Accelerating Maven by Delaying Dependencies

Jonathan Bell*, Eric Melski†, Gail Kaiser* and Mohan Dattatreya†
*Department of Computer Science, Columbia University, New York, NY 10027, {jbell,kaiser}@cs.columbia.edu
†Electric Cloud, Inc, San Jose, CA 95114, {ericm,mohan}@electric-cloud.com

***Abstract*—Modular build systems (such as Maven) may simplify build maintenance, but significantly reduce opportunities for parallelism where they may be most helpful: when running tests. If tests are contained in each module, and modules contain dependencies on each other, their tests can not run in parallel. This poster will present a technique for achieving significantly greater parallelism in running the tests of Maven-built Java projects, cutting build times in half in our case study.**

I. Extended Abstract

Slow builds are a perpetual nuisance to the software release lifecycle: they reduce the frequency with which builds can occur, delaying build results from getting back to developers and build engineers. In our previous work, we determined that build times of Java projects are often dominated by the execution of tests, and presented two approaches to significantly reduce the time necessary to run entire test suites by cutting time spent isolating tests, and safely parallelizing them [1].

We have discovered a new way to accelerate the testing phase of builds (specifically, Java builds performed with Maven) even further by introducing a new level of parallelism, allowing tests to begin executing before projects are fully built without introducing a risk of build failure or nondeterminism. While there have been attempts to support fully parallel Java builds (e.g., Maven's $-T$ feature), realizing high degrees of parallelism remains challenging due to difficult-to-break dependencies. Maven is a modular build system, allowing projects to consist of multiple sub-modules, each of which may depend on other modules. When a multi-module Maven project is built, each module goes through the entire Maven build lifecycle (compile, test, package, etc.)

We measured the amount of time spent running tests in 10 large open source java projects (building on Amazon's EC2 with m3.medium instances), finding that tests are often distributed among many modules. Table I shows the results of this study: most projects have many modules, and many of those projects have tests.

Project	**Build Time (mins)**	**Testing Time**	**Modules w/ Tests**
titan	380.77	94.99%	13/15
camel	359.57	84.68%	195/271
mule	198.87	92.81%	57/72
spring-data-mongodb	123.17	99.32%	3/3
cdap	110.62	97.15%	19/33
hadoop	108.03	97.78%	27/36
opennms	120.73	76.89%	122/220
ks-final-milestone	124.23	71.08%	17/46
mongo-java-driver	74.92	99.35%	1/1
netty	67.63	92.24%	16/19

TABLE I
Testing statistics for large Maven projects. Shows the build time, percent of build time running tests, and the number of Maven modules of each project that have tests.

We have observed that while one module may depend on the code or other artifacts generated by a previous module, they do not rely on the execution of the tests of a prior module. However, due to Maven's modular nature, it is impossible to specify that a single component of a module (for instance, its tests) should run in parallel with other modules. Therefore, we have built a plugin for test execution in Maven that *delays* the dependency between each module's test phase, allowing tests to execute in parallel to the build of other modules. This way, Maven considers individual modules as fully built (for the purpose of dependency resolution) even if that module's tests haven't finished running yet. Since the dependence is delayed (rather than dropped), Maven still executes the tests for each module, and won't consider the build overall complete until the tests finish executing.

We applied this technique to build a proprietary, internal system that had previously taken 20 minutes to build, even when utilizing Maven's provided parallelism features. After applying our delayed dependency technique, the system took only 8 minutes to build, using the same number of processor cores as Maven's automatic parallelism provided. We are continuing to refine this prototype system and are actively working on applying it to new projects to gain more evidence of its usefulness.

This poster will describe in greater detail the mechanism that we use to delay module dependencies on test execution, as well as further results showing the applicability of this technique.

Acknowledgements

Bell and Kaiser are members of The Programming Systems Laboratory, which is funded in part by NSF CCF-1302269, CCF-1161079, and NIH U54 CA121852. Bell was partially supported by Electric Cloud while completing this work.

References

[1] J. Bell, E. Melski, M. Dattatreya, and G. Kaiser. Vroom: Faster Build Processes for Java. In *IEEE Software Special Issue: Release Engineering*. IEEE Computer Society, March/April 2015. To Appear. Preprint: http://jonbell.net/s2bel.pdf.