Parallel Computing di Python dengan PYRO

Amru Rosyada
amrupunya@gmail.com
http://amrurosyada.blogspot.com
http://technocratiz.wordpress.com

Parallel Computing merupakan teknologi yang sudah tidak baru lagi di dunia IT, kebanyakan orang sering meyebutnya sebagai *Distributed Computing*, dengan adanya *distributed computing* maka beban proses yang berat dalam perhitungan akan dapat dibagi-bagi menjadi proses proses yang lebih kecil kemudian proses-proses tersebut akan didistribusikan ke komputer-komputer yang lain atau kita kenal dengan istilah *clustering*.

Jadi dengan permodelan clustering kita dapat memanggil method yang ada di komputer lain secara *remote,* kemudian setelah di proses pada masing-masing komputer hasilnya akan di kembalikan di monputer induk/master untuk kemudian diproses menjadi suatu bagian.

Pendahuluan

Apakah PYRO ?

PYRO adalah kependekan dari PYthon Remote Object. PYRO merupakan sistem teknologi distribusi objek yang powerful, yang didesain mudah untuk digunakan. Tak perlu khawatir tentang pembuatan kode-kode komunikasi jaringan, ketika menggunakan PYRO kita tinggal menulis kode-kode program seperti membuat objek seperti halnya pembuatan program python biasa. Dengan sedikit kode ekstra, PYRO akan menangani komunikasi jaringan antara objek-objek yang terpisah dalam mesin-mesin komputer yang berbeda. Semua detail dari socket programming akan ditangani PYRO, kita tinggal memanggil method pada remote objek sebagaimana kita memanggil local objek.

PYRO terdiri dari bentukan objek oriented RPC (Remote Procedure Call). Kita dapat menggunakan PYRO dengan sistem tunggal tetapi juga digunakan untuk IPC. PYRO menyerupai RMI (Remote Procedure Call) pada Java. PYRO lebih mirip dengan CORBA (Common Object Request Broker Architecture) yang mana sistem dan bahasa teknologi objek distribusi berdiri sendiri dan lebih lengkap jika dibanding PYRO atau RMI. Tetapi PYRO simpel, kecil, fun dan free. Untuk mendapatkan dokumentasi tentang PYRO dan download paket bisa diperoleh dari http://pyro.sourceforge.net/.

Parallel Computing dengan PYRO

PYRO teknologi :

1. PYRO protocol didasarkan atas TCP/IP menggunakan pickle
2. Dapat digunakan pada jaringan yang heterogen (LAN. WAN), kluster begitu juga untuk mesin multiprosessor.
3. Server dan client dapat saling bertukar aturan (callback)
4. Penggunaannya seperti python biasa
5. Konfigurasi yang mudah

Prinsip dari PYRO :

PYRO menggunakan name server (NS) untuk menemukan remote objek:

- Remote objek harus diregister menggunakan NS
- Client mencari NS remote objek untuk mendapatkan URI remote objek (seperti halnya WWW URL)

Client menggunakan proxy untuk memforward method call ke remote objek. Terdapat tiga jenis proxi:

- Static proxy
- Dinamic proxy dan
- Dinamic proxy dengan atribut akses suport

Contoh simpel :

Contoh dari distribusi yang mendemonstrasikan prinsip dari PYRO.

- tst.py : Remote kode yang akan melakukan komputasi/perhitungan.
- server.py : Server yang mengijinkan mengeksekusi ke tst.py
- client.py : Client yang memanggil method dari tst.py secara remote

```
# tst.py
# class ini yang akan diakses secara remote.
class testclass:
    def mul(s, arg1, arg2): return arg1*arg2
    def add(s, arg1, arg2): return arg1+arg2
    def sub(s, arg1, arg2): return arg1-arg2
    def div(s, arg1, arg2): return arg1/arg2
    def error(s):
        x=foo()
        x.crash()
```

```
class foo:
   def crash(s):
      s.crash2('going down...')
   def crash2(s, arg):
      # statemen ini sengaja untuk mencoba jika program terjadi kesalahan:
      x=arg/2

# server.py
# class yang mengijinkan untuk mengakses tst.py
import Pyro.core
import Pyro.naming
import tst

class testclass(Pyro.core.ObjBase, tst.testclass):
        def __init__(self):
                Pyro.core.ObjBase.__init__(self)
                Pyro.core.initServer()

ns=Pyro.naming.NameServerLocator().getNS()
daemon=Pyro.core.Daemon()
daemon.useNameServer(ns)
uri=daemon.connect(testclass(),"simple")
print "Server is ready."
daemon.requestLoop()

# client.py
import Pyro.util
import Pyro.core
Pyro.core.initClient()
test = Pyro.core.getProxyForURI("PYRONAME://simple")
print test.mul(111,9)
print test.add(100,222)
print test.sub(222,100)
print test.div(2.0,9.0)
print test.mul('.',10)
print test.add('String1','String2')
print '*** invoking server method that crashes ***'
print test.error()
```

