Installation

From Conda

Conda is a package manager for Python, CPP and other packages.

Currently, SINGA has conda packages (Python 2.7 and Python 3.6) for Linux and MacOSX.Miniconda3 is recommended to use with SINGA. After installing miniconda, execute the one of the following commands to install SINGA.

1. CPU only

   ```
   conda install -c nusdbsystem singa-cpu
   ```

2. GPU with CUDA and cuDNN

   ```
   conda install -c nusdbsystem singa-gpu
   ```

   CUDA driver (for CUDA >=9.0) must be installed before executing the above command. Singa packages for other CUDA versions are also available. The following instruction lists all the available Singa packages.

   ```
   conda search -c nusdbsystem singa
   ```

If there is no error message from

```
python -c "from singa import tensor"
```

then SINGA is installed successfully.

From source

The source files could be downloaded either as a tar.gz file, or as a git repo

```
$ git clone https://github.com/apache/incubator-singa.git
$ cd incubator-singa/
```

Use Conda to build SINGA

Conda-build is a building tool that installs the dependent libraries from anaconda cloud and executes the building scripts. The generated package can be uploaded to anaconda cloud for others to download and install.

To install conda-build (after installing miniconda)

```
conda install conda-build
```

To build the CPU version of SINGA

```
conda build tool/conda/singa/ --python 3.6
```

The above commands have been tested on Ubuntu 16.04 and Mac OSX. Refer to the Travis-CI page for more information.

To build the GPU version of SINGA

```
export CUDA=x.y (e.g. 9.0)
conda build tool/conda/singa/ --python 3.6
```

The commands for building on GPU platforms have been tested on Ubuntu 16.04 (cuDNN>=7 and CUDA>=9). Nvidia's Docker image provides the building environment with cuDNN and CUDA.

The location of the generated package file is shown on the screen. Refer to conda install for the instructions of installing the package from the local file.

Use native tools to build SINGA on Ubuntu

The following libraries are required to compile and run SINGA. Refer to SINGA Dockerfiles for the instructions of installing them on Ubuntu 16.04.

- cmake (>=2.8)
- gcc (>=4.8.1)
- google protobuf (>=2.5)
- blas (tested with openblas >=0.2.10)
- swig(>=3.0.10) for compiling PySINGA
- numpy(>=1.11.0) for compiling PySINGA

1. create a `build` folder inside incubator-singa and go into that folder
2. run `cmake [options] ..` by default all options are OFF except `USE_PYTHON`

   - `USE_MODULES=ON`, used if protobuf and blas are not installed a prior
   - `USE_CUDA=ON`, used if CUDA and cuDNN is available

- `USE_PYTHON3=ON`, used for compiling with Python 3 support. (The default is Python 2)
- `USE_OPENCL=ON`, used for compiling with OpenCL support
- `USE_MKLDNN=ON`, used for compiling with Intel MKL-dnn support
- `PACKAGE=ON`, used for building the Debian package
- `ENABLE_TEST`, used for compiling unit test cases

3. compile the code, `make`
4. goto python folder
5. run `pip install .` or `pip install -e .` The second command creates symlinks instead of copying files into python site-package folder.

Execute step 4 and 5 are to install PySINGA when USE_PYTHON=ON.

After compiling SINGA with ENABLE_TEST=ON, you can run the unit tests by

```
$ ./bin/test_singa
```

You can see all the testing cases with testing results. If SINGA passes all tests, then you have successfully installed SINGA.

Compile SINGA on Windows

Instructions for building on Windows with Python support can be found here.

More details about the compilation options

USE_MODULES

If protobuf and openblas are not installed, you can compile SINGA together with them

```
$ In SINGA ROOT folder
$ mkdir build
$ cd build
$ cmake -DUSE_MODULES=ON ..
$ make
```

cmake would download OpenBlas and Protobuf (2.6.1) and compile them together with SINGA.

You can use `ccmake ..` to configure the compilation options. If some dependent libraries are not in the system default paths, you need to export the following environment variables

```
export CMAKE_INCLUDE_PATH=<path to the header file folder>
export CMAKE_LIBRARY_PATH=<path to the lib file folder>
```

USE_PYTHON

Similar to compile CPP code, PySINGA is compiled by

```
$ cmake -DUSE_PYTHON=ON ..
$ make
$ cd python
$ pip install .
```

USE_CUDA

Users are encouraged to install the CUDA and cuDNN for running SINGA on GPUs to get better performance.

