Support Symlink Runfiles Tree on Windows

Author: pcloudy@google.com

This is a public document

TL;DR: Bazel now supports symlink runfiles tree on Windows with `--experimental_enable_runfiles` flag. It requires admin right or enabling developer mode on Windows 10 Creator Update (1703) or later. Be aware that it might slow down your build if your targets have lots of runfiles, because creating symlink is slow on Windows. This feature is available with Bazel built from HEAD and will be released in 0.18.0.

Motivation

A binary's runfiles are files needed during runtime. Bazel creates a runfiles directory next to the binary, which contains a symlink tree of its runfiles. When binaries are running under the runfiles directory, they have access to their runfiles by relative path. On Windows, the symlink runfiles tree is disabled because creating symlink is not allowed by default. This causes the following problems:

- Binaries cannot access their runfiles by relative path. To solve this, we introduced Bazel Runfiles Libraries. It provides libraries for Python, Java, C++ to map a relative runfile path to its actual absolute path.
- Python binary cannot be built as a collection of python source files under runfiles directory. We had to build a self-extracting python zip file in order to run on Windows. See more details at Build Python on Windows.

However, both solutions are not perfect for their problem.

The Runfiles Libraries require user to introduce Bazel specific code in their program. This is problematic when your program depends on a third party module, which has no knowledge about Bazel Runfiles Libraries and still requires a relative path for the data file. (See an issue from Angular, #5926) Moreover, for languages that do not have official runfiles libraries, rule authors have to write their own runfiles library. If any of the rules you depend on doesn't support runfiles library, your whole project will have reduced functionality.

Building python binary as a self-extracting zip file generally works. But it requires compressing all runfiles during build time and extracting them during **each** runtime. When the python binary depends on large size data files or C++ extensions (TensorFlow), the performance regression is significant, especially for running python tests.

Fortunately, Microsoft still leaves us a way to create symlink on Windows. As requested by our user (#5807), we should support symlink runfiles tree on Windows when it is possible.

Symlink on Windows

Creating symlink on Windows is allowed if any of the two requirements are fulfilled:

- The program is running with elevated privileges (administrator rights)
- The system version is Windows 10 Creators Update (1703) or later and developer mode is enabled.

For more history about Windows symlink, read this blog post.

Like Unix, symlink can be created and deleted even when the target is in use. And it can even point to a path that doesn't exist (dangling symlink).
Unlike Unix, Windows distinguishes file symlink from directory symlink. You have to specify the symlink type during the creation and use `DeleteFile` or `RemoveDirectory` API to delete them respectively.

Performance

Compared to Linux, symlink operation is much slower on Windows.
A benchmark: https://github.com/meteorcloudy/my_tests/tree/master/symlink_test

In this benchmark, creating and deleting symlink directly use Win32 API (CreateSymbolicLinkW and DeleteFileW/RemoveDirectoryW). There's also a Win32 API GetFinalPathNameByHandleW for resolving symlink, but it is very slow, therefore we implemented symlink resolving by reading symlink reparse point.

The time consumed by symlink operations on 10000 files				
	System Info	**Create file symlink**	**Resolve file symlink**	**Delete file symlink**
Windows workstation	Windows 10, 12 cores, SSD	12.04s	1.16s	0.51s
GCE Windows worker	Windows Server 2016, 32 cores, HDD	3.80s	1.15s	0.77s
Windows laptop	Windows 10, 8 cores, SSD	4.07s	1.48s	0.78s
Linux	Debian 4.9, 12 cores, SSD	0.10s	0.030s	0.071s

A few things observed from the result:

- Symlink creating time varies on different Windows machines, it's about **40 ~ 120 times** longer than Linux.
- Resolving symlink is almost the same time on Windows, it's much faster than creating symlink but still about **40 times** slower than Linux.
- Deleting symlink is much faster than creating symlink on Windows.

Implement build-runfiles tool

Bazel uses a native C++ binary (build-runfiles) to create symlink runfiles tree. On Windows, the build runfiles tool consists of following parts:

Check Symlink Creation

The build-runfiles tool first tries to create a dummy symlink with `CreateSymbolicLinkW` API. `SYMBOLIC_LINK_FLAG_ALLOW_UNPRIVILEGED_CREATE` flag will be used if it's possible to create symlink without admin privilege. If it's not possible to create symlink, the tool prints out an error message asking for symlink privilege:

```
CreateSymbolicLinkW failed:
Bazel needs to create symlink for building runfiles tree.
Creating symlink on Windows requires either of the following:
    1. Program is running with elevated privileges (Admin rights).
    2. The system version is Windows 10 Creators Update (1703) or later and
developer mode is enabled.
```

Read Manifest File

The build-runfiles tool reads a given manifest file and stores every runfile entry (runfile path -> absolute path) in a map.

Scan and Prune Runfiles Tree

It's very possible there's already an existing runfiles tree from previous build. We walk the runfiles directory recursively, check every existing file and directory. We preserve a symlink if it meets all the following requirements:

- It exists in the runfiles map
- It's already pointing to the correct target
- the target type matches the symlink type (file vs directory)

At the same time, we remove the corresponding runfile entry from the runfiles map to avoid recreating it. We then delete all other files and empty directories.

Create Runfiles Tree

The build-runfiles tool creates symlinks for all remaining entries in the runfiles map. It checks the target type first to tell which symlink type it should create (file vs directory). Note that, there might be entries pointing to an empty target (eg. __init__.py -> “”). In this case, we just create an empty file at its runfile path.

Summary

With `--experimental_enable_runfiles` flag, Bazel now supports building symlink runfiles tree on Windows. Before you use this feature, you should consider its pros and cons.

Pros

- Programs can access data files (or other runfiles) in the same way as Linux. No special consideration needed for Windows, this will help porting Bazel rules to Windows.
- Python binary doesn’t have to be built as a zip file.

Cons

- Requires admin right or developer mode on Windows 10 Creator Update (1703) or later to be enabled.
- Creating symlink is very slow on Windows. This is a performance issue when targets have a large number of runfiles. But this issue can be eased during rebuild, because
    - If runfiles doesn’t change, the whole runfiles action will be cached by Bazel
    - If runfiles changes, build-runfiles tool will only create symlinks for new runfiles, which is usually a small number.