MPI Cluster Programming with Python and Amazon EC2

Pete Skomoroch

juice analytics

datawrangling.com
juiceanalytics.com/writing

Outline

Netflix Prize & Amazon EC2

Python Parallel Programming Options

ElasticWulf

MPI basics in Python

Demo: ElasticWulf, mpi4py, Ipython1

ElasticWulf Performance

EC2 pointers + Q&A

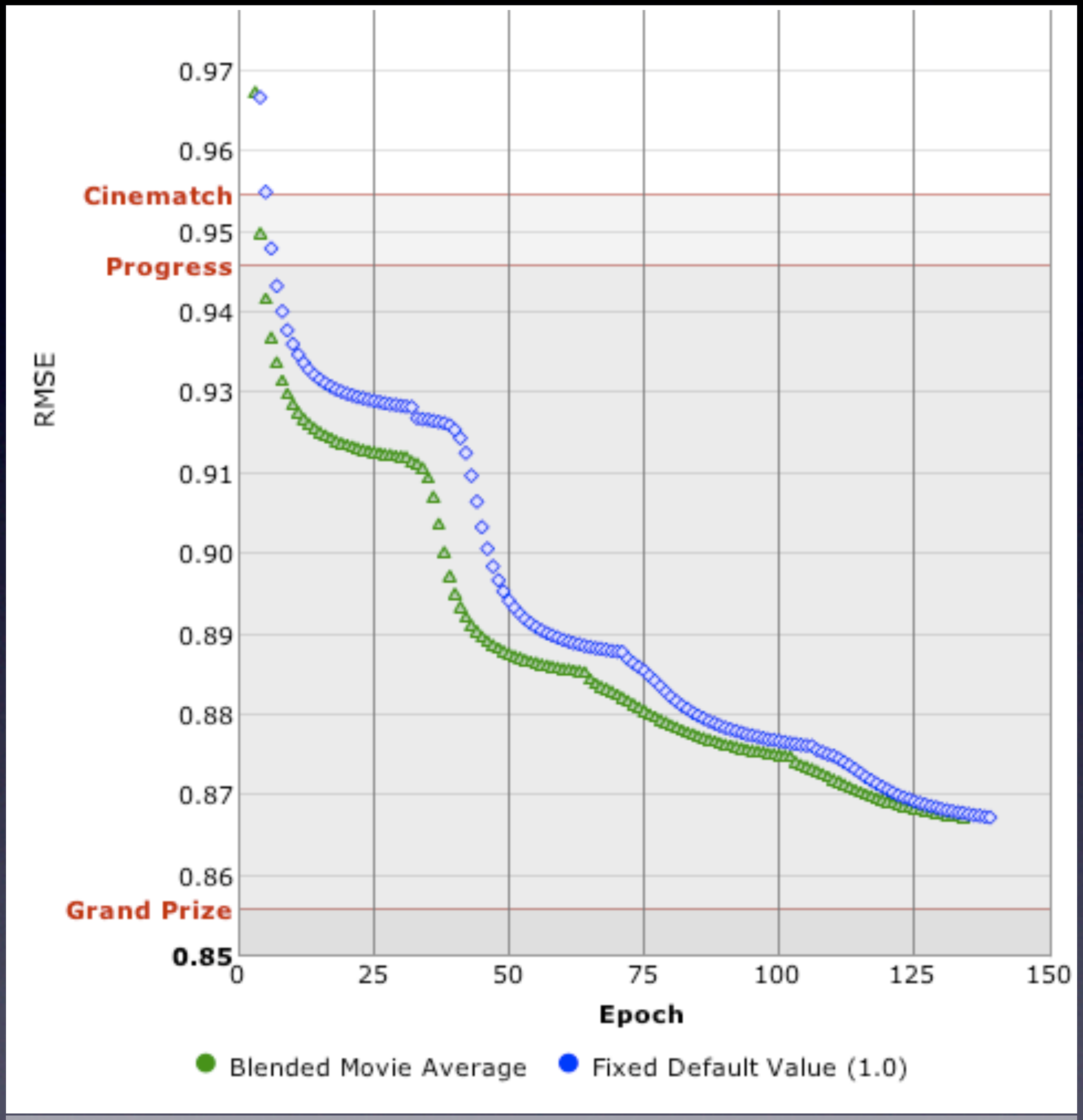
The Netflix Prize

17K movies
500K users
100M ratings
2 GB raw data

A “big” computation?

10% improvement of RMSE wins $1 Million

Crunching the data at home...



Use the Pyflix library to squeeze dataset into 600 MB

http://pyflix.python-hosting.com

Simon Funk iterative SVD algorithm runs on your laptop overnight

“Timely Development” code gets us to 4% improvement over Cinematch

Use Numpy/Scipy, call C when needed

I needed more CPUs

Basic algorithm ties up your machine for 13 hours

Some algorithms take weeks...

