Python wheels & ML framework issues

[This document is public]

Overview and previous meeting notes are below

Conference Call 19th Feb 5pm UTC

https://meet.google.com/udn-afkm-rsa
Dial-in: +1 510-957-3168 PIN: 209 105 591#

Attendees:

Please add your name and affiliation here. And say who you are the first time you speak.

1. Jason Zaman (Gentoo Linux / TF SIG-Build, jason@perfinion.com)
2. William Irons (IBM Power, wdirons@us.ibm.com)
3. Manuel Klimek (Google, klimek@google.com)
4. Uwe Korn (Blue Yonder / Apache Arrow, mail@uwekorn.com)
5. Jonathan Helmus (Anaconda, jhelmus@anaconda.com,)
6. Jason Furmanek (IBM, furmanek@us.ibm.com)
7. Brad Nemanich (IBM, brad.nemanich@ibm.com)
8. Dustin Ingram (Google, dustiningram@google.com, PyPA, di@python.org)
9. Taylor Jakobson (IBM Power, tjakobs@us.ibm.com)
10. Dmitri Gribenko (Google, dmitrig@google.com)
11. Soumith Chintala (Facebook / PyTorch, soumith@fb.com)

Meeting Notes

1. Conclusion from last meeting:
    a. some form of devtoolset 7 for a different toolchain and building using that
    b. Discussion about writing a new PEP for manylinux
        i. Recent discussion on perennial manylinux tags: https://github.com/pypa/warehouse/issues/3668#issuecomment-464018800
2. Manuel from Google:
    a. Internally use cuda-clang (instead of nvcc) for everything
    b. Bugs in nvcc take time to resolve, but Google is an active contributor to Clang.
    c. Is cuda-clang upstream? Yes, all of it.

d. Google provides public Docker images of LLVM that are used internally at Google, which are chosen after lots of careful testing and are good enough for Google internally, can be good enough for people outside of Google as well?
e. Does cuda-clang imply the need to use libc++? No.
f. Older CentOS has older libstdc++, can Clang target those? Yes. The only issue is that TensorFlow can depend on newer C++ standards. LLVM is going to require C++14 in March 2019.
g. How much of this toolchain can we shim on top devtoolset 7? The libsdc++ in devtoolset 7 does not have the new ABI. In devtoolset 7 libstdc++ is not a real library, it is a linker script that redirects to the system library for symbols that exist, and statically links other symbols.
h. What is the advantage of using this linker script approach over shipping a new libcstdc++ or libc++?
i. In SIG-Networking there were concerns about linking TF into a library that links Python, and Infiniband bits that would all talk using C++ objects over the ABI boundary.
j. One issue is that Python's manylinux spec says that you can use only a certain list of symbols, but shipping another standard library kinda violates that. However, in practice, it maybe does not matter since the new binary does not require those symbols from the system standard library.
k. In Gentoo, we had a hard time during the libstdc++ C++11 ABI change. However, at some point, there will be a new manylinux spec, and we can't stay at the old ABI forever, would we able to switch ABI?
l. Jonathan: the assumption is that a package should be self-contained. As long as you don't pass any C++ types outside of libraries inside a package, you can use any ABI you want.
m. There are two things we are trying to solve. One is, manylinux tries to define minimal requirements for the host system. The other is, providing a modern toolchain for building PIP packages. And we want both -- we want a manylinux that is very compatible, but we also want a modern toolchain.
n. Jason: the biggest problem for me is the ABI change.
o. If we shipped a libc++ in a wheel, we would not depend on the system libstdc++.
p. Jonathan: it is difficult to use a newer GCC to target an older glibc. Anaconda uses a new GCC (bootstrapped), with glibc 2.17.
q. Shipping a newer libstdc++ would be effectively "putting Anaconda into PIP".
r. Manuel: we could have a "toolchain-base" wheel that ships all runtime libraries, and TensorFlow would depend on toolchain-base. We would also have a "toolchain" wheel that provides the compiler that targets those runtime libraries.
s. pynativelib has been proposed
t. libcuda.so needs to come from the system though. TensorFlow will not link to it, but it will dlopen() it.
u. How stable is the ABI of libcuda.so? Can you build against one and run against another? Pretty stable in some version ranges, but gets complicated.
v. CUDA 10 deprecated CentOS 6.

