Why Modern, High-performance Networking Matters for Interactive Computing

Fernando Pérez
Helen Wills Neuroscience Institute, UC Berkeley
http://fperez.org
Fernando.Perez@berkeley.edu

Brian Granger, Cal Poly, San Luis Obispo
Evan Patterson, California Institute of Technology

SIAM CSE 2011, Reno, NV
March 2, 2011
IPython: I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ Exploratory computing
IPython: I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ Exploratory computing
IPython: I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ Exploratory computing
Python: an excellent base for an interactive scientific system

- Dynamic code evaluation
- No variable declarations
- Powerful introspection
- Very regular object model
- Excellent string processing

IPython: far beyond Python’s interactive shell
Python: an excellent base for an interactive scientific system

- **Dynamic** code evaluation
- No variable declarations
- **Powerful** introspection
- Very **regular** object model
- Excellent **string** processing

IPython: far beyond Python’s interactive shell
Python: an excellent base for an interactive scientific system

- Dynamic code evaluation
- No variable declarations
- Powerful introspection
- Very regular object model
- Excellent string processing

IPython: far beyond Python’s interactive shell
Utilities needed to emulate Python's interactive interpreter.

# Inspired by similar code by Jeff Epler and Fredrik Lundh.

```python
import sys
import traceback
from codeop import CommandCompiler, compile_command

__all__ = ["InteractiveInterpreter", "InteractiveConsole", "interact", "compile_command"]

def softspace(file, newvalue):
    oldvalue = 0
    try:
        oldvalue = file.softspace
    except AttributeError:
        pass
    try:
        file.softspace = newvalue
```
When things go wrong

```
In [13]: run ~/scratch/error
reps: 5

ValueError: Traceback (most recent call last)
/home/fperez/scratch/error.py in <module>()
    70 if __name__ == '__main__':
    71     #explode()
    ---> 72     main()
    73     g2='another global'

/home/fperez/scratch/error.py in main()
    60     array_num = zeros(size,'d')
    61     for i in xrange(reps):
    ---> 62         RampNum(array_num, size, 0.0, 1.0)
    63     RNtime = time.clock()-t0
    64     print 'RampNum time:', RNtime

/home/fperez/scratch/error.py in RampNum(result, size, start, end)
    43     tmp = zeros(size+1)
    44     step = (end-start)/(size-1-tmp)
    ---> 45     result[:] = arange(size)*step + start
    46
    47 def main():

ValueError: shape mismatch: objects cannot be broadcast to a single shape
```

In [14]:
Plotting at the console

In [1]: import math, numpy
In [2]: from scipy.integrate import quad
In [3]: from scipy.special import j0
In [4]: def j0i(x):
   ...:     """Integral form of J_0(x)""
   ...:     def integrand(phi):
   ...:         return math.cos(x * math.sin(phi))
   ...:     return (1.0 / math.pi) * quad(integrand, 0, math.pi)[0]
   ...
In [5]: x = numpy.linspace(0, 20, 200) # sample grid: 200 points between 0 and 20
In [6]: y = j0i(x) # sample J0 at all values of x
In [7]: x1 = x[::10] # subsample the original grid every 10th point
In [8]: y1 = map(j0i, x1) # evaluate the integral form at all points in x1
In [9]: # Make a plot with these values (the ; suppresses output)
In [10]: plot(x, y, label=r'$J_0(x)$');
In [11]: plot(x1, y1, 'ro', label=r'$\int_0^{\infty} \cos(x \sin \phi) \, d\phi$');
In [12]: axhline(0, color='green', label='_nolegend_');
In [13]: title(r'\text{Verify } J_0(x) = \int_0^{\infty} \cos(x \sin \phi) \, d\phi$');
In [14]: xlabel('x$');
In [15]: legend();
In [16]: matshow(numpy.random.random((32, 32)))
Out[16]: <matplotlib.figure.Figure instance at 0x4630042c>
Getting all the power from interactive computing in Python

1. A better Python shell: object introspection, system access, extensible 'magic' commands, ...

2. A flexible, embeddable interpreter:
   1. debugging, mix batch/interactive work.
   2. build custom systems based on Python with new syntax, etc.