Need runs over many sets of parameters

Need to run cross validation over large datasets

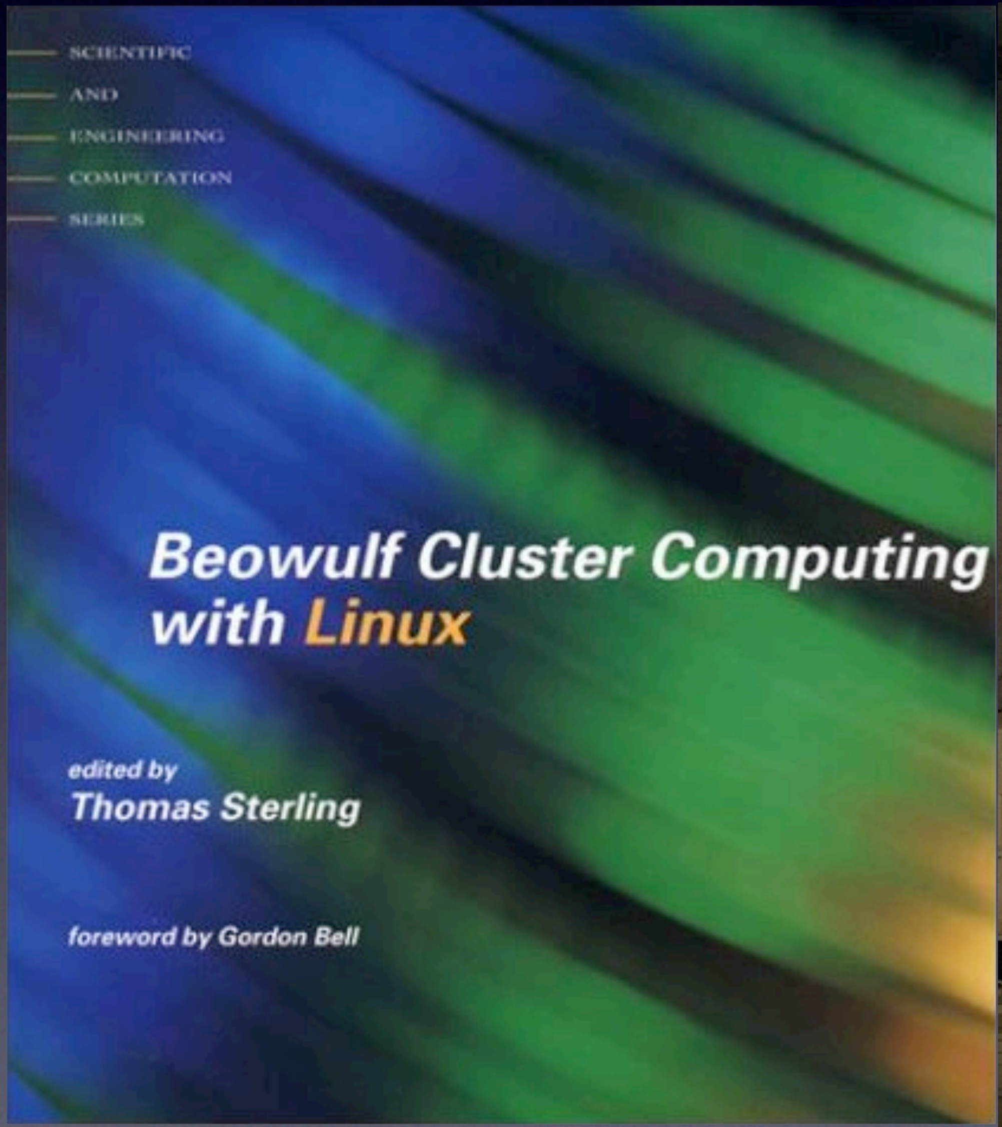
Bugs in long jobs suck

The best approach so far (Bell Labs) merged the successful results of over 107 different algorithms

But this was my only machine...

How can I get a Beowulf cluster?

Cluster of nearly identical commodity machines
Employ message passing model of parallel computation
Often have shared filesystem



Amazon EC2

`__init__(`  `)`

Launch instances of Amazon Machine Images (AMIs) from your Python code

Pay only for what you use (wall clock time)

http://www.flickr.com/photos/steverosebush/2241578490/

EC2 Instance Types

	Small 1x	Large 4x	ExtraLarge 8x
RAM	1.7GB	7.5GB	15GB
Disk	160GB	850GB	1.6TB
CPU (1.7Ghz)	1 32bit	4 64bit	8 64bit
I/O Performance	Moderate	High	High
Price (per instance-hour)	$0.10	$0.40	$0.80

Driving EC2 using Python

yum -y install python-boto

Blog post: “Amazon EC2 Basics for Python Programmers”
- James Gardner

PyCon talk: “Like Switching on the Light: Managing an Elastic Compute Cluster with Python”
George Belotsky; Heath Johns

http://del.icio.us/pskomoroch/ec2

Parallel Programming in Python

MapReduce (Hadoop + Python)
Python MPI options (mpi4py, pyMPI, ...)
Wrap existing C++/Fortran libs (ScaLAPACK,PETSc, ...)
Pyro
Twisted
IPython1

Which one you use depends on your particular situation

These are not mutually exclusive or exhaustive...

