

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



Overview

- What is Pyro?
- Pyro Principles
- Pyro Technology
- Simple Example
- Cluster Example
- Security
- Questions?

What is Pyro?

- Python Remote Objects, <http://pyro.sourceforge.net/>
- pure Python package that makes distributed computing (nearly) effortless
- author: Irmien de Jong, irmien@users.sourceforge.net
- MIT licences
- current version 3.4, API stable no new features planned for the next releases
- extensive Documentation



Pyro technology

- PYRO protocol based on TCP/IP using pickle
- can be used on heterogeneous networks (LAN, WAN), clusters ... even multiprocessor machines
- server and client can switch roles (callbacks)
- feels (mostly) just like ordinary Python
- plenty of configuration options



Principles of Pyro

Pyro uses a name server (NS) to find the remote objects :

- remote objects have to be registered with the NS
- clients query the NS for an object and get an URI (looks like a WWW URL)

Clients use proxies to forward their method calls to remote objects. There are three kinds of proxies:

- static proxies
- dynamic proxies and
- dynamic proxies with attribute access support



Simple Example

Example from the distribution that demonstrates principles.

- **tst.py:** Remote code that does computation.
- **server.py:** Server that allows exes to tst.py.
- **client.py:** Client that calls methods from tst.py remotely.

tst.py

```
# tst.py
# These classes will be remotely accessed.

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):
        # this statement will crash on purpose:
        x=arg/2
```

server.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()
```

Starting the calculation

On computer/shell 1 start the name server:

```
computer1$ ns
*** Pyro Name Server ***
Pyro Server Initialized. Using Pyro V3.4
URI is: PYRO://169.254.10.233:9090/a9fe0ae9017c2f9e44b865150e28f3da
URI written to: ...\\Pyro-3.4\\examples\\simple\\Pyro_NS_URI
Name Server started.
```

On computer/shell 2 start pyro server:

```
computer2$ server.py
Pyro Server Initialized. Using Pyro V3.4
Server is ready.
```

On computer/shell 3 call the remote object with the client:

```
computer3$ client.py
```

Results

```
...Pyro-3.4\examples\simple>client.py
Pyro Client Initialized. Using Pyro V3.4
999
322
122
0.222222222222
.....
String1String2
*** invoking server method that crashes ***
Traceback (most recent call last):
  File "...Pyro-3.4\examples\simple\client.py", line 18, in ?
    print test.error()
  File "...Pyro\core.py", line 444, in __invokePYRO__
    constants.RIF_VarargsAndKeywords, vargs, kargs)
  File "...Pyro\protocol.py", line 431, in remoteInvocation
    answer.raiseEx()
  File "...Pyro\errors.py", line 115, in raiseEx
    raise self.excObj
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```



Cluster Example

calculator.py

```
"""
Very simple calculation to demonstrate the use pyro on a cluster.
"""

import Numeric

class Calculator:
    def setParameters(self, offset, multiplier):
        """
        __init__ can not be called remotely.
        Use other method instead.
        """
        self.offset = offset
        self.multiplier = multiplier

    def calculate(self, numericArray, t):
        result = (self.offset + numericArray * self.multiplier) * 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 calcultor_%s is ready.' %hostname
daemon.requestLoop()
```

clusterClient.py

```
import Pyro.util
import Pyro.core
import Numeric
import threading

class simpleSimulation:
    def __init__(self, offset, multiplier, arraySize,
                 deltaT, steps, nodes):
        self.deltaT = deltaT
        self.steps = steps
        self.nodes = nodes
        self.offset = offset
        self.multiplier = multiplier
        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
```

PYTHON ACADEMY

DIE PROGRAMMIERSCHULE

PYRO

clusterClient.py II

```
class simpleSimulation:
```

```
...
```

```
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
```

PYTHON ACADEMY

DIE PROGRAMMIERSCHULE

PYRO

clusterClient.py III

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

PYTHON ACADEMY

DIE PROGRAMMIERSCHULE

PYRO

clusterClient.py IV

```
class simpleSimulation:
    ...
    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.multiplier))

                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
```

PYTHON ACADEMY

DIE PROGRAMMIERSCHULE

PYRO

clusterClient.py V

...

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

clusterClient.py VI

```
if __name__ == '__main__':
    offset = 3
    multiplier = 5
    arraySize = 7
    deltaT = 0.5
    steps = 5
    s = simpleSimulation(offset, multiplier, arraySize,
                        deltaT, steps, ['node1', 'node2'])
    s.runSimulation()

    # check results
    t = 0
    for pyroResult in s.results:
        localResult = (offset + Numeric.array(range(arraySize),
                                                Numeric.Float) * multiplier) * t

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

Starting the calculation

Start the name server:

```
ns_computer$ns
*** Pyro Name Server ***
Pyro Server Initialized. Using Pyro V3.4
URI is: PYRO://192.168.1.50:9090/c0a80132044c2fc732fbadf23b086a2e
URI written to: ...Python24\Scripts\Pyro_NS_URI
Name Server started.
```

On node 1 start pyro server 1:

```
node1$ clusterServer.py node1
Pyro Server Initialized. Using Pyro V3.4
'Server with object calcualtor_node1 is ready.'
```

On node 2 start pyro server 2:

```
node2$ clusterServer.py node2
Pyro Server Initialized. Using Pyro V3.4
'Server with object calcualtor_node2 is ready.'
```

Result

On client computer call the remote object with the client:

```
client_computer$ 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.]
```

Security

- pickle is potentially insecure
- use XML instead of pickle, order of magnitude slower
- SSL is available
- don't allow mobile code
- use a code validator for mobile code
- write custom connection validator
- limit number of connections
- allow access from certain IP numbers only
- check passphrase

Questions?

PYTHON ACADEMY

DIE PROGRAMMIERSCHULE

PYRO