Reviewed by Andrew Binstock

December 4, 2020

The JVM is surely one of the most extraordinary pieces of software ever written. The complexity it hides includes an interpreter, two compilers, a performance profiler, and several layers of garbage collection (GC)—all of which work together seamlessly. (When was the last time a Java program didn’t work because of a bug in the JVM?)

Far from being a brittle set of technologies (which pundits repeatedly predict is the inevitable result of great complexity), the JVM is highly configurable with literally hundreds of command-line options and switches. These switches provide performance engineers a gold mine of possibilities to explore in the pursuit of the optimal configuration for a given workload on a given platform.

Because Java is so often deployed on servers, this kind of performance tuning is an essential activity for many organizations—especially enterprises, where thousands of users might be simultaneously accessing a single application. Such enterprises often have performance analysts on staff, and those experts rely on books such as the present volume for guidance. This edition covers Java 11 (with a limited discussion of Java 12) and follows the well-received first edition, which appeared in 2014.

GC is the core focus of much Java performance tuning. And with the advent of new GC models, it is difficult to assess which one works best for a given type of workload. So, as you would expect, author Scott Oaks spends many pages discussing this topic, starting with detailed explanations of how memory is managed in the JVM and where GC fits in. Then he analyzes the various options. The CMS and G1 collectors are examined in great detail. ZGC, Shenandoah, and Epsilon (the no-GC option) are explored in more summary fashion.

The explanations are remarkably clear and are a showcase for Oaks’ deep understanding of JVM internals, which is doubtlessly rooted in his work at Oracle with the Java team (and in similar work earlier at Sun Microsystems).

However, unlike some other treatments of Java performance, Oaks visits many additional areas that affect the runtime profile of a given app.
Oaks begins with a review of benchmarking Java and a tour of the principal Java performance tools. He then dives into the JVM internals, specifically, the memory model, GC, and the JIT compiler. He follows this discussion with two sections on best practices: one on managing the heap (45 dense pages) and the other on managing native memory.

The book then covers threading and synchronization issues, where Oaks digs into, among other things, the effects of hyperthreading, the benefits and costs of explicitly setting the number of threads the JVM should use, and so on. His explanations are frequently backed with benchmarks that convincingly support his recommendations and every so often reveal unexpected performance anomalies, which Oaks carefully explains.

These chapters lead up to server-specific topics, which include detailed analyses of serving HTML and retrieving data from REST servers. Even JSON handling (parsing, marshaling, and so on) are examined with an eye to optimal performance.

These topics are followed by a lengthy chapter on database access. SQL and NoSQL databases are examined in detail, with benchmarks and recommendations presented for both JDBC and JPA. Here again, Oaks shows his command of the subject by explaining how to correctly understand benchmarks, which are easy to get wrong with databases—leading to ineffective attempts at optimization.

For those who are not working on server apps, Oaks includes a 60-page section that focuses just on Java SE. He provides insights into optimal string handling, buffered I/O, collections, and several lesser topics. Like the previous chapters, the section illustrates that there are gotchas that can appear unexpectedly by making a suboptimal choice in otherwise perfectly normal code.

This insightful experience is a consistent thread throughout the book, which is one reason I highly recommend the book to all intermediate-to-advanced developers. Even in the unlikely event you don’t pick up some new best practices, the view you will gain into what is happening as your code executes is worth the time you spend reading. Fortunately, Oaks’ style is clear and articulate, so the reading is pleasant and easy to follow.

**Comparison to other performance books**

There are quite a few books on Java performance. The most significant early treatment was the iconic *Java Performance* by Charlie Hunt and Binu John. It first came out in 2011 and had a partial update in 2016 from Hunt and Monica Beckwith, which I reviewed in the September/October 2016 issue of *Java Magazine*. While both the book and the update were very good in their time, the JVM has changed significantly in the intervening years. These advances are surely what led Oaks to
publish his second edition, which replaces his first edition from 2014.

A more recent and very relevant book on performance is *Optimizing Java* by Benjamin Evans, James Gough, and Chris Newland, which appeared in 2018 and which I reviewed [here](#). Like the present volume, it is excellent. The Evans title takes a somewhat more technical dive into the Java language and the JVM. For example, it includes a lot more Java code and frequently analyzes it at the level of the generated bytecode. In this sense, the Evans book is more oriented to developers than the current volume.

I highly recommend getting both volumes if Java performance is an important concern for you. Where the books overlap, they reinforce each other by bringing different perspectives to common topics—of which there are many.


---

**Andrew Binstock**

Andrew Binstock (@platypusguy) was formerly the editor in chief of *Java Magazine*. Previously, he was the editor of *Dr. Dobb's Journal*. He co-founded the company behind the open-source iText PDF library, which was acquired in 2015. His book on algorithm implementation in C went through 16 printings before joining the long tail. Previously, he was the editor in chief of *UNIX Review* and, earlier, the founding editor of the *C Gazette*. He lives in Silicon Valley with his wife. When not coding or editing, he studies piano.