Introducing ElasticWulf

ElasticWulf - batteries included

Master and **worker** beowulf node Amazon Machine Images
Python command line scripts to launch/configure cluster
Includes Ipython1 and other good stuff...

The AMIs are publicly visible
Updated python scripts + docs will be on Google Code

BLAS/LAPACK	OpenMPI	Xwindows	Twisted
Numpy/Scipy	MPICH	NFS	Ipython1
ipython	MPICH2	Ganglia	mpi4py
matplotlib	LAM	C3 tools	boto

What is MPI?

- High performance message passing interface (MPI)
- Standard protocol implemented in multiple languages
- Point to Point - Collective Operations
- Very flexible, complex...
- We will just look at a use case involving a master process broadcasting data and receiving results from slave processes via simple primitives (bcast(), scatter(), gather(), reduce())

MPI Basics in Python

Lets look at some code...

size attribute: number of cooperating processes

rank attribute: unique identifier of a process

We will cover some PyMPI examples, but I would recommend using mpi4py (moved over to scipy, used by Ipython1)

http://mpi4py.scipy.org/

The program structure is nearly identical in either implementation.

MPI Broadcast

```python
import mpi
import math

# create an array on the master node only
if mpi.rank == 0:
    data = [sin(x) for x in range(0,10)]
else:
    data = None

# each process gets a copy of the data
common_data = mpi.bcast(data)
```

mpi.bcast()

broadcasts a value from the root process to all other processes

MPI Scatter

```
import mpi
import math

if mpi.rank == 0:
    array = [sin(x*pi / 99) for x in range(100)]
else:
    array = None

# each node gets a portion of the data
local_array = mpi.scatter(array)
```

mpi.scatter()

scatters an array roughly evenly across processes

MPI Gather

```
import mpi
import math

# each process calculates an array based on rank (the process number)
local_data = [sin(mpi.rank*x*pi)/99 for x in range(100)]
# append all the data together
root_data = mpi.gather(local_data)
```

mpi.gather()

Collects results from all processes back to the master process

MPI Reduce

```python
import mpi

# calculate a value based on the process rank
x_cubed = mpi.rank**3
# sum all the process values
sum_x_cubed = mpi.reduce(x_cubed, mpi.SUM)
```

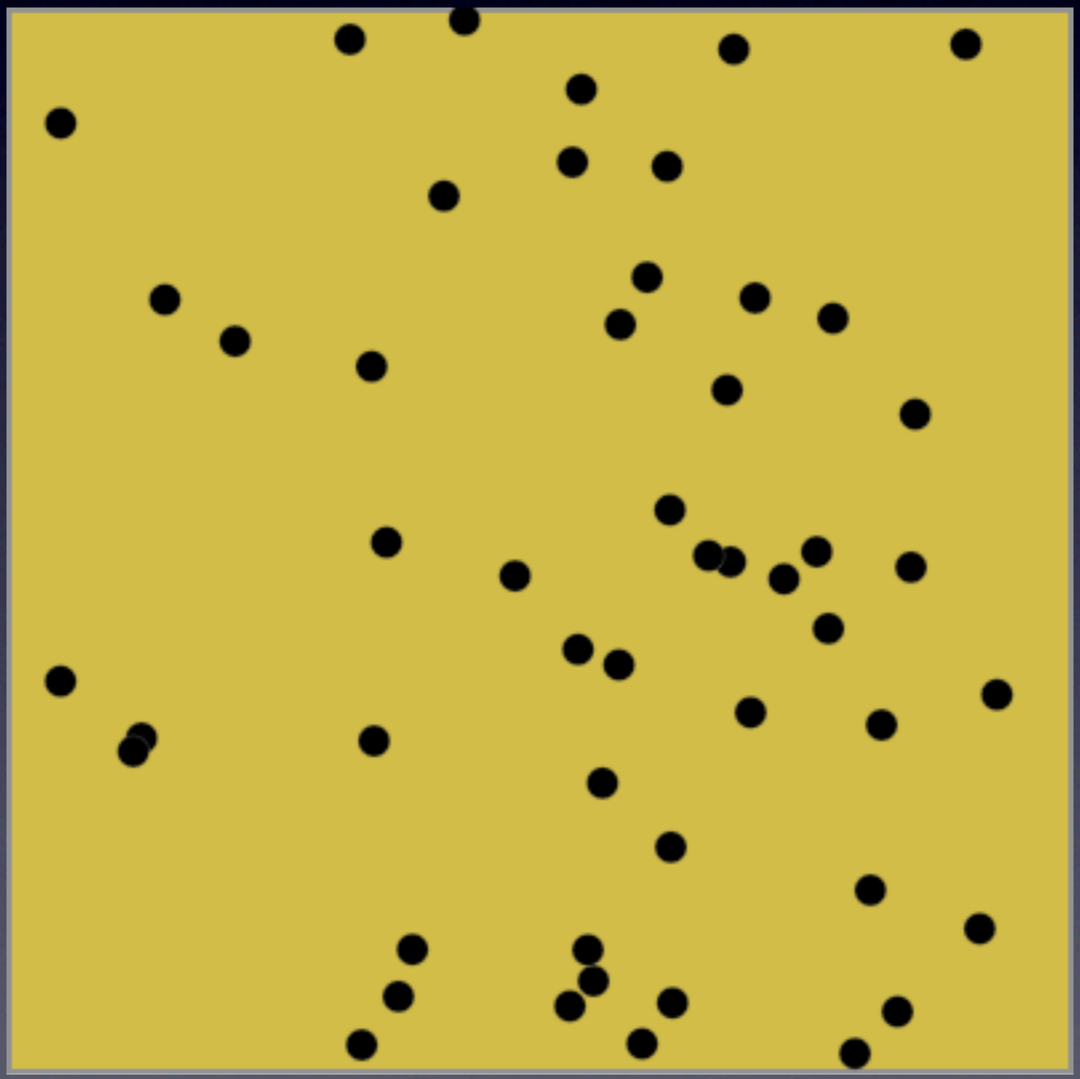
mpi.reduce()

