実践 機械学習システム
第 12 章 ビッグデータ

山木翔馬

July 22, 2016

本章の目的

ビッグデータとは

- 処理パワーよりもデータの増加するスピードが早い
- これまでうまく行った手法や技術がデータサイズの増加に対応できない
- データ全体がメモリに入りきらない場合，現状のアルゴリズムで対応出来ない
- データ管理自体が主要タスクになる
- コンピュータクラスタやマルチコアが必要不可欠

本章前半では マルチコアを使って計算を高速化する構成方法 について，後半では jug を用いたオンラインクラスタの作成とジョブの振り分けについて焦点を当てる．

jug

- タスクベースの並行処理フレームワーク
- 一時的な結果をディスクやデータベースに保存したり，必要なときに読み込んだり
- マルチコアもしくはクラスタ上の複数台のコンピュータを利用できる

jug のインストール

```
$ sudo pip install jug
```

タスクについて

ここでのタスクとは jug で行う処理の構成要素を指し，関数のこと．

jugfile01.py

```
def double(x):
    return 2 * x

# jug を用いてタスクを分割すると

from jug import Task
t1 = Task(double, 3)
t2 = Task(double, 642.34)
```

普通の Python ファイルとして保存する．

タスクの実行

jugfile01.py の実行

```
$ jug-execute jugfile01.py
    Executed    Loaded    Task name
-----------------------------------------------
           2         0    jugfile01.double
-----------------------------------------------
           2         0    Total
```

ここで作成される jugfile01.jugdata というディレクトリがキャッシュ用のファイル.

2 回目の実行

```
$ jug-execute jugfile01.py
    Executed    Loaded    Task name
-----------------------------------------------
           0         2    jugfile01.double
-----------------------------------------------
           0         2    Total
```

デコレータを用いた記述 (1/2)

Python のデコレータ機能を使えばタスクを簡潔に記述できる.

jugfile02.py (1/2)

```
from jug import TaskGenerator
from time import sleep

@TaskGenerator
def double(x):
    sleep(4)
    return 2*x

@TaskGenerator
def add(a, b):
    return a+b

@TaskGenerator
def print_final_result(oname, value):
    with open(oname, 'w') as output:
        print >>output, "Final result:", value
```

デコレータを用いた記述 (2/2)

jugfile02.py (2/2)

```
y = double(2)
z = double(y)

y2 = double(7)
z2 = double(y2)
print_final_result('output.txt', add(z, z2))
```

実行結果

```
$ jug-execute jugfile02.py
    Executed      Loaded  Task name
-------------------------------------------------------
           1           0  jugfile02.add
           4           0  jugfile02.double
           1           0  jugfile02.print_final_result
-------------------------------------------------------
           6           0  Total
$ cat output.txt
Final result: 36
```

jug status でステータスを確認

```
$ jug status jugfile02.py
     Waiting    Ready    Finished     Running  Task name
-------------------------------------------------------------------
           1        0           0           0  jugfile02.add
           2        2           0           0  jugfile02.double
           1        0           0           0  jugfile02.print_final_result
-------------------------------------------------------------------
           4        2           0           0  Total
```

プロセスを 2 つ同時に実行してみる.

```
$ jug-execute jugfile02.py &
$ jug-execute jugfile02.py &
$ jug status jugfile02.py
     Waiting    Ready    Finished    Running  Task name
-------------------------------------------------------------------
           1        0           0           0  jugfile02.add
           2        0           0           2  jugfile02.double
           1        0           0           0  jugfile02.print_final_result
-------------------------------------------------------------------
           4        0           0           2  Total
```

jug で部分的な結果を再利用する

たとえば新しい特徴量を追加したい場合など……

第 10 章で使ったコード

```
@TaskGenerator
def new_features(im):
    import mahotas as mh
    im = mh.imread(fname, as_grey=1)
    es = mh.sobel(im, just_filter=1)
    return np.array([np.dot(es.ravel(), es.ravel())])

hfeatures = as_array([hfeature(f) for f in filenames])
efeatures = as_array([new_features(f) for f in filenames])
features = Task(np.hstack, [hfeatures, efeatures])
# 学習コード ...
```

ここで古い特徴量はキャッシュから読み込まれるため，再度計算されず，計算の無駄を省くことができる．

内部で行われていること

```
for t in alltasks:
    if t.has_not_run() and not backend_has_value(t.hash()):
        value = t.execute()
        save_to_backend(value, key=t.hash())
```

- あるタスクが他のタスクを引数に取れば，その 2 つのタスクには依存関係ができる
- 1 つ目のタスクの結果が準備されるまで，2 つ目のタスクは実行されない
- これにより 複数のプロセッサ上でタスクを実行できる
- ネットワークディスクや Redis データベースなどを用いれば複数台のマシン上でも実行できる

AWS (Amazon Web Service)を利用する(しません)

- Amazon が提供している処理能力の高い環境(をレンタルできる)
- 参考書では Elastic Compute Cluster (EC2) を使用
- 仮想マシンとディスクスペースを提供している
- stracluster パッケージを用いればローカルマシンからクラスタを生成できる
- 前払い制，単位時間あたりのレート固定モード，全体の状況に応じてレートを変動するモードがある
- GPU 搭載したマシンなども提供している

お金かかるので利用しません.
* 近日中に nlken サーバに GPU 載せるのでそっち使えば OK.

jug を複数台マシンで実行する

たとえば nlken サーバと自分のマシンで jug を実行する場合，
nlken の redis-server を共有バックエンドとして用いる．

サーバ側

```
nlken: $ jug-execute jugfile.py --jugdir=redis://127.0.0.1
```

クライアント側

```
$ jug-execute jugfile.py --jugdir=redis://nlken.dse.ibaraki.ac.jp
```

サーバとクライアントで jug を実行する．キャッシュはサーバ側の redis データベースを共有する．

nlken サーバには redis-server を入れてるが，動かしてない．
redis-server のポートも開放してない．

まとめ

- jug：並列処理のためのパッケージ
- 分割処理したいタスク（関数）を @TaskGenerator でデコレート
- jug ファイルの実行は `jug-execute jugfile.py`
- ステータスの確認は `jug status jugfile.py`
- 複数台のマシンによる実行は，データベースサーバなどを共有バックエンド（キャッシュファイル）として利用する
- キャッシュファイルの指定は
  `jug-execute jugfile.py --jugdir=filepath`