3. Data visualization and GUIs: Matplotlib, Mayavi, all GUIs toolkits.

4. A rich toolkit: terminal, Qt console, HTTP client.

5. High level (and interactive!) parallel computing interfaces.
Quick stats. http://www.ohloh.net/p/ipython

Ohloh Analysis Summary

- Mostly written in Python
- Mature, well-established codebase
- Very large, active development team
- Extremely well-commented source code

Updated 01 Mar 2011 14:14 UTC

Ratings & Reviews

Community Rating: 4.6/5.0

Your Rating

Avg. Salary: $55,000/year

Project Cost

This calculator estimates how much it would cost to hire a team to write this project from scratch.

Include: Markup And Code

Codebase: 78,759

Effort (est.): 19 Person Years

Total Cost: $1,065,165
### Lines of Code By Language

<table>
<thead>
<tr>
<th>Language</th>
<th>Code Lines</th>
<th>Comment Lines</th>
<th>Comment Ratio</th>
<th>Blank Lines</th>
<th>Total Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>72,290</td>
<td>44,415</td>
<td>38.1%</td>
<td>28,013</td>
<td>144,718</td>
</tr>
<tr>
<td>XML</td>
<td>3,931</td>
<td>0</td>
<td>0.0%</td>
<td>115</td>
<td>4,046</td>
</tr>
<tr>
<td>HTML</td>
<td>819</td>
<td>0</td>
<td>0.0%</td>
<td>65</td>
<td>884</td>
</tr>
<tr>
<td>Emacs Lisp</td>
<td>522</td>
<td>364</td>
<td>41.1%</td>
<td>89</td>
<td>975</td>
</tr>
<tr>
<td>CSS</td>
<td>499</td>
<td>11</td>
<td>2.2%</td>
<td>91</td>
<td>601</td>
</tr>
<tr>
<td>Perl</td>
<td>401</td>
<td>440</td>
<td>52.3%</td>
<td>208</td>
<td>1,049</td>
</tr>
<tr>
<td>Make</td>
<td>254</td>
<td>49</td>
<td>16.2%</td>
<td>108</td>
<td>411</td>
</tr>
<tr>
<td>shell script</td>
<td>143</td>
<td>106</td>
<td>42.6%</td>
<td>63</td>
<td>312</td>
</tr>
<tr>
<td>Vim Script</td>
<td>124</td>
<td>2</td>
<td>1.6%</td>
<td>17</td>
<td>143</td>
</tr>
<tr>
<td>Objective-C</td>
<td>30</td>
<td>7</td>
<td>18.9%</td>
<td>12</td>
<td>49</td>
</tr>
</tbody>
</table>

This analysis was updated about 18 hours ago. (01 Mar 2011 14:14 UTC)
Some projects using IPython

**Scientific**
- **PyRAF**: Space Telescope Science Institute
- **CASA**: National Radio Astronomy Observatory.
- **Ganga**: CERN.
- **PyMAD**: neutron spectrometer, Institut Laue Langevin.
- **Sardana**: European Synchrotron Radiation Facility.
- **ASCEND**: engineering modeling (Carnegie Mellon).
- **JModelica**: dynamical systems.
- Denver Aerosol Sources and Health (**DASH**), CU Boulder.
- **PyIMSL** Studio, by Visual Numerics.
- **Trilinos**: Sandia National Lab.
- **Sage**: open source mathematics.
- **Pymerase**: microarray gene expression.

**Web/Other**
- **Django** web framework.
- **Turbo Gears** web framework.
- **Pylons** web framework
- **Zope** and **Plone** CMS.
- Axon Shell, BBC Kamaelia.
- **Schevo** database.
- **Pitz**: distributed task/bug tracking.
- **iVR** (interactive Virtual Reality).
- **Movable Python** (portable Python environment).
- ...
Core idea: manage a namespace

- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment
Core idea: manage a namespace

- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment
IPython: a REPL (Read/Eval/Print Loop)

Core idea: manage a namespace

- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment
More complex interactive uses?

Kernel

Client - Terminal

Client - Qt

Client - ...