Apply a summary function across all nodes. Like Python reduce (at least until Guido kills it)

A simple example

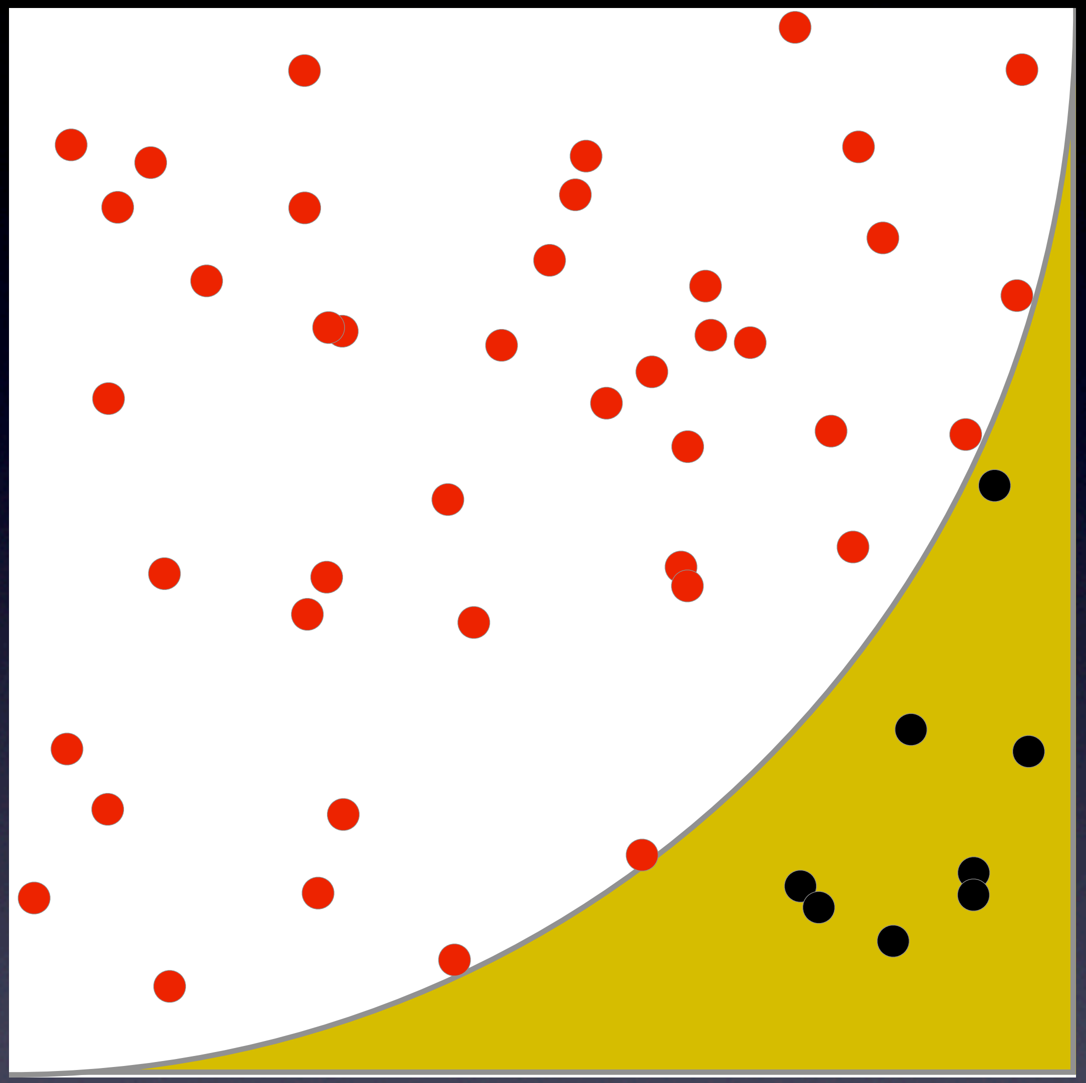
Calculating Pi.

Throw random darts at a dartboard.

