ScaleGraph/Python Install

ScaleGraph/Pythonのインストール環境およびインストール手順について説明します。

Setup Environment

下記のコンパイラ、ミドルウェア、ライブラリを事前にインストールする必要があります。

> Note: Mac OS Xで試験する場合、下記のドキュメント内で設定されている LD_LIBRARY_PATHの値はDYLD_LIBRARY_PATHに設定します。

GCC 4.8+

MPICH 3.1.4

```
$ export PATH=/path/to/install/mpich/bin:$PATH
$ export
LD_LIBRARY_PATH=/path/to/install/mpich/lib:$LD_LIBRARY_PATH
$ export
LIBRARY_PATH=/path/to/install/mh/mpich/lib:$LIBRARY_PATH
```

Hadoop 2.7.1

```
$ mvn package -Pdist,native,docs -DskipTests -Dtar
$ cd /path/to/install/hadoop
$ tar xvfz /path/to/project/hadoop/hadoop-
dist/target/hadoop-2.7.1.tar.gz
$ export
PATH=/path/to/install/hadoop/bin:/usr/local/mh/hadoop/sbin:$PATH
$ export C_INCLUDE_PATH=/path/to/install/hadoop/include
$ export CPLUS_INCLUDE_PATH=/path/to/install/hadoop/include
$ export
LD_LIBRARY_PATH=/path/to/install/hadoop/lib/native:$LD_LIBRARY_PA
TH
$ export
LIBRARY_PATH=/path/to/install/hadoop/lib/native:$LIBRARY_PATH
$ export HADOOP_CONF_DIR=/path/to/install/hadoop/etc/hadoop
$ export HADOOP_PREFIX=/path/to/install/hadoop
$ export CLASSPATH=`hdfs classpath --glob`
```

Edit core-site.xml

以下はkfc025を接続先にする場合

```
<property>
        <name>fs.defaultFS</name>
        <!-- <value>hdfs://localhost:9000</value> -->
        <value>hdfs://kfc025-ib:9000</value>
</property>
```

Edit hdfs-site.xml

```
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
<property>
        <name>dfs.hosts</name>

<value>/path/to/install/hadoop/etc/hadoop/hosts.txt</value>
</property>
<property>
        <name>dfs.hosts.exclude</name>
        <value>/path/to/install/hadoop/etc/hadoop/hosts-
exclude.txt</value>
</property>
```

Edit hosts configuration

以下のファイル群は利用するノードによって設定を変更します。

```
$ cd /path/to/install/hadoop/etc
$ touch hosts-exclude.txt
$ cat > slaves
localhost
kfc026-ib
kfc027-ib
kfc028-ib
$ cat > hosts.txt
kfc025-ib
kfc026-ib
kfc027-ib
```

```
kfc028-ib
```

Clean old data

なんらかの事情で破損したHDFS Dataがディスク上に残っている場合、HDFSの起動時にエラーが出ます。そのデータをサルベージする必要がなければ消去します。

```
$ rm -rf /tmp/hadoop-username #各ノードで実行
```

Start HDFS and Create home directory

HDFS利用時には常にデーモンを動かしておく必要があります。

```
$ hdfs namenode -format
$ start-dfs.sh
$ hdfs dfs -mkdir /user
$ hdfs dfs -mkdir /user/username
```

Stop HDFS

以下のコマンドでHDFSのデーモンを終了します。

```
$ stop-dfs.sh
```

Test libhdfs.a

HDFSが利用できることを確認します。

> Note: 下記のコードは以下のサイトからコピーしました。
> https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/LibHdfs.html

```
$ cat > above_sample.c
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include "hdfs.h"

int main(int argc, char **argv) {

    hdfsFS fs = hdfsConnect("default", 0);
    const char* writePath = "/tmp/testfile.txt";
    hdfsFile writeFile = hdfsOpenFile(fs, writePath, O_WRONLY
|O_CREAT, 0, 0, 0);
    if(!writeFile) {
          fprintf(stderr, "Failed to open %s for writing!\n",
writePath);
          exit(-1);
    }
    char* buffer = "Hello, World!";
    tSize num_written_bytes = hdfsWrite(fs, writeFile,
(void*)buffer, strlen(buffer)+1);
    if (hdfsFlush(fs, writeFile)) {
           fprintf(stderr, "Failed to 'flush' %s\n", writePath);
          exit(-1);
    }
    hdfsCloseFile(fs, writeFile);
}
$ gcc above_sample.c -lhdfs -o above_sample
$ hadoop classpath --glob > classpaths
$ CLASSPATH=`cat classpaths` ./above_sample
$ hdfs dfs -ls /tmp
Found 1 items
-rw-r--r--   1 username supergroup         14 2015-10-21 06:07
/tmp/testfile.txt
$ hdfs dfs -cat /tmp/testfile.txt
Hello, World!
```