*Note : Dapat dijalankan pada satu komputer secara otomatis akan bounding ke localhost

Menjalankan program dengan cara :

1. Jalankan PYRO name server dahulu pada komputer 1 dengan menjalankan perintah pada console/command prompt "pyro-ns"

```
amru@icEcubE:~$ pyro-ns
*** Pyro Name Server ***
Pyro Server Initialized. Using Pyro V3.7
URI is: PYRO://169.254.10.233:9090/a9fe0ae9017c2f9e44b865150e28f3da
URI written to: ...\Pyro-3.4\examples\simple\Pyro_NS_URI
Name Server started.
```

2. Jalankan server pada komputer 2 dengan menjalankan :

```
amru@icEcubE:~/Documents/distributed$ python server.py
Pyro Client Initialized. Using Pyro V3.7
Pyro Server Initialized. Using Pyro V3.7
Server is ready.
```

3. Jalankan client pada komputer 3 dengan menjalankan :

```
amru@icEcubE:~/Documents/distributed$ python client.py
Pyro Client Initialized. Using Pyro V3.7
999
322
122
0.222222222222
.........
String1String2
*** invoking server method that crashes ***
Traceback (most recent call last):
  File "client.py", line 12, in <module>
    print test.error()
```

```
  File "/usr/lib/python2.5/site-packages/Pyro/core.py", line 390, in __call__
    return self.__send(self.__name, args, kwargs)
  File "/usr/lib/python2.5/site-packages/Pyro/core.py", line 468, in _invokePYRO
    return self.adapter.remoteInvocation(name, constants.RIF_VarargsAndKeywords,
vargs, kargs)
        File "/usr/lib/python2.5/site-packages/Pyro/protocol.py", line 491, in
remoteInvocation
    answer.raiseEx()
  File "/usr/lib/python2.5/site-packages/Pyro/errors.py", line 81, in raiseEx
    raise self.excObj
TypeError: ("unsupported operand type(s) for /: 'str' and 'int'", 'This error occured
remotely (Pyro). Remote traceback is available.')
```

Contoh clustering menggunakan PYRO :

Contoh program dibawah ini akan mendemonstrasikan clustering menggunakan PYRO.

 # calculator.py

```
"""
Very simple calculation to demonstrate the use pyro on a cluster.
"""
import Numeric
class Calculator:
  def setParameters(self, offset, multiplyer):
    """
    __init__ can not be called remotely.
    Use other method instead.
    """
    self.offset = offset
    self.multiplyer = multiplyer
  def calculate(self, numericArray, t):
    result = (self.offset + numericArray * self.multiplyer) * t
    return result
```

 # clusterServer.py

```
import Pyro.core
import Pyro.naming
```

```
import calculator
import sys
class CalculatorProxy(Pyro.core.ObjBase, calculator.Calculator):
    def __init__(self):
        Pyro.core.ObjBase.__init__(self)
try:
    hostname = sys.argv[1]
except IndexError:
    print 'Please give hostname as command line argument.'
    print 'It will be used to register the calculator with the NS.'
    print 'Example: hostname:node1, object_name:calculator_node1'
Pyro.core.initServer()
ns=Pyro.naming.NameServerLocator().getNS()
daemon=Pyro.core.Daemon()
daemon.useNameServer(ns)
uri=daemon.connect(CalculatorProxy(),'calculator_%s' %hostname)
print 'Server with object calcualtor_%s is ready.' %hostname
daemon.requestLoop()
```