SINGA has been tested over CUDA 9, and cuDNN 7. If cuDNN is installed into non-system folder, e.g. /home/bob/local/cudnn/, the following commands should be executed for cmake and the runtime to find it

```
$ export CMAKE_INCLUDE_PATH=/home/bob/local/cudnn/include:$CMAKE_INCLUDE_PATH
$ export CMAKE_LIBRARY_PATH=/home/bob/local/cudnn/lib64:$CMAKE_LIBRARY_PATH
$ export LD_LIBRARY_PATH=/home/bob/local/cudnn/lib64:$LD_LIBRARY_PATH
```

The cmake options for CUDA and cuDNN should be switched on

```
# Dependent libs are install already
$ cmake -DUSE_CUDA=ON ..
$ make
```

USE_OPENCL

SINGA uses opencl-headers and viennacl (version 1.7.1 or newer) for OpenCL support, which can be installed using via

```
# On Ubuntu 16.04
$ sudo apt-get install opencl-headers, libviennacl-dev
# On Fedora
$ sudo yum install opencl-headers, viennacl
```

Additionally, you will need the OpenCL Installable Client Driver (ICD) for the platforms that you want to run OpenCL on.

- For AMD and nVidia GPUs, the driver package should also install the correct OpenCL ICD.
- For Intel CPUs and/or GPUs, get the driver from the Intel website. Note that the drivers provided on that website only supports recent CPUs and Iris GPUs.
- For older Intel CPUs, you can use the `beignet-opencl-icd` package.

Note that running OpenCL on CPUs is not currently recommended because it is slow. Memory transfer is on the order of whole seconds (1000's of ms on CPUs as compared to 1's of ms on GPUs).

More information on setting up a working OpenCL environment may be found here.

If the package version of ViennaCL is not at least 1.7.1, you will need to build it from source:

Clone the repository from here, checkout the `release-1.7.1` tag and build it. Remember to add its directory to `PATH` and the built libraries to `LD_LIBRARY_PATH`.

To build SINGA with OpenCL support (tested on SINGA 1.1):

```
$ cmake -DUSE_OPENCL=ON ..
$ make
```

USE_MKLDNN

User can enable MKL-DNN to enhance the performance of CPU computation.

Installation guide of MKL-DNN could be found here.

SINGA has been tested over MKL-DNN v0.17.2.

To build SINGA with MKL-DNN support:

```
# Dependent libs are installed already
$ cmake -DUSE_MKLDNN=ON ..
$ make
```

PACKAGE

This setting is used to build the Debian package. Set PACKAGE=ON and build the package with make command like this:

```
$ cmake -DPACKAGE=ON
$ make package
```

FAQ

- Q: Error from 'import singa' using PySINGA installed from wheel.

  A: Please check the detailed error from `python -c "from singa import _singa_wrap"`. Sometimes it is caused by the dependent libraries, e.g. there are multiple versions of protobuf, missing of cudnn, numpy version mismatch. Following steps show the solutions for different cases

  1. Check the cudnn and cuda and gcc versions, cudnn5 and cuda7.5 and gcc4.8/4.9 are preferred. if gcc is 5.0, then downgrade it. If cudnn is missing or not match with the wheel version, you can download the correct version of cudnn into ~/local/cudnn/ and

     ```
     $ echo "export LD_LIBRARY_PATH=/home/<yourname>/local/cudnn/lib64:$LD_LIBRARY_PATH" >> ~/.bashrc
     ```

  2. If it is the problem related to protobuf, then download the newest whl files which have compiled protobuf and openblas into the whl file of PySINGA. Or you can install protobuf from source into a local folder, say ~/local/; Decompress the tar file, and then

     ```
     $ ./configure --prefix=/home/<yourname>local
     $ make && make install
     $ echo "export LD_LIBRARY_PATH=/home/<yourname>/local/lib:$LD_LIBRARY_PATH" >> ~/.bashrc
     $ source ~/.bashrc
     ```

  3. If it cannot find other libs including python, then create virtual env using pip or conda;
  4. If it is not caused by the above reasons, go to the folder of `_singa_wrap.so`,

     ```
     $ python
     >> import importlib
     >> importlib.import_module('_singa_wrap')
     ```

  ```
  Check the error message. For example, if the numpy version mismatches, the error message would be,

        RuntimeError: module compiled against API version 0xb but this version of numpy is 0xa

  Then you need to upgrade the numpy.
  ```

- Q: Error from running `cmake ..`, which cannot find the dependent libraries.