Python 3.5.0+

Python 3系の利用を前提として開発をしているため、Python 2系は利用できません。

Cloudpickle

Pythonで利用するプラグインモジュールです。

```
$ pip3 install cloudpickle
```

SX10

```
$ git clone https://github.com/scalegraph/sx10.git SX10
$ cd SX10
$ git checkout develop
$ ant dist -DGCC_SYMBOLS=true -DX10RT_MPI=true
-Davailable.procs=4
$ export PATH=/path/to/install/sx10/x10.dist/bin:$PATH
```

Original ScaleGraph

```
$ git clone https://github.com/scalegraph/scalegraph.git
ScaleGraph
$ cd ScaleGraph
$ git checkout noemm
$ make
```

Install ScaleGraph/Python

Extract source code of ScaleGraph/Python

ScaleGraph/Pythonのソースを展開します。本ドキュメントにおいては展開先を/path/to/project/of/scalegraph-devであると仮定します。

```
$ git clone https://path/to/scalegraph-dev.git scalegraph-dev
$ cd scalegraph-dev
$ git checkout feature_pythonapi/release
$ pwd
```

```
/path/to/project/of/scalegraph-dev
```

Copy static link libraries from ScaleGraph build

Original ScaleGraphのビルド時にコンパイルされているARPACKライブラリを持ってきます。

```
$ cp /path/to/scalegraph/build/lib/libarpack.a
/path/to/project/of/scalegraph-dev/lib
$ cp /path/to/scalegraph/build/lib/libparpack.a
/path/to/project/of/scalegraph-dev/lib
```

Edit configuration

Hard codingされているpathをコンパイル・インストール時の環境のものに書き換えます。

```
./cpp/pyxpregelworker/main.cc:27:
PyRun_SimpleString("sys.path.append('/path/to/project/of/scalegrap
h-dev/src/python/scalegraph')");
./cpp/pyxpregelworker/Makefile:2:INCLUDE := -
I/path/to/source/of/sx10/x10.runtime/src-cpp
./org/scalegraph/api/NativePyXPregelAdapter.cc:159:
execl("/path/to/project/of/scalegraph-
dev/src/cpp/pyxpregelworker/pyxpregelworker",
./org/scalegraph/api/PyXPregel.x10:56:
python.sysPathAppend("/path/to/project/of/scalegraph-
dev/src/python/scalegraph");
./org/scalegraph/api/PyXPregel.x10:84:          val path =
FilePath(FilePath.FILEPATH_FS_OS,
"/path/to/project/of/scalegraph-dev/build/" +
"_xpregel_closure.bin");
./org/scalegraph/xpregel/PyWorkerPlaceGraph.x10:135:
val SCALEGRAPHPYTHONLIB :String =
"/path/to/project/of/scalegraph-dev/src/python/scalegraph";
./python/scalegraph/config.py:14:work_dir =
'/path/to/project/of/scalegraph-dev/build/'
./python/scalegraph/config.py:15:apidriver_path =
```

```
'/path/to/project/of/scalegraph-dev/build/apidriver'
```

> Note: Hard codingされているpathは、今後の修正で一箇所の設定ファイルにまとめる予定です。

Edit runtime configuration

バックエンドのScaleGraphをフロントエンドのPython moduleから呼び出すためのMPI設定をsrc/python/scalegraph/config.pyに記述します。

```
mpirun_path = 'mpirun'
mpirun_opts = ['-np', '4']
```

mpirun_pathのところにsshを挟んだmpirun呼び出しを書くことで、リモートにあるクラスタをScaleGraphバックエンドとして使うことができるようになります。

> Note: 上記の設定はpython関数の引数や環境変数で実行時に変更できるようにすべきだと考えています。今後の修正で、そのようにする予定です。
>
> Note: この設定は現状、Graph Algorithm API関数呼び出しにのみ有効です。XPregel APIの利用時には関係ありません。今後はXPregel APIをGraph Algorithm APIと統合する予定です。