 # clusterClient.py

```
import Pyro.util
import Pyro.core
import Numeric
import threading
class simpleSimulation:
    def __init__(self, offset, multiplyer, arraySize,
                 deltaT, steps, nodes):
        self.deltaT = deltaT
        self.steps = steps
        self.nodes = nodes
        self.offset = offset
        self.multiplyer = multiplyer
        self.numbers = Numeric.array(range(arraySize), Numeric.Float)
        self.results = []
        self.bounds = self.findArrayBounds(self.numbers.shape[0],
                                           len(self.nodes))
        self.connectToNodes()
        self.t = 0
```

```
    def findArrayBounds(self, arrayLength, numberOfThreads):
        """Split array in nearly equal sized parts.
            Maximum difference between longest and shortest part is 1.
        """
        if numberOfThreads > arrayLength:
            numberOfThreads = arrayLength
        minArrayLength, extra = divmod(arrayLength, numberOfThreads)
        bounds = []
        lower = 0
        oneMore = 0
        for thread in range(numberOfThreads):
            if extra:
                oneMore = 1
            else:
                oneMore = 0
            upper = lower + minArrayLength + oneMore
            bounds.append((lower, upper))
            extra = max(0, extra - 1)
            lower = upper
        return bounds

    def connectToNodes(self):
        Pyro.core.initClient()
        self.pyroNodes = []
        for node in self.nodes:
                self.pyroNodes.append(Pyro.core.getProxyForURI('PYRONAME://calculator_%s'
%node)

    def runSimulation(self):
        for step in range(self.steps):
            n = 0
            result = Numeric.zeros(self.numbers.shape, Numeric.Float)
            threads = []
            for pyroNode in self.pyroNodes:
                threads.append(CalculatorThread(pyroNode, self.offset,
                        self.multiplyer))
                lower = self.bounds[n][0]
```

```
            upper = self.bounds[n][1]
            threads[n].numericArray = self.numbers[lower:upper]
            threads[n].t = self.t
            threads[n].start()
            n += 1
        for thread in threads:
            thread.join()
        n = 0
        for thread in threads:
            lower = self.bounds[n][0]
            upper = self.bounds[n][1]
            result[lower:upper] = thread.result
            n += 1
        self.results.append(result)
        self.t += self.deltaT

class CalculatorThread(threading.Thread):
    def __init__(self, remoteCalculator, offset, multiplyer):
        threading.Thread.__init__(self)
        self.remoteCalculator = remoteCalculator
        self.remoteCalculator.setParameters(offset, multiplyer)
    def run(self, numericArray, t):
        return self.remoteCalculator.calculate(numericArray, t)

if __name__ == '__main__':
    offset = 3
    multiplyer = 5
    arraySize = 7
    deltaT = 0.5
    steps = 5
    s = simpleSimulation(offset, multiplyer, arraySize,
                         deltaT, steps, ['node1', 'node2'])
    s.runSimulation()
    # check results
    t = 0
    for pyroResult in s.results:
        localResult = (offset + Numeric.array(range(arraySize),
                    Numeric.Float) * multiplyer) * t
```

```
        t += deltaT
        print 'pyro results:'
        print pyroResult
        print 'local results:'
        print localResult
```

Menjalankan kalkulasi menggunakan clustering :

1. Jalankan pyro-ns

```
amru@icEcubE:~$ pyro-ns
*** Pyro Name Server ***
Pyro Server Initialized. Using Pyro V3.7
Name server listening on: ('0.0.0.0', 9090)

WARNING: daemon bound on hostname that resolves to loopback address 127.0.x.x

URI is: PYRO://127.0.1.1:9090/7f00010135b173e35414859c960aedf1
URI written to: /home/amru/Pyro_NS_URI
Name Server started.
```

2. Jalankan server pada node 1

```
amru@icEcubE:~/Documents/distributed$ python clusterServer.py node1
Pyro Server Initialized. Using Pyro V3.7
Server with object calcualtor_node1 is ready.
```

3. Jalankan server pada node 2

```
amru@icEcubE:~/Documents/distributed$ python clusterServer.py node2
Pyro Server Initialized. Using Pyro V3.7
Server with object calcualtor_node1 is ready.
```

4. Jalankan client

```
amru@icEcubE:~/Documents/distributed$ python clusterclient.py
Pyro Client Initialized. Using Pyro V3.4
```

```
pyro results:
[ 0. 0. 0. 0. 0. 0. 0.]
local results:
[ 0. 0. 0. 0. 0. 0. 0.]
pyro results:
[ 1.5  4.  6.5  9.  11.5 14.  16.5]
local results:
[ 1.5  4.  6.5  9.  11.5 14.  16.5]
...
pyro results:
[ 6. 16. 26. 36. 46. 56. 66.]
local results:
[ 6. 16. 26. 36. 46. 56. 66.]
```

Scurity Note :

1. Pickle sangatlah tidak aman
2. Gunakan XML disamping pickle
3. Gunakan SSL
4. Jangan gunakan mobile code, kalaupun akan digunakan pakailah validator
5. Gunakan validator yang sudah dicustom
6. Batasi jumlah koneksi
7. Izinkan akses dari IP address yang dikenali saja
8. Chek passphrase

Penutup

Diharapkan dengan adanya artikel ini bisa membantu dalam meningkatkan kemajuan teknologi informasi di Indonesia dan mendukung suksesnya IGOS

Referensi

http://python.org
http://pyro.sourceforge.net
Tutorial High Performance Computing in Python
1. Treffen Leipzig Python User Group, 14.02.2006
Parallel Computing in Python with Pyro
Comfortable, Flexible, Transparent, Pythonic
Mike Müller, mmueller@python-academy.de
Python Academy Leipzig

Biografi Penulis

Amru Rosyada. Lahir pada tanggal 22 Mei 1986, menamatkan pendidikan dasar sampai pendidikan menengah akhir di kota Ngawi kemudian terdampar di Jogja mengambil program Diploma tiga Teknik Elektro Universitas Gadjah Mada dan Sekarang masih menamatkan Strata satu di Ilmukomputer Universitas Gadjah Mada.