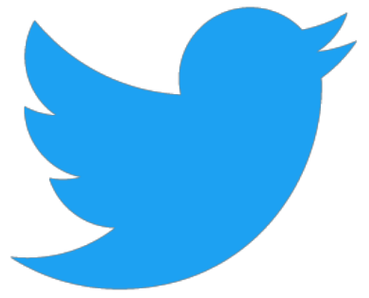
Swapping in Graal native-image to multiply Pants JVM tool performance

(by Danny McClanahan, at Twitter)



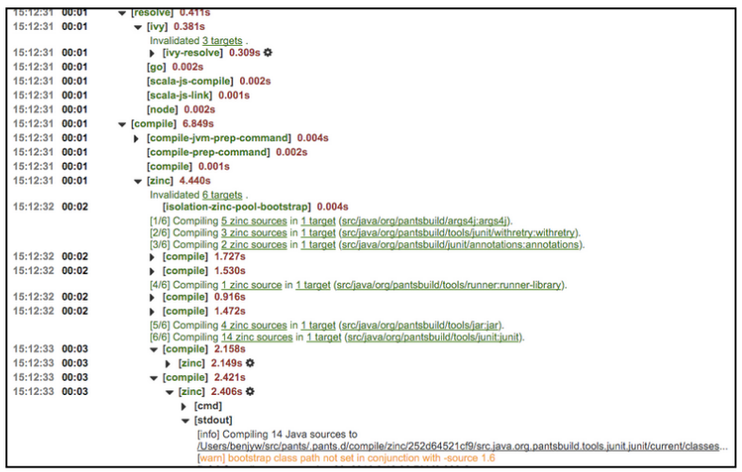
https://www.pantsbuild.org

Pants | Docs | Community | GitHub | Search

Pants: A fast, scalable build system

Pants is a build system designed for codebases that:

- Are large and/or growing rapidly.
- Consist of many subprojects that share a significant amount of code.
- Have complex dependencies on third-party libraries.
- Use a variety of languages, code generators and frameworks.



Pants supports Java, Scala, Python, C/C++, Go, Javascript/Node, Thrift, Protobuf and Android code. Adding support for other languages, frameworks and code generators is straightforward.

https://github.com/pantsbuild/pants

pants usage at Twitter

- pants is a build tool which Twitter contributes to extensively
  - written in python and rust
  - very strong support for jvm code, as we mainly use scala and apache thrift
- focuses a lot on integrating external tools
  - lots of different ways to invoke subprocesses
- at twitter it is used on osx laptops and linux ci machines

mix-and-match jvm tool execution strategies in pants

- we can execute jvm code in a one-shot java subprocess, or a persistent nailgun instance

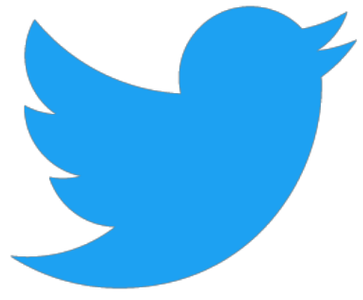
    - nailgun avoids jvm startup and allows the jit to warm up

```
dmcclanahan@tw-mbp-dmcclanahan: ~/tools/pants
X> ./pants -ldebug \
       --jvm-platform-compiler=rsc \
       compile.rsc --execution-strategy=graal --worker-count=1 \
       testprojects/src/scala/org/pantsbuild/testproject/javadepsonscala
```

providing execution strategy options to pants for jvm tools

- nailgun is the default

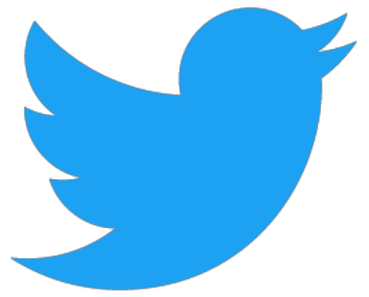
    - one-shot can be more appropriate for *ephemeral environments*



pants can orchestrate scalafmt parallelism

https://github.com/scalameta/scalafmt

- scalafmt is a formatter and linter for scala code
    - we would like to be able to ship this as a lint that runs on every commit
        - scalafmt workload is embarassingly parallel, can be split by input files
            - no dependencies, just some global settings
- some edits to the pants scalafmt task were required to use the *parallelism* described herein



adding native-image as a pants jvm execution strategy

- it was incredibly easy to slot in graal native-image through the SubstrateVM as a pants jvm tool execution backend

@ShaneDelmore suggested this!

  - required *no* changes to the scalafmt tool!