Build Graph Algrithm API Runtime

Graph Algorighm APIをPythonから利用するために使う、ScaleGraphバックエンドアプリケーションをコンパイルします。

```
$ x10c++ -v -x10rt mpi -sourcepath src -cxx-prearg -Iinclude -cxx-postarg "-Llib -lhdfs -lparpack -larpack -lgfortran -lmpifort" -cxx-prearg "`python3-config --cflags`" -cxx-postarg "`python3-config --ldflags`" -g -o build/apidriver -make -make-arg -j src/org/scalegraph/api/ApiDriver.x10
```

Build Pregel Model Computation API Runtime

Pregel Model Computation APIをPythonから利用するために使う、ScaleGraphバックエンドアプリケーションをコンパイルします。

```
$ x10c++ -v -x10rt mpi -sourcepath src -cxx-prearg -Iinclude -cxx-postarg "-Llib -lhdfs -lparpack -larpack -lgfortran -lmpifort" -cxx-prearg "`python3-config --cflags`" -cxx-postarg "`python3-config --ldflags`" -g -o testpyxpregel -make -make-arg -j src/test/TestPyXPregel.x10
```

Test ScaleGraph/Python

Test Graph Algorithm API

ユニットテストの実行方法は以下の通りです。

```
$ cd /path/to/project/scalegraph-dev/src/python/scalegraph
$ python3 testalgorithm.py
test_ErrorInvalidExtraOptions (__main__.TestGenerateGraph) ... ok
test_ErrorInvalidInput (__main__.TestGenerateGraph) ... ok
test_ErrorInvalidInputFs (__main__.TestGenerateGraph) ... ok
test_ErrorInvalidOutputFs (__main__.TestGenerateGraph) ... ok
test_ErrorInvalidRmatScale (__main__.TestGenerateGraph) ... ok
test_ErrorNoInput (__main__.TestGenerateGraph) ... ok
以下略
```

Test Pregel Model Computation API

以下のPageRankアルゴリズムのサンプルコードを例に実行手順を以下に示します。

```
$ cd /path/to/project/scalegraph-dev/src/python/scalegraph
```

```
$ cat pagerank.py # ソース内にサンプルコードがあります。
#コメント略
import xpregel

class PageRank(xpregel.XPregelBase):

    def compute(self, ctx, messages):
        if ctx.superstep == 0:
            value = 1.0 / ctx.numVertices
        else:
            value = 0.15 / ctx.numVertices + 0.85 * sum(messages)
        ctx.aggregate(abs(value - ctx.getVertexValue()))
        ctx.setVertexValue(value)
        numOutEdges = len(ctx.outEdges)
        if numOutEdges > 0:
            msg = value / numOutEdges
            for d in ctx.outEdges:
                ctx.sendMessage(d, msg)
#         if numOutEdges > 0:
#             ctx.sendMessageToAllNeighbors(value / numOutEdges)


    def aggregator(self, outputs):
        return sum(outputs)

    def terminator(self, superstep, aggregatedValue):
#         ctx.log("PageRank at superstep " + superstep + " = " +
aggValue + "\n")
        return superstep == 30


if __name__ == '__main__':
    pr = PageRank()
    pr.run()
```

クロージャのシリアライズを行います。config.pyのwork_dir変数で設定されているパスにシリアライズ結果である_xpregel_closure.binが吐かれます。このファイルはScaleGraphバックエンドを実行するクラスタのファイルシステムに存在する必要があります。
シリアライズを行う処理は、任意のフロントエンド（クラスタとは関係ないマシン）で実行することができます。

```
$ python3 pagerank.py
```

> Note: クロージャのシリアライズから、アルゴリズムの実行までの一連の流れは、フロントエンドのpythonランタイムから自動的にバックエンドまで引き継がれるべきです。改修によって対応予定です。

下記のコマンドをScaleGraphバックエンドを実行するクラスタで実行します。上記のシリアライズ結果が読み込まれ、Pythonインタプリタで処理されます。

```
$ X10_NTHREADS=2 mpirun -np 2 ./testpyxpregel
```

上記の実行において、利用されるグラフデータはプログラム内で自動生成されています。グラフデータの設定はsrc/org/scalegraph/api/PyXPregel.x10の下記のコードです。

```
    public def test_xpregel() {

        val loadedClosures = loadClosures();
        shareClosures(loadedClosures);

        val graph = makeGraphFromRmat();
        val matrix = graph.createDistSparseMatrix[Double](
            Config.get().distXPregel(), "weight", true, false);
        graph.del();

        val xp = this;
        val result = xp.execute(matrix);
//      val result = xp.execute_x10xpregel(matrix);

//      CSV.write(getFilePathOutput(), new
NamedDistData(["pagerank" as String], [result as Any]), true);
    }
```