A: If you haven't installed the libraries, install them. If you installed the libraries in a folder that is outside of the system folder, e.g. /usr/local, you need to export the following variables

```
$ export CMAKE_INCLUDE_PATH=<path to your header file folder>
$ export CMAKE_LIBRARY_PATH=<path to your lib file folder>
```

- Q: Error from `make`, e.g. the linking phase

  A: If your libraries are in other folders than system default paths, you need to export the following varaibles

  ```
  $ export LIBRARY_PATH=<path to your lib file folder>
  $ export LD_LIBRARY_PATH=<path to your lib file folder>
  ```

- Q: Error from header files, e.g. 'cblas.h no such file or directory exists'

  A: You need to include the folder of the cblas.h into CPLUS_INCLUDE_PATH, e.g.,

  ```
  $ export CPLUS_INCLUDE_PATH=/opt/OpenBLAS/include:$CPLUS_INCLUDE_PATH
  ```

- Q:While compiling SINGA, I get error `SSE2 instruction set not enabled`

  A:You can try following command:

  ```
  $ make CFLAGS='-msse2' CXXFLAGS='-msse2'
  ```

- Q:I get `ImportError: cannot import name enum_type_wrapper` from google.protobuf.internal when I try to import .py files.

  A: You need to install the python binding of protobuf, which could be installed via

  ```
  $ sudo apt-get install protobuf
  ```

  or from source

  ```
  $ cd /PROTOBUF/SOURCE/FOLDER
  $ cd python
  $ python setup.py build
  $ python setup.py install
  ```

- Q: When I build OpenBLAS from source, I am told that I need a Fortran compiler.

  A: You can compile OpenBLAS by

  ```
  $ make ONLY_CBLAS=1
  ```

  or install it using

  ```
  $ sudo apt-get install libopenblas-dev
  ```

- Q: When I build protocol buffer, it reports that GLIBC++_3.4.20 not found in /usr/lib64/libstdc++.so.6.

  A: This means the linker found libstdc++.so.6 but that library belongs to an older version of GCC than was used to compile and link the program. The program depends on code defined in the newer libstdc++ that belongs to the newer version of GCC, so the linker must be told how to find the newer libstdc++ shared library. The simplest way to fix this is to find the correct libstdc++ and export it to LD_LIBRARY_PATH. For example, if GLIBC++_3.4.20 is listed in the output of the following command,

  ```
  $ strings /usr/local/lib64/libstdc++.so.6|grep GLIBC++
  ```

  then you just set your environment variable as

  ```
  $ export LD_LIBRARY_PATH=/usr/local/lib64:$LD_LIBRARY_PATH
  ```

- Q: When I build glog, it reports that "src/logging_unittest.cc:83:20: error: 'gflags' is not a namespace-name"

  A: It maybe that you have installed gflags with a different namespace such as "google". so glog can't find 'gflags' namespace. Because it is not necessary to have gflags to build glog. So you can change the configure.ac file to ignore gflags.

  ```
  1. cd to glog src directory
  2. change line 125 of configure.ac  to "AC_CHECK_LIB(gflags, main, ac_cv_have_libgflags=0,
ac_cv_have_libgflags=0)"
  3. autoreconf
  ```

  After this, you can build glog again.

- Q: When using virtual environment, everytime I run pip install, it would reinstall numpy. However, the numpy would not be used when I `import numpy`

  A: It could be caused by the `PYTHONPATH` which should be set to empty when you are using virtual environment to avoid the conflicts with the path of the virtual environment.

- Q: When compiling PySINGA from source, there is a compilation error due to the missing of <numpy/objectarray.h>

  A: Please install numpy and export the path of numpy header files as

  ```
  $ export CPLUS_INCLUDE_PATH=`python -c "import numpy; print numpy.get_include()"`:$CPLUS_INCLUDE_PATH
  ```

- Q: When I run PySINGA in Mac OS X, I got the error “Fatal Python error: PyThreadState_Get: no current thread Abort trap: 6”

  A: This error happens typically when you have multiple version of Python on your system and you installed SINGA via pip (this problem is resolved for installation via conda), e.g, the one comes with the OS and the one installed by Homebrew. The Python linked by PySINGA must be the same as the Python interpreter. You can check your interpreter by `which python` and check the Python linked by PySINGA via `otool -L <path to _singa_wrap.so>`. To fix this error, compile SINGA with the correct version of Python. In particular, if you build PySINGA from source, you need to specify the paths when invoking cmake

  ```
  $ cmake -DPYTHON_LIBRARY=`python-config --prefix`/lib/libpython2.7.dylib -DPYTHON_INCLUDE_DIR=`python-config --prefix`/include/python2.7/ ..
  ```

  If installed PySINGA from binary packages, e.g. debian or wheel, then you need to change the python interpreter, e.g., reset the $PATH to put the correct path of Python at the front position.