Master's Thesis

Runtime Analysis of High-Level Java Synchronization Mechanisms on the VM Level

Student: *David Gnedt*
Course No.: *921*
Student No.: *1055171*

Institute for System Software

Dipl.-Ing. Peter Hofer

Phone.: +43 732 2468-4369

Fax: +43 732 2468-4345

peter.hofer@jku.at

Linz, 10.11.2014

To take advantage of now common multi-core and multi-processor hardware, software must perform tasks in parallel in multiple threads. This requires proper synchronization between the threads to ensure safe access to shared data. Java provides built-in basic support for synchronization through *object monitors* and the *synchronized* keyword in the language. In addition, Java 1.5 introduced the *java.util.concurrent* package, which contains solutions to many common synchronization problems. Both the implementation of object monitors in the Java virtual machine and of the classes of the java.util.concurrent package are well-tested and tuned for performance.

Still, when synchronization mechanisms are not used efficiently, it can diminish the benefits of the parallelization. Most notably, frequent *lock contentions* cause threads to wait for each other instead of making progress. In extreme cases, this can result in worse performance than that of a single-threaded implementation. However, the efficiency and scalability of a synchronization mechanism for real-world usage is often difficult to judge during development and tools are needed to detect and comprehend performance problems.

At our laboratory, we are currently researching approaches for detecting performance problems with synchronization in Java applications and their underlying causes. Reproducing performance problems on development or test infrastructure is often difficult, but existing tools have too much overhead for monitoring in production environments. With the integration of our techniques into the Java virtual machine, we strive to achieve overheads that are low enough for production use. The task for this thesis is to extend these techniques to the high-level synchronization mechanisms of the java.util.concurrent package. A generic approach that covers many of those mechanisms is desirable, especially if it can be applied to custom-made synchronization mechanisms created with *ownable synchronizers.* However, readily available data from known synchronization mechanisms should be utilized as well, such as wait queues or the semantics of a read/write lock.

Dipl.-Ing. Peter Hofer