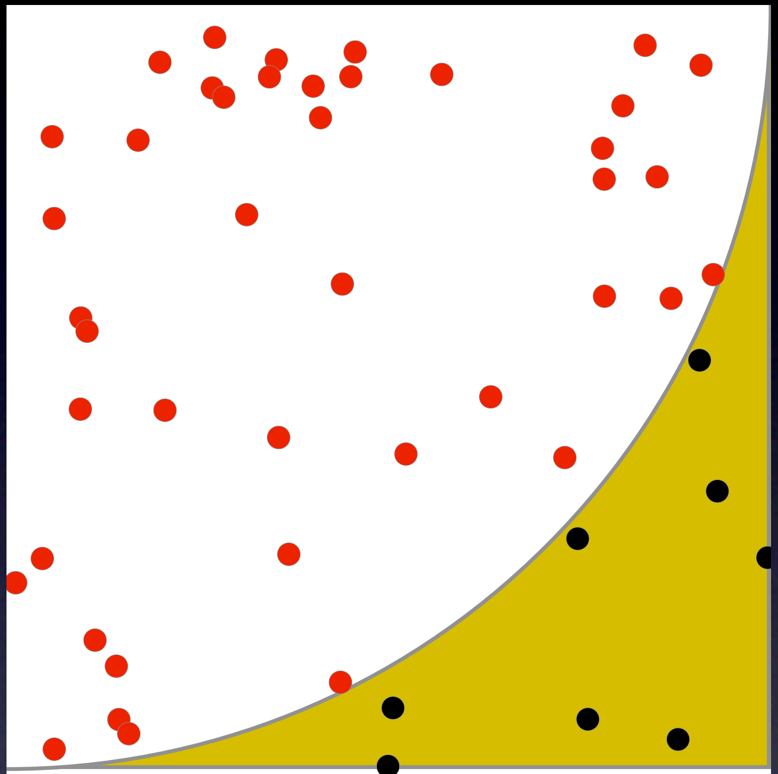


Monte Carlo Pi

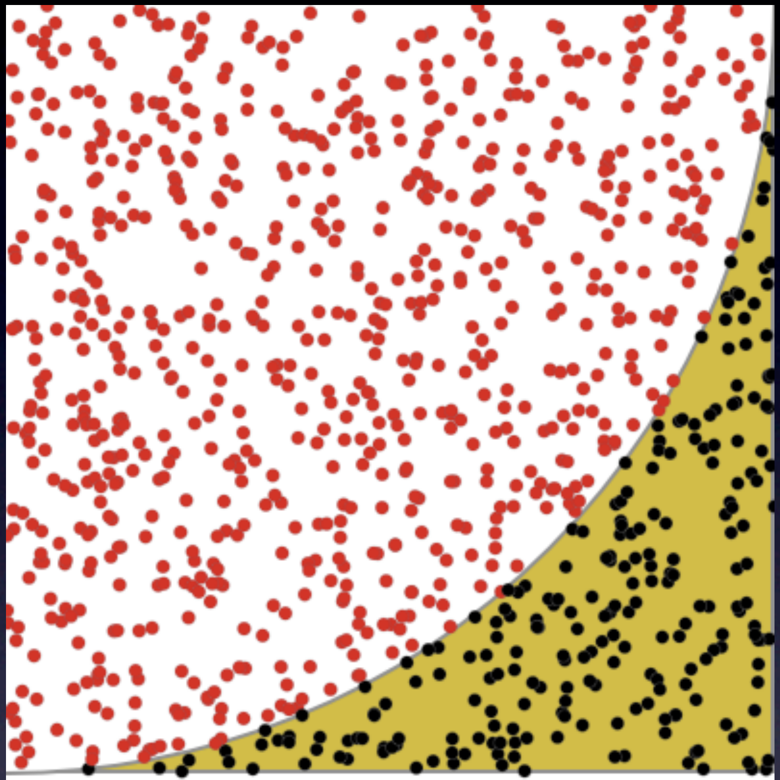
Measure the distance from the origin, count the number of "darts" that are within 1 unit of the origin, and the total number of darts...



