Code Coverage - Areas to improve

Please read Bazel Code of Conduct before commenting.

Authors: cmita@google.com
Status: **Draft** | In review | Approved | Rejected | In progress | Implemented
Reviewers:
Created: 2023-07-28
Updated: 2023-07-28
Discussion thread: https://github.com/bazelbuild/bazel/discussions/19144

Overview

This document lists the current deficiencies and potential areas for improvement with respect to Bazel's code coverage support. The goal is to better aid rule authors in implementing coverage support in rules.

Problems

Documentation deficiencies

The Bazel docs do not adequately explain:

- What is currently supported for combined coverage reports.
- What is expected of rule implementations that want to support coverage collection.
- How rules can integrate with Bazel's coverage collection.

The principal reason for the lacking documentation is the lack of support actually available. Despite this, some rules have been able to implement coverage support that integrates with Bazel's combined reporting, although with some limitations.

API deficiencies

Bazel provides the following explicit APIs for rules to implement coverage collection:

- A means to determine if a build is being run in coverage mode via `ctx.configuration.coverage_enabled`.
- Means to determine if a target (the current target or any dependency) should be instrumented via `ctx.coverage_instrumented`.

- A way of indicating which source files will be instrumented (and, as of recently, what other files are relevant for coverage collection) in `InstrumentedFilesInfo`.

Additionally, rules can define the following attributes which affect the behavior of coverage collection, but this is not a supported API.

- `_lcov_merger` attribute - test rules can specify this attribute and provide a tool that will be invoked after test execution. It collates whatever supported coverage files (LCOV or gcov outputs) are found for that test run into a single LCOV file.
- `_collect_cc_coverage` - can be used to provide a script that will also be invoked after test coverage, but before the `_lcov_merger` tool. The actual purpose of this is for C++ rules to process the toolchain's coverage output before further processing with lcov tool.

In addition to being undocumented and unsupported, they only affect the behavior of test rules. This leaves the issue of cross-language support (where a test binary in language X depends on a library in language Y); the only way this can currently be supported is either:

- `x_test` handles coverage for `y_library`
- `y_library` (or the Y rules in general) handles its own coverage data, likely at the boundary between X and Y (feasible for languages with their own runtimes, but tricky if they produce libraries or archives that can be linked by other language toolchains (i.e. C/C++)).

Additionally, the semantics of `InstrumentedFilesInfo` is more complicated for rule implementations than it arguably should be. Rules must specify which dependencies should have their `InstrumentedFilesInfo` passed along (by specifying the attribute name rather than targets within). Failure to include all dependencies will "break the chain" (coverage won't be reported for anything along the missing edge). This motivates the existence of code to automatically forward the provider from all non-tool dependencies if none is configured for a particular target (https://cs.opensource.google/bazel/bazel/+/f868582d:src/main/java/com/google/devtools/build/lib/analysis/test/InstrumentedFilesCollector.java;l=65).

Confusing flags

There are several command line flags related to coverage collection, some of which are confusingly named, alter tooling that is otherwise internal, or just have unexpected behavior:

- `--coverage_output_generator` controls the setting of `_lcov_merger` for many rules.
- `--coverage_report_generator` determines what tool is used to generate the combined coverage report after all tests have run (the merged output).
- `--coverage_support` provides "collect_coverage.sh".
- `--collect_code_coverage` doesn't actually trigger coverage collection, but simply requests that targets are built with instrumentation enabled.

- `--instrumentation_filter` takes a set of regexes matching target names, not source paths, which may be unintuitive. It also has some complicated default behavior based on test targets that is not fully documented.
- `--instrument_test_targets` enables instrumentation of test targets (disabled by default). Rules have no way of customizing this behavior since this is handled inside of `ctx.coverage_instrumented()`.
- `--combined_report` triggers the creation of the merged coverage report (combining lcov reports from all tests). Only the values `lcov` and `none` are supported.

The naming of these flags isn't consistent; half begin with "--coverage_" and half do not. Further, several change global implementation details, affecting test runs for all rules.

Reporting deficiencies

Currently Bazel can only process and output LCOV files (the tracefile format used by lcov). Whilst these are very easy to read and produce because they are line-based plain text files, it is not a widely used format for other languages and tools and is not useful for human consumption.

HTML output is much better for a person to read and something like Cobertura XML may be better for integration with other tooling.

Baseline Coverage

The Bazel docs briefly mention baseline coverage, describing it as "broken".

This technically exists at the "file" level; during a build with `--collect_code_coverage`, Bazel will generate "baseline_coverage.dat" files for each target, but they will effectively only contain a list of files for that target; there will be no function, line, or branch details.

Because the details of how to generate a useful baseline report differ between languages, it will again be up to rule implementations to decide how to do it. However; because baseline coverage doesn't require output binaries to be executed to gather data, it should be something that can be reasoned about at analysis time.

Additional platform specific problems

OS X

C++ coverage doesn't integrate well at the moment. For gcov coverage, Bazel depends on the ability to set "PWD" to `/proc/self/cwd` during compilation; overriding the path for the `.gcda` that will be output later, providing an easy fixed prefix that can be stripped. This facility is not available on OS X.

There is experimental support to process LLVM's source-level coverage into the LCOV format. However, for this to correctly integrate with Bazel's LCOV merging, the

`/proc/self/cwd` issue still needs to be resolved. This may be fixable within the merge tool however (unlike with the GCC solution, an LCOV file is actually produced by `llvm-cov`, so the merge tool may be able to match filenames to files within its input list).

Windows

Coverage execution in general requires that Bash. However Bazel as a whole generally assumes that Bash is available, so this may not be an issue.

As far as I know, C++ coverage is not supported on Windows.

Potential API improvements

Cleanup but "change nothing"

This acknowledges that coverage collection "basically works" (that is, sufficient for many languages), although is undocumented. Documentation would be improved and the expectations of rules would be made clear.

The issue around `_collect_cc_coverage` would remain. However we could formalize this as part of the C++ rules which Bazel controls. There may be issues for rules_rust, but if they use LLVM, it "may just work".

Allow rules to specify test-time conversions

For the majority of rules, this would not be necessary. However for some there really is no alternative to having some shell code run after test execution (C/C++ and potentially Rust). It is also desirable for test rules depending on such rules to not need to worry about specifying a `_collect_cc_coverage` target.

How this would work needs to be determined, with care taken to avoid too much running during test execution (we don't want a shell script running per transitive cc_library target for instance after the tests have run).

Baseline Coverage

The requirements here are simple:

1. Baseline coverage should be generated as part of a build, not execution.
2. Rules need a way to provide a baseline coverage artifact to Bazel. This could either be through an Output Group or a special provider.
3. Bazel needs to collate all output baseline reports and merge them as it would for coverage runs.
4. Baseline coverage needs to be collectible for non-test targets.

The details for how to generate a baseline coverage report vary between languages and tools, so Bazel cannot do anything itself (beyond generating an empty one as it does now), so it will be up to rule authors to implement.

Because a baseline coverage report does not require execution data (it represents a "null-run"), it should be able to be generated after a build, without execution, and can therefore be reasoned about at analysis time. That is, a rule can declare a "baseline coverage file" and generate an action for it.

Platform support

If the full logic for coverage collection and conversion to LCOV is handled by the rules (that is, rules are given a way to specify post test execution steps), then this is largely a matter for rules. Bazel must simply ensure that whatever wrapping logic supports multiple platforms and that report merging works. This doesn't change the amount of work to be done, rather it just moves it from Bazel's internal Java code to the Starlark API and rule implementations.

Document History

Date	Description
2023-07-28	First suggestions