w. Anaconda has a solution for CUDA, with the cuda-toolkit package. What about us?
x. Also cuDNN, which can't be re-distributed. Only the headers.
y. We would be in a better position to make a request to NVidia, if we had a good idea about what we want the solution to look like.
z. cuDNN is unstable in its performance characteristics, both improvements and regressions. PyTorch wants to ship the latest version possible because of that.
aa. Is ABI stable? Yes. However the performance changes are very drastic.
bb. When should we do a follow-up? Maybe at the dev summit? A breakout session on day 2?

Overview

TensorFlow and other python-based projects are having issues with compatibility with the pip manylinux1 spec. There are two main issues that need solving: C++11 support and CUDA support.

There has been a lot of discussion on this mailing list thread:
https://groups.google.com/a/tensorflow.org/forum/#!topic/build/WgtWKA4t_bs

Writeup of issues and solutions by Dmitri Gribenko <dmitrig@google.com>. Reproduced here from the mailing list:

> So I wanted to go back to the requirements and enumerate possible solutions.
>
> From soumith's email:
> 1. CUDA support
> 2. C++11 support
>
> Neither newest CUDA, nor C++11 work on manylinux1 (CentOS 5.11).
>
> The original email does not go into detail why CUDA does not work, but I can imagine it is because of the old userspace libraries (libc, libstdc++, libpthread etc). C++11 does not work because of an old libstdc++ and old GCC.
>
> **So what can we do about old userspace libraries?**

Option "Userspace-1": Pip package uses libraries installed on the system where the pip package runs. (AKA the current manylinux approach.)

Advantages:
- Smaller download size.

Disadvantages:
- Pip packages have to be built against an old version of userspace libraries to be maximally-compatible.
- No nice upgrade path. When we need a specific new feature for something (e.g., today it is modern CUDA and C++11), we have to bump the requirements for the host system. We will always be extremely cautious about not bumping the requirements too much, and therefore we will be always stuck with oldest possible libraries that can do the job.

Option "Userspace-2": When the pip package runs, ignore the system userspace libraries. Use libraries from somewhere else.

Advantages:
- We control which versions of userspace libraries we use. We can use libraries that are newer than system ones.
- Complete isolation from userspace of the system where the pip package runs. The only remaining point of contact with the user's system is the kernel.

Disadvantages:
- We need to figure out where to get these libraries from.
- Bigger download size for users.

So where do we get the userspace libraries from?

Option "Userspace-2a": Pip community owns all userspace libraries that binaries in a pip package can use.
All userspace components defined by manylinux are packaged into a pip package. TensorFlow/PyTorch/... pip packages declare what version of the userspace pip package they depend on.

Advantages:
- Pip community owns all userspace components.

Disadvantages:
- Pip community owns way more stuff than before.

Option "Userpace-2b": Pip takes all userspace libraries from an existing packager.
Same as "Userspace-2a", but instead of owning the build process for the userspace libraries, we take them from an existing packager, for example, Debian, CentOS, Anaconda, Nix package manager, whatever we decide on.

Advantages:

- Pip community controls userspace components.

Disadvantages:
- Pip community owns more stuff than before.

What can we do about old toolchain?

Option "Toolchain-1": Use a toolchain from a certain old distribution, so that the output is maximally-compatible.
This option is compatible with any choice of userspace, as long as the libraries don't require a new compiler or language features.

Disadvantages:
- Ancient toolchain that does not support modern C++.

Option "Toolchain-2": Make a modern toolchain that produces maximally-compatible output.
This option is difficult to implement, since a modern toolchain using a modern C++ version will require a using a contemporary C++ standard library (libc++ or libstdc++).

Option "Toolchain-3": Make a modern toolchain that requires a modern C++ library.
AKA what Manuel is proposing. Package modern libc++ as a wheel, make a Docker container with the corresponding Clang for building binary packages like Tensorflow.