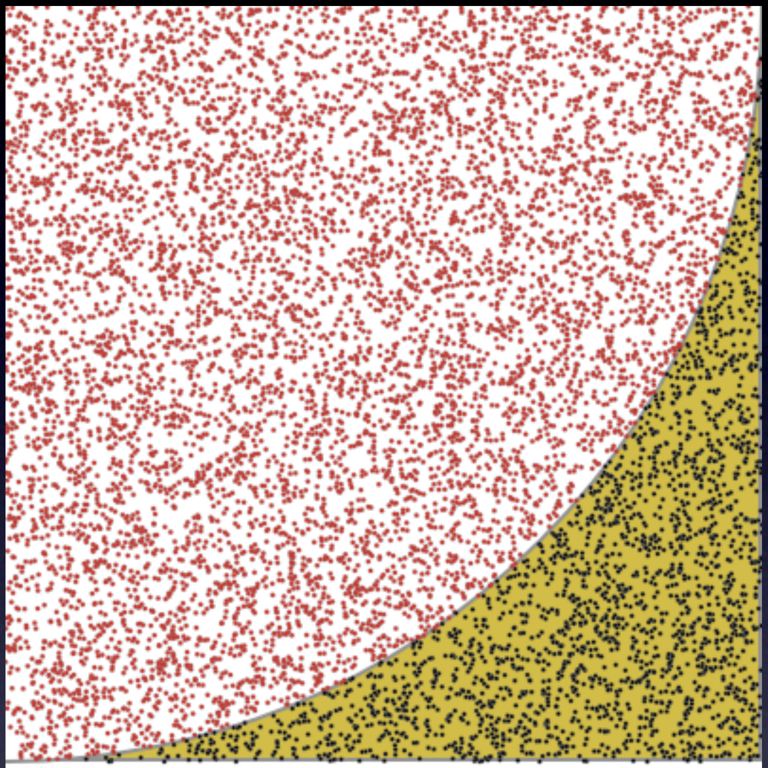
Throwing more darts



42 inside the circle
8 outside the circle
estimated pi is 3.360 (42*4 / (42+8))



767 inside the circle
233 outside the circle
estimated pi is 3.068 (767*4 / (767+233))

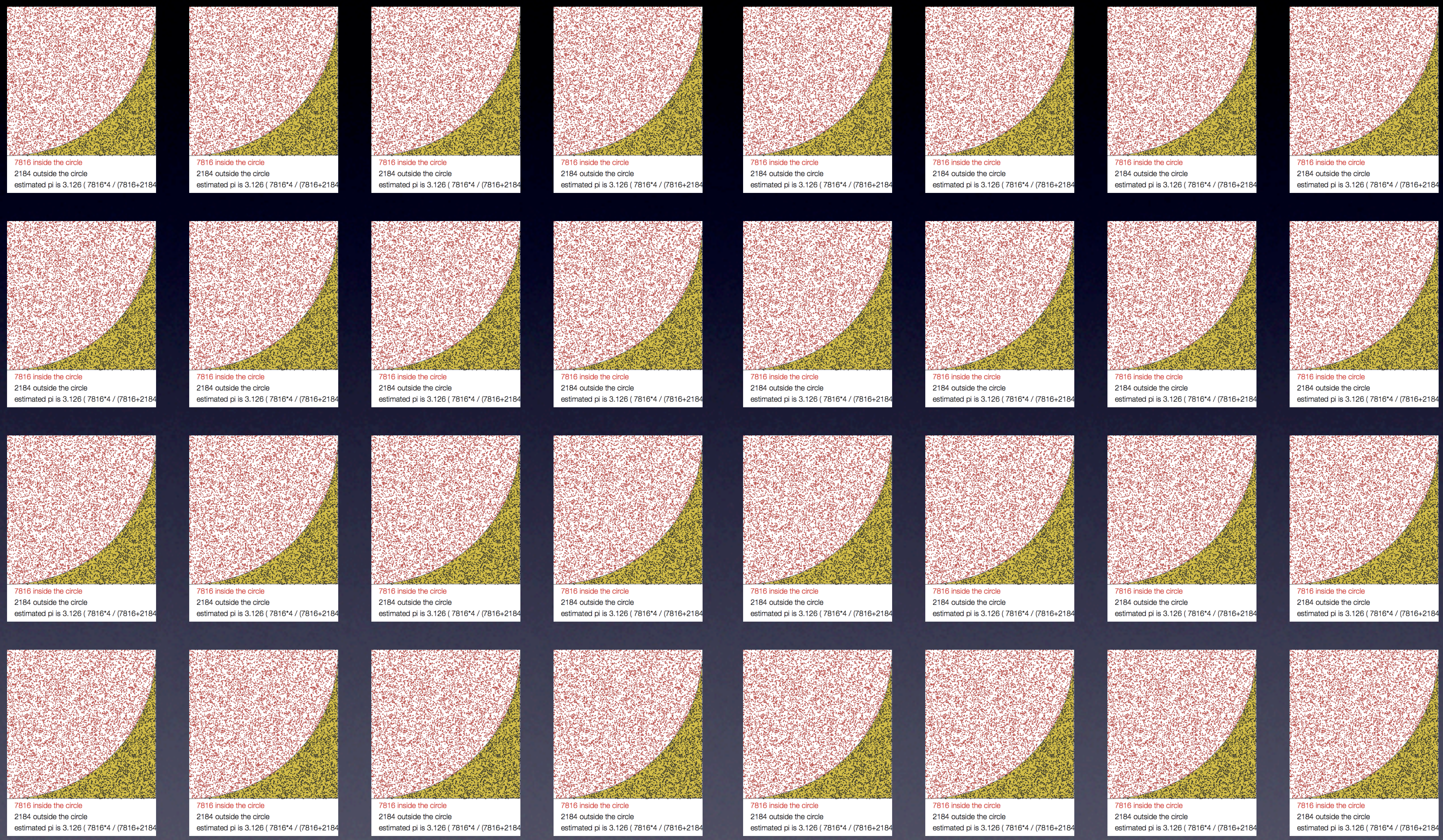


7816 inside the circle
2184 outside the circle
estimated pi is 3.126 (7816*4 / (7816+2184