Client: monitor email, publish, ...
A messaging protocol

Direct communication
- Execute code (‘eval’)
- Object information
- Complete
- History
- Connect

Broadcasting
- Functional execution:
  - Python inputs
  - Python outputs
  - Python errors
- Side effects:
  - Streams (stdout, stderr, etc)
  - Display data: plots, other payloads
The socket library that acts as a concurrency framework

- Pure C++ library.
- Python bindings in Cython (Brian Granger, Min RK). Python 2.5-3.2.
- Python bindings run messaging in native threads - no GIL
- Abstractions are at the message delivery level, not the raw-bytes level.
- Socket types encapsulate messaging patterns.
- Open source (LGPL), actively developed.
ØMQ: Messaging patterns

Figure 1 – Request-Reply

Figure 4 – Publish-Subscribe

Image credit: official ØMQ documentation
Interactive IPython on ØMQ
Back to the clients: a rich Qt Console
Enthought: sponsorship, Evan Patterson.

Feels like a console, runs like a GUI

- Inline and floating images
- Syntax highlighting, full multiline editing
- Session saving
  - HTML (with PNG or SVG)
  - PDF/printing
- Help viewer
- %magics, !system access, IPython...
- Detach/reattach support
Forward ØMQ to HTTP: a web frontend!

James Gao, UC Berkeley

```python
In [16]: t = linspace(-pi, pi, 1024)
s = sin(10*t) / t
plot(t, s, linewidth=1.0)
xlabel('time (s)')
ylabel('voltage (mV)')
title('About as simple as it gets, folks')
grid(True)
```

![Graph](image-url)
Multiple users of one process instead of many processes for one user

These could be two different hosts on separate networks
A short demo

(time permitting)
To wrap up: why should you care?

- **Interactive user**: interfaces adapted to each environment
  - Terminal. For those days when you can only SSH into your cluster...
  - Web.
  - Rich Qt client...

- **Scientific programmer**: embed an IPython kernel into your code
  - Monitor it over the network,
  - Inspect it interactively, ...

- IPython as **your interactive shell**: custom magics.
  - `%pyomo [options] <model.py> [model.dat]`

- Build **custom interactive environments**
  - even beyond the Python language - Sage

- **Civilized parallelism** - next talk.

**Just one idea**

Reuse a common set of abstractions to manipulate a namespace, define the protocols to do it and use a networking layer that makes it possible.
Support for IPython

- Bivio Software http://www.bivio.biz
- Tech-X corporation http://txcorp.com
- Ohio Super Computing Center and Defense High Performance Computing Modernization Program (HPCMP) - José Unpingco, Ohio State.
- NIH (NiPy project) - Grant 5R01MH081909-02 from National Institute Of Mental Health, IRG: ZRG1. Mark D’Esposito, UC Berkeley.
- Google (Summer of Code).
- Microsoft - Wenming Ye and Shahrokh Mortazavi.
(Incomplete) Cast of Characters

- **Brian Granger** - Physics, Cal State San Luis Obispo
- **Min Ragan-Kelley** - UC Berkeley
- **Robert Kern** - Enthought
- **Jörgen Stenarson** - Sweden.
- **Evan Patterson** - Physicfs, Caltech
- **Thomas Kluyver** - Plant biology, U. Sheffield
- Stéfan van der Walt - Applied Math, U. Stellenbosch, South Africa
- Satra Ghosh- MIT Neuroscience
- Gaël Varoquaux - Neurospin (Orsay, France)
- Ville Vainio - CS, Tampere University of Technology, Finland
- Ondrej Certik - Physics, U Nevada Reno
- Laurent Dufréchou - France
- Darren Dale - Cornell
- Justin Riley - MIT
- James Gao - UC Berkeley
- Mark Voorhies - UC San Francisco
- Thomas Spura - Fedora project
- Many more! (~60 commit authors)
  - **SciPy 2011**: Austin, TX. July 11-16.


Scipy India: since 2009.
  - **Scipy India 2011**: December.

Scipy Japan 2011
  - **Being planned**...

Many workshops (general and discipline specific)
Thank You!

http://ipython.scipy.org

http://github.com/ipython

http://fperez.org

Fernando.Perez@berkeley.edu