TensorFlow SIG-Build meeting 5th Feb

Summary:

1. Need to use a newer compiler and statically link newer dependencies.
    a. PyPA is more concerned with compatibility than with any wheel size increase.
    b. DevToolSet7 on centos6/centos7 should be able to handle most of our needs for C++11
2. Should investigate submitting PEP for newer manylinux versions. (eg manylinux2014)
3. Also need PEP to support more architectures (eg ppc64le) for existing and new manylinux standards
4. Many from Europe could not attend meeting time. Will schedule a follow up call at a better time slot.

Meeting Agenda:

1. Full SIG-Build notes are in this doc: https://docs.google.com/document/d/10_3IQ5aF-88ADJNLF0WOpb09bZ15x-sBnRSnDHNCNr8/edit#
2. Separate discussion from normal agenda to focus on the pip wheel issues we and other projects are hitting
3. Manylinux1 / manylinux2010
4. https://groups.google.com/a/tensorflow.org/forum/#!topic/build/WgtWKA4t_bs
    - Big Email thread with discussion. Basically, TF does not build a pip wheel in the standard way (many others do not too)
        - TF uses C++11, wheel uses an old glibc that doesn't support it
        - CUDA issues are not supported
5. Two sets of issues: CPU-only (C++11) vs CUDA
    - Both fail auditwheel :(
6. Lets fix cpu-only first since cuda is harder?
    - Will we hit issues later if we ignore cuda at first?
    - CUDA is also more complicated because we cannot legally redistribute cudnn
7. Hopefully people from other communities can make the meeting.
    - Need to send out meeting notes for those who could not make it.
    - Follow up meeting specifically for wheel discussion only. When?
8. Centos7+devtoolset+manylinux2010+CUDA

Attendees:

1. Jason Zaman (Gentoo Linux / TF SIG-Build, jason@perfinion.com)
2. Subin Modeel (RedHat , smodeel@redhat.com)
3. Austin Anderson (TensorFlow DevInfra, angerson@google.com)
4. William Irons (IBM Power, wdirons@us.ibm.com)
5. Michael Sarahan (Anaconda, msarahan@anaconda.com) - on the bus home; on mute.
6. Ton Ngo (IBM Cognitive Open Tech, ton@us.ibm.com)
7. Dennis Jenkins (Google, dennisjenkins@google.com)
8. Taylor Jakobson (IBM Power, tjakobs@us.ibm.com)
9. Edd Wilder-James (Google, ewj@google.com)
10. Jonathan Helmus (Anaconda, jhelmus@anaconda.com)
11. Yifei Feng (Google, yifeif@google.com)
12. Gunhan Gulsoy (Google, gunan@google.com)
13. Paige Bailey (Google, webpaige@google.com)
14. Jason Furmanek (IBM, furmanek@us.ibm.com)
15. Sean Morgan (TensorFlow Addons SIG, seanmorgan@outlook.com)
16. Jonathan Wakely (RedHat, libstdc++ jwakely@redhat.com https://github.com/jwakely)
17. Dustin Ingram (Google, dustiningram@google.com, PyPA, di@python.org
18. Philipp Moritz (UCB)

General notes from discussion:

