobile that is slower than the fastest sports car may lack in performance, but reliable and repeatable operation every day, in every weather and other circumstance, makes it a “robust performer.”

On the other hand, robust performance is more than predictable performance. Just like a car that fails to start every time it rains is predictable but not a robust performer, predictable but “unreasonable” malfunction contradict robustness.

## 2 Robust performance effects

What, then, is robust performance in a cloud application? Moreover, how can it be measured by means of benchmarks? And even more concretely, how can successive releases or “software builds” be compared in order to ensure not just increasing performance of specific test cases but increasing robustness of performance? Regression testing of functionality is widely employed, even regression testing of performance and scalability for specific test cases deemed relevant or typical; robustness of performance, however, remains elusive.

Causes of surprising performance can be discrete or continuous. For example, if two jobs frequently start at the same time, and workload management (based on memory contention) executes these two jobs sometimes concurrently and sometimes serially one after another, then the second job will have run-times that are unpredictable, surprising, and just the opposite of robust. If workload management is regarded as an unchangeable “black box,” this lack of robustness has a discrete cause, namely the scheduling decision of the workload management module. The remedy might be to force one decision or the other, i.e., either concurrent or serial execution, and thus to eliminate workload management and its policies.

If, on the other hand, the decisions are driven by a comparison of the available memory with a predefined threshold, then the true cause is pretty much continuous. Thus, the appropriate remedy might be modify the jobs, their execution mechanisms, and their treatment by workload management. After the jobs are, if necessary, recoded such that they can work reasonably efficiently with any amount of memory, then workload management might turn into resource management, i.e., memory allocation policies replace admission control policies.

In general, continuous causes, effects, mechanisms, and policies lend themselves more readily to robust execution. By definition, graceful degradation is a continuous effect. Good (or good enough) performance plus graceful degradation are the hallmarks of robust performance.

For scalability, a new term is needed, because scalability is the opposite of degradation. "Graceful growth" may be an appropriate term.

## 3 Benchmarking

Let us come back to measuring, benchmarking, and regression-testing robustness in performance and scalability: The first focus should be on verifying that continuous causes have continuous effects, i.e., effects do not become discrete. A second focus should be on the ranges of causing and caused metrics, i.e., the range of values for which effects degrade gracefully does not diminish or deteriorate. Third, graceful degradation must not become less graceful, i.e., detrimental effects should become smaller from build to build, not larger. Finally, graceful growth should become better from build to build and from release to release.

These guidelines for robust performance and scalability  suggest specific approaches to and actions in benchmarking. Specifically, after performance-limiting causes are identified, their effects can varied and graceful degradation ensured with appropriate test series and success metrics. Moreover, for remaining discrete causes and effects, competing systems, their policies, and their mechanisms can be analyzed to verify that discrete, non-graceful behavior is the industry norm. Finally, where discrete effects remain today's standard but are onerous or even expensive in real-world usage, research might increase the number of discrete steps in causes and in effects, or it might even turn a discrete effect into a continuous, graceful one.

## 4 Summary

In summary, robustness of performance remains a research challenge, from a conceptual definition to concrete metrics and measuring by means of benchmarks. Graceful degradation and graceful growth can be implemented, measured, and if necessary researched in more cases than is realized with today's standard focus on a few, even if typical, test cases.

## Biography

Goetz Graefe is a HP Fellow researching database systems at Hewlett-Packard Laboratories, specifically robust query processing, query execution algorithms, automatic indexing, and transactional storage. Prior work includes academic research into database query optimization and product architecture for a major commercial database server. His best-known papers are surveys on query execution and on B-tree indexing.