Performance Metrics

Unlike performance goals, performance metrics are not connected to a specific scenario or environment. A performance metric is a measurable numeric quantity that reflects the application's behavior. You can measure a performance metric on any hardware and in any environment, regardless of the number of active users, requests, or sessions. During the development lifecycle, you choose the metrics to measure and derive from them specific performance goals.

Some applications have performance metrics specific to their domain. We do not attempt to identify these metrics here. Instead, we list, in Table **1-2** , performance metrics often important to many applications, as well as the chapter in which optimization of these metrics is discussed. (The CPU utilization and execution time metrics are so important that they are discussed in *every* chapter of this book.)

Table 1-2. *List of Performance Metrics (Partial)*

Performance Metric	Units of Measurement	Specific Chapter(s) in This Book
CPU Utilization	Percent	All Chapters
Physical/Virtual Memory Usage	Bytes, kilobytes, megabytes, gigabytes	Chapter 4 – Garbage Collection Chapter 5 – Collections and Generics
Cache Misses	Count, rate/second	Chapter 5 – Collections and Generics Chapter 6 – Concurrency and Parallelism
Page Faults	Count, rate/second	–
Database Access Counts/Timing	Count, rate/second, milliseconds	–
Allocations	Number of bytes, number of objects, rate/second	Chapter 3 – Type Internals Chapter 4 – Garbage Collection
Execution Time	Milliseconds	All Chapters
Network Operations	Count, rate/second	Chapter 7 – Networking, I/O, and Serialization Chapter 11 – Web Applications
Disk Operations	Count, rate/second	Chapter 7 – Networking, I/O, and Serialization
Response Time	Milliseconds	Chapter 11 – Web Applications
Garbage Collections	Count, rate/second, duration (milliseconds), % of total time	Chapter 4 – Garbage Collection
Exceptions Thrown	Count, rate/second	Chapter 10 – Performance Patterns
Startup Time	Milliseconds	Chapter 10 – Performance Patterns
Contentions	Count, rate/second	Chapter 6 – Concurrency and Parallelism

Some metrics are more relevant to certain application types than others. For example, database access times are not a metric you can measure on a client system. Some common combinations of performance metrics and application types include:

- For client applications, you might focus on startup time, memory usage, and CPU utilization.
- For server applications hosting the system's algorithms, you usually focus on CPU utilization, cache misses, contentions, allocations, and garbage collections.
- For Web applications, you typically measure memory usage, database access, network and disk operations, and response time.

A final observation about performance metrics is that the level at which they are measured can often be changed without significantly changing the metric's meaning. For example, allocations and execution time can be measured at the system level, at the single process level, or even for individual methods and lines. Execution time within a specific method can be a more actionable performance metric than overall CPU utilization or execution time at the process level. Unfortunately, increasing the granularity of measurements often incurs a performance overhead, as we illustrate in the next chapter by discussing various profiling tools.

PERFORMANCE IN THE SOFTWARE DEVELOPMENT LIFECYCLE

Where do you fit performance in the software development lifecycle? This innocent question carries the baggage of having to *retrofit* performance into an existing process. Although it is possible, a healthier approach is to consider every step of the development lifecycle an opportunity to understand the application's performance better: first, the performance goals and important metrics; next, whether the application meets or exceeds its goals; and finally, whether maintenance, user loads, and requirement changes introduce any regressions.

1. During the requirements gathering phase, start thinking about the performance goals you would like to set.
2. During the architecture phase, refine the performance metrics important for your application and define concrete performance goals.
3. During the development phase, frequently perform exploratory performance testing on prototype code or partially complete features to verify you are well within the system's performance goals.
4. During the testing phase, perform significant load testing and performance testing to validate completely your system's performance goals.
5. During subsequent development and maintenance, perform additional load testing and performance testing with every release (preferably on a daily or weekly basis) to quickly identify any performance regressions introduced into the system.

Taking the time to develop a suite of automatic load tests and performance tests, set up an isolated lab environment in which to run them, and analyze their results carefully to make sure no regressions are introduced is very time-consuming. Nevertheless, the

performance benefits gained from systematically measuring and improving performance and making sure regressions do not creep slowly into the system is worth the initial investment in having a robust performance development process.