1. Pytorch and anything with C++11 or CUDA features also run into the same problems TF does
2. Manylinux versions are only defined as “build with CentOS 5” or “build with CentOS 6”, (centos 6 is 9 years old) nothing about platform (x86, ARM?)
    - Manylinux 2010 does specify c++
    - architecture is embedded in wheel name
    - manylinux only designed for x86_64, i686
    - guidance seems to have been “someone should have submitted another PEP”
3. little-endian PPC standard old as well for manylinux
4. In addition to glibc, TF would like a lot more information included in the packages (libstdc++ version, glibc version, CUDA + dependent system libraries, etc.),
5. can we use centos 6 and dev toolset to solve gcc/libc issues?
    - can install new clang, but can’t install a too-new libstdc++ version
6. devtool sets are supposed to add a linker script so that base version of libstdc++ version is fixed, but can dynamically link in newer versions
7. (libstdc++ head dev visiting in call, thanks for being here!)
8. some folks(subin,Jonathan,Torvald) from redhat offered to assist with kernels / libstdc++ version development if we can resolve things that way
    - we could try centos 6 with devtool set, and if that doesn’t work perhaps we could go with moving the TF build to CentOS 7 (as previously discussed internally)
9. Manylinux using the name of the package to set platform deps is a limiting format, is there anything else we can do?
10. many related fixes in libstdc++ have gone in fairly recently, so newer gcc may address some of the bugs TF’s build has seen
11. devtoolset 2,3 or gcc 4.8 - c++11 support is not great (not finished until gcc 5), so that shouldn’t be our foundation
    - However, TF first started with Ubuntu 14 (gcc 4.8) with support of limited c++11 set
    - Ubuntu 16 attempted upgrade had huge community backlash because 16 builds did not work on 14 (std string change incompatible, gcc dual ABI adjustment is impossible)
12. manylinux 2010, 2014 - do all of them need that macro?
13. on a new system, we can avoid needing any of those changes, but because of pypi we can only put up 1 manylinux 1 package
14. however, we could possibly use that macro on centos6 with devtoolset 7 as a way to build for both for to fix the problems c++11, libstdc++ problems (see thread, linked above for more information)
    - certain c++11 features cannot be backported due to .so needs, but those are mostly locale-related

        - can we warn on those types of issues? it sounds like they just won't compile, because devtoolset forces the old ABI
    - TF code has to work anyway because we use gcc 4.8 right now
        - (although we could possibly use some newer features)
15. Does newer devtoolset make the wheel slower, because of the dynamic linking?
    - No, probably not, although binaries will be a bit bigger because of the linker script that does some more static linking than usual
    - Basically statically cherry-picks newer symbol availability to use them as able, to get a hybrid linking system with new features for an old OS
16. Anaconda python ships with libstdc++ and libgccs, if TF is split into multiple shared objects (which is planned, and hoping to have them be built by other groups, not just google, so they would build and ship their shared object). If we used devtoolset 7 (which has some static linking in our so) and we get another shared object loaded into the same address space (and may be built on a new os with new features), does that mean the newer features will be loaded twice?
    - That shouldn't be a problem, the symbols should be compatible (symbols should not be changed too much)
        - For c++11, only true since gcc 5 (not true if mixing with gcc < 5)
    - If we use a newer GCC, we may be able to solve TF's problems on Anaconda :)
17. gcc5 was a big improvement for Gentoo also
18. Jason is planning to meet with the Sean Morgan (Addons Lead) and other SIG guides so that SIG Build can look at providing build standards for TF extensions
    - TF also wants to have a C api to help with this, but not getting into it now (to avoid having everyone need to compile with the same image, except that Pypi requires it)
19. From Jonathan Wakely <jwakely@redhat.com>:
    - I explained what's supported when mixing gcc versions and c++11/c++14 etc. at https://stackoverflow.Leacom/a/49119902/981959
20. NVidia won't be able to follow Pypi guidelines if they publish TensorRT stuff
21. CUDA is tough because we can't redistribute it, so we depend on them
22. Installing a random CUDA image from pip sounds like it wouldn't work because they would likely conflict with system CUDA, etc.
23. RedHat constantly tests NVidia/CUDA/gcc stuff, so RedHat is ensuring compatibility, also so that hopefully things downloaded from PyPi would continue to work on rhel platform.
24. What about the other possible wheel dependencies (CUDA, MKL)? Could they conflict with installed system libraries?
    - Should we propose a system to let pip packages define system library requirements (PEP)?
        - Would also need to include a way to say "this is where you should get these things" (like consistent set of environment variables, etc.)
            1. See "environment markers" idea? Could possibly expand this to say "the system needs to support CUDA, etc.)