Serial Pi Calculation

```
import random

def compute_pi(nsamples):
    inside_circle_cnt = 0
    for i in xrange(nsamples):
        # randomly choose a point in the box
        x = random.random()
        y = random.random()
        if ((x*x) + (y*y) < 1.0):
            inside_circle_cnt += 1

    mypi = (4.0 * inside_circle_cnt)  / nsamples
    return mypi

if __name__=="__main__":
    pi = compute_pi(10000)
    print "Computed value of pi is ", pi
```

Throwing darts in parallel



Parallel Pi Calculation

```
import random
import mpi

def compute_pi(nsamples):
    rank, size = mpi.rank, mpi.size

    inside_circle_cnt = 0
    for i in xrange(nsamples):
        # randomly choose a point in the box
        x = random.random()
        y = random.random()
        if ((x*x) + (y*y) < 1.0):
            inside_circle_cnt += 1

    mypi = float(inside_circle_cnt) / nsamples
    pi =  (4.0 / mpi.size) * mpi.allreduce(mypi, mpi.SUM)
    return pi

if __name__=="__main__":
    pi = compute_pi(10000)
    if mpi.rank == 0:
        print "Computed value of pi on", mpi.size, "processors is", pi
```

Starting up your ElasticWulf Cluster

1) Sign up for Amazon Web Services
2) Get your keys/certificates + define environment variables
4) Download ElasticWulf Python scripts
4) Add your Amazon Id to the ElasticWulf config file
5) ./start-cluster.py -n 3 -s ‘xlarge’
6) ./ec2-mpi-config.py
7) ssh in to the master node...

Demo: Start the cluster

$ ./ec2-start-cluster.py

```
[rumblefish:~] pete$
[rumblefish:~] pete$
[rumblefish:~] pete$ ./ec2-start-cluster.py
image ami-eb13f682
master image ami-e813f681
----- starting master -----
RESERVATION     r-ff2cdc96      567513963419    default
INSTANCE        i-3717ef5e      ami-e813f681                    pending
----- starting workers -----
RESERVATION     r-fe2cdc97      567513963419    default
INSTANCE        i-3617ef5f      ami-eb13f682                    pending
INSTANCE        i-0917ef60      ami-eb13f682                    pending
INSTANCE        i-0817ef61      ami-eb13f682                    pending
```

Demo: check progress

$ ./ec2-check-instances.py

```
[rumblefish:~] pete$ ./ec2-check-instances.py
----- listing instances -----
RESERVATION     r-e82cdc81      567513963419    default
INSTANCE        i-5117ef38      ami-e813f681                            terminated
RESERVATION     r-eb2cdc82      567513963419    default
INSTANCE        i-5017ef39      ami-eb13f682                            terminated
RESERVATION     r-ff2cdc96      567513963419    default
INSTANCE        i-3717ef5e      ami-e813f681    ec2-67-202-58-163.compute-1.amazonaws.com       ip
RESERVATION     r-fe2cdc97      567513963419    default
INSTANCE        i-3617ef5f      ami-eb13f682    ec2-67-202-30-147.compute-1.amazonaws.com       do
INSTANCE        i-0917ef60      ami-eb13f682    ec2-67-202-0-182.compute-1.amazonaws.com        do
INSTANCE        i-0817ef61      ami-eb13f682    ec2-67-202-0-91.compute-1.amazonaws.com domU-12-31
[rumblefish:~] pete$
```

Demo: configure the nodes

$ ./ec2-mpi-config.py

```
Mounting other filesystems:  [  OK  ]

Configuration complete, ssh into the master node as lamuser and boot the cluster:
$ ssh lamuser@ec2-67-202-58-163.compute-1.amazonaws.com
> mpdboot -n 4 -f mpd.hosts

 You can check that MPI is working by running the following commands...

> mpdtrace
> mpiexec -n 4 /usr/local/src/mpich2-1.0.6p1/examples/cpi
> mpdringtest 100
> mpiexec -l -n 4 hostname

Test PyMPI:
> mpirun -np 4 pyMPI /home/beowulf/pyMPI-2.4b2/examples/fractal.py

Test mpi4py:
> mpiexec -n 4 python /home/beowulf/mpi4py-0.5.0/tests/cpi-buf.py


You can monitor your cluster performance in your web browser at:
  http://ec2-67-202-58-163.compute-1.amazonaws.com/ganglia

Directories /mnt/data and /home/beowulf are NFS mounted on all nodes

For interactive graphical sessions, start x11 on your local machine then ssh in wit
h the following options:
ssh -2 -C -Y lamuser@ec2-67-202-58-163.compute-1.amazonaws.com

Once logged in, you can start up ipython:
[lamuser@ip-12-345-67-89]$ ipython -pylab


[rumblefish:~] pete$
```

Demo: login to the master node

```
[rumblefish:~] pete$
[rumblefish:~] pete$ ssh lamuser@ec2-67-202-58-163.compute-1.amazonaws.com

         __|  __|_  )  Rev: 3
         _|  (     /   Arch: x86_64
        ___|\___|___|

 Welcome to an EC2 Public Image

 DataWrangling EC2Wulf Fedora 6 Master Node

 see http://www.datawrangling.com


[lamuser@ip-10-251-167-15 ~]$ ls
_trial_temp  _trial_temp_old55297705  hosts  mpd.hosts
```