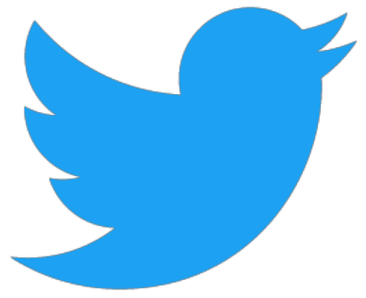
```
cls.register_jvm_tool(
  register,
  'scalafmt',
  classpath=[
    JarDependency(org='com.geirsson',
                  name='scalafmt-cli_2.11',
                  rev='1.5.1'),
```

scalafmt jar dependency declaration

- pants can now turn maven coordinates into a native-image executable of that jvm package automagically

persistent JVM processes can be difficult to manage

- we would like to avoid having nailguns if possible
  - takes up memory and (some) cpu on laptops
  - it is *significantly* more difficult to manage a fleet of CI machines if we have to wait for more processes to start every time we spin up a CI box
- it can be more difficult to profile nailgun executions due to the shared heap



relying on a persistent JIT also means relying on the tool's own performance

- with nailgun, we're limited to the parallelism that the tool itself provides
    - scalafmt has its own threading
- using native-image, we can scale the parallelism arbitrarily high with multiple processes!

scalafmt speedup through multiprocess parallelism

- introduced a `--files-per-process` option to the pants scalafmt task

NOTE: As noted in Q&A, this speedup could be obtained by connecting to a scalafmt nailgun server with multiple threads! The utility here is of not maintaining a persistent nailgun server (with a fixed heap size)!

- observed a 25% speedup when using 150 files/process on the open source finagle project

- a warm nailgun converged to 15 seconds

- 150 files/process for native-image ran in 12 seconds, **every time**

- similar numbers for larger parts of the monorepo

why is multiprocess parallelism faster for scalafmt?

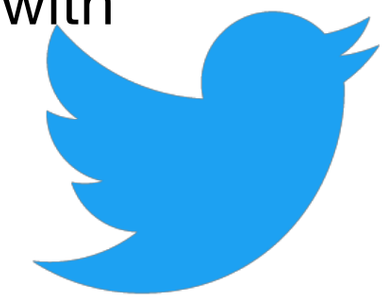
- formatting is embarrassingly parallel and IO-bound
    - pants can scale the parallelism arbitrarily high to saturate the OS with IO
- it's difficult to expect each command line tool to implement its own threading optimally for every scenario
    - pants knows beforehand how many files there are to format
    - pants can bucket the input files so processes receive approximately similar workloads

generalizing the approach for distributed compiles

- noted earlier that it can be difficult to maintain persistent nailguns on a CI fleet
    - twitter is working on remoting jvm compiles
        - bazel already does this so google devs don't compile locally
- we use nailguns with threading in pants to compile scala right now
    - we do this on individual ci machines, but if we try to distribute compiles, we need to maintain a nailgun on each machine, and this makes it much harder to provision

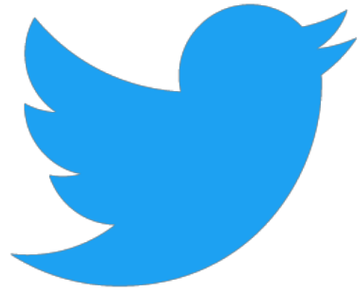
generalizing the approach for distributed compiles (with rsc!)

- **rsc** (**https://github.com/twitter/rsc**) is an experimental scala compiler focused on compilation speed (no codegen yet) -- incredibly well written
    - for very interesting reasons, we can use rsc to completely parallelize the actual compilations
- using native-image with rsc/scalac/javac means we don't have to maintain nailguns on a fleet of ci machines
    - we can now remote individual compile processes, because starting up a jvm doesn't take any time at all
    - we don't have to focus all of our compiles onto specific machines with warm nailguns -- can distribute across a fleet!



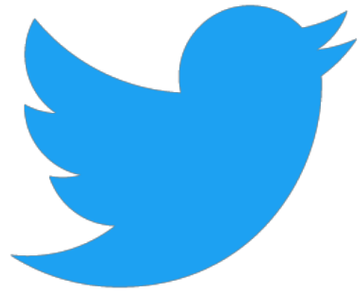
how can this be used with other tools?

- tools we have tried:
  - scalafmt: no changes needed to make a native-image
  - rsc: no changes
  - scalac: some substitutions/reflection configs
  - javac: will try this later this week
  - zinc: OOMed, bytecode parsing errors (gone after 1.0.0-rc12!), then ran into svm internal errors



how can other build tools copy this approach?

- they would first need to copy pants's fantastic jvm tool support
    - pants makes it extremely easy to drop in nailgun vs native-image
- we can run apples-to-apples comparisons because the process executions are the exact same
    - pants tasks or the jvm tools they wrap don't need to make any modifications to allow this!



takeaways

- native-image removes the difficulty of managing a persistent jvm process

- native-image gives the build tool parent process control over parallelism instead of relying on the tool itself to optimize for all possible use cases

- Twitter funds exciting open source experimental build, compiler, and VM work!