        2. “Environment markers for GPU/CUDA availability”: https://mail.python.org/archives/list/distutils-sig@python.org/thread/LXLF4YSC4WUZOYRX65DW7CESIX7UUBK5/?sort=date
    - Could also define architectures (PPC, ARM, Endian support) (should probably do these two things separately)
        1. Wheels can declare what architectures they support, but we can’t do RPi, etc.
        2. Someone would need to have a new PEP to support more architectures (perhaps a new PEP for each new architecture)
  - Numba example - hard to find the nvidia dependencies
25. PPC support would probably need CentOS 7 to build. Would we lose anything by moving to CentOS 7 for the builds? DevToolSet7 sounds promising, however
26. TF also had a meeting last week with Dustin Ingram and Thea; there is interest from Python’s side for these features as long as we can provide work to implement it.
    - Also discussion of perennial manylinux tags, may help support the architecture problems (“Idea: perennial manylinux tag” https://mail.python.org/archives/list/distutils-sig@python.org/thread/6AFS4HKX6PVAS76EQNI7JNTGZZRHQ6SQ/?sort=date)
    - Work: write the PEP, once it is approved would need to update auditwheel, pip, as well as pypi backends, etc.
    - Pip has 300Mb limits for packages, TF’s wheels are historically very large (at one point may have been 30% of PyPI’s storage usage)
    - Dustin is also a moderator/maintainer of PyPI, sizing the wheels is slightly less critical now because of cloud support from Google
27. Generally we may need separate meetings (with a Europe-compatible time) for this separate discussion chunk (Edd to help facilitate)
28. If we go with a new PEP, would we need to rebuild a bunch of stuff?
    - If it works right, the old wheels should still be compatible but the new ones would have more symbols/info in them
29. From TF side, we would only rebuild last 6 months of packages (if needed) (we do want backwards compatibility, since a breaking change would likely not be accepted by Pypi)
    - Building with c++11=1 would be a breaking change, however, unsure when that would happen (string mangling would change; if everything only had a C API and used strings internally, no problem)Would also need to include a way to say “this is where you should get these things” (like consistent set of environment variables, etc.)
        - Both c++ abis are running inside the par right now
    - TF only guarantees newer than Ubuntu14 and newer than CentOS 7 support
30. advantage: passing through Python makes it easier
31. For numpy, we directly include numpy headers (but numpy does not have C++ at all)
32. How do we prevent running into the kind of problem we’re running into now a second time?

33. We currently use Manylinux1 because that's the only tag we're able to use (we need to tell them apart from Win, MacOS packages)
34. Jonathan Wakely -- w.r.t "how do we prevent running into the same problems again?" -- I think using gcc5 or newer will help there. As explained at https://stackoverflow.com/a/49119902/981959 since gcc5 the c++11 support is stable and backwards compatible. Before gcc5, if you mix c++11 code built with gcc 4.8 and c++11 code built with gcc 4.9, or 5, or 6, all bets are off .More info
    - C++17 noexcept should not affect ABI, so this should not be a problem for TF's future (see notes above)
    - C++20 also will have a char type change, but newer C++ has not seen anything so severe as the C++11 string change

If there's going to be effort towards one or more PEP's, we should start with the history (there's a lot):

- https://www.python.org/dev/peps/pep-0426/
- https://www.python.org/dev/peps/pep-0566/
- https://mail.python.org/mm3/archives/list/distutils-sig@python.org/message/SKNPR5ZFGMF7MQJNGIRCSQH4YOKYTVQZ/
- https://github.com/pypa/interoperability-peps/pull/30/files
- http://code.activestate.com/lists/python-distutils-sig/32275/
- [Distutils] Opinions on requiring younger glibc in manylinux1 wheel?
    - https://mail.python.org/archives/list/distutils-sig@python.org/thread/M4MSVY5MPAPXFWHH4PBLE6PEBPOBIA44/?sort=date
- [Distutils] Environment markers for GPU/CUDA availibility
    - https://mail.python.org/archives/list/distutils-sig@python.org/thread/LXLF4YSC4WUZOYRX65DW7CESIX7UUBK5/?sort=date
- [Distutils] Idea: perennial manylinux tag
    - https://mail.python.org/archives/list/distutils-sig@python.org/thread/6AFS4HKX6PVAS76EQNI7JNTGZZRHQ6SQ/?sort=date