Demo: start the MPI daemons

```
[lamuser@ip-10-251-167-15 ~]$
[lamuser@ip-10-251-167-15 ~]$ mpdboot -n 4 -f mpd.hosts
[lamuser@ip-10-251-167-15 ~]$ mpdtrace
ip-10-251-167-15
domU-12-31-38-00-61-91
domU-12-31-38-00-08-51
domU-12-31-38-00-05-A1
[lamuser@ip-10-251-167-15 ~]$ mpdringtest 100
time for 100 loops = 0.176836967468 seconds
[lamuser@ip-10-251-167-15 ~]$ mpiexec -n 2 /usr/local/src/mpich2-1.0.6p1/examples/c
pi
Process 0 of 2 is on ip-10-251-167-15
Process 1 of 2 is on domU-12-31-38-00-61-91
pi is approximately 3.1415926544231318, Error is 0.0000000008333387
wall clock time = 0.004676
```

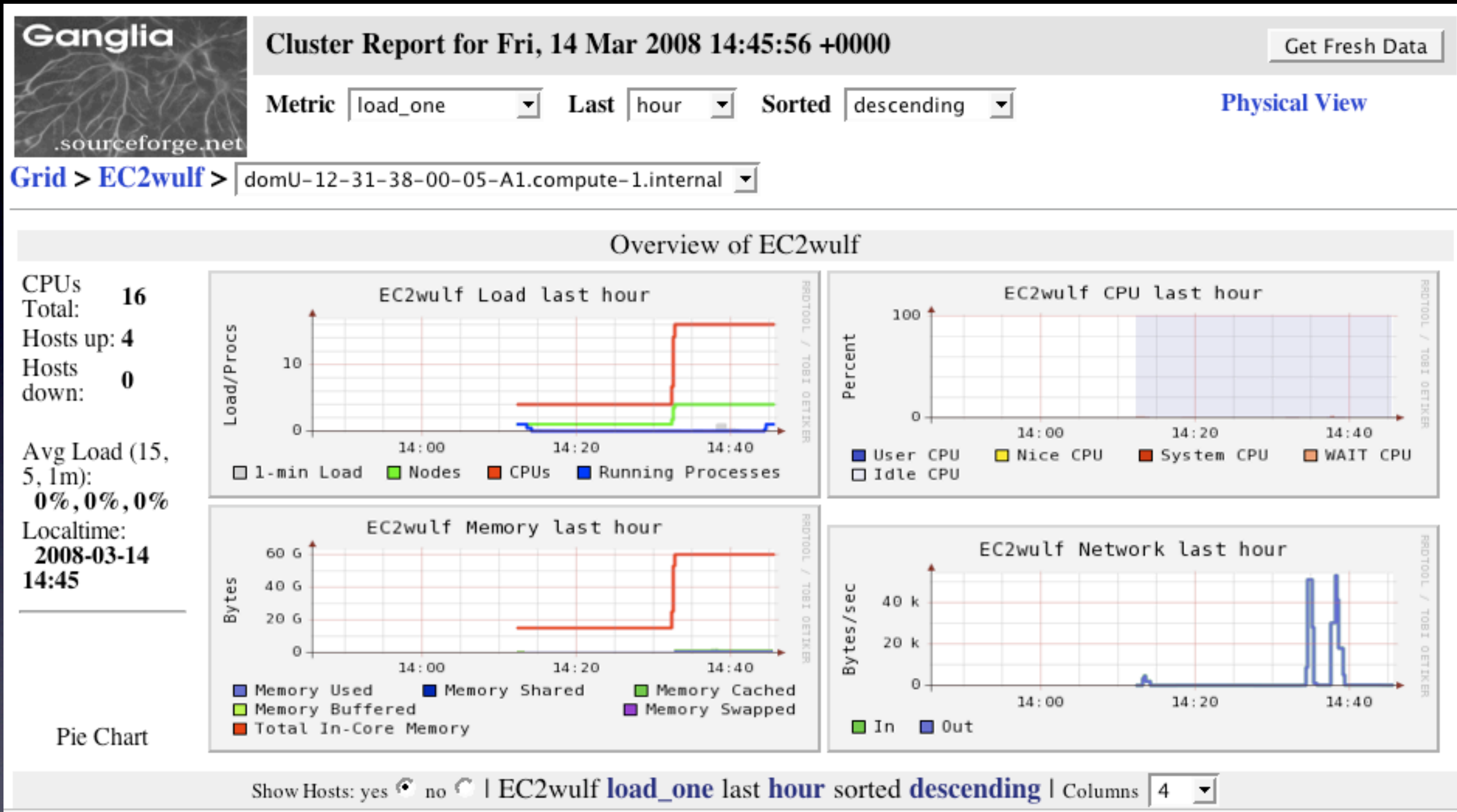
Demo: mpi4py code

```
#!/bin/env python
"""
Parallel PI computation using Collective Communications
of Python objects exposing memory buffers (requires NumPy).

usage::

  $ mpiexec -n <nprocs> cpi-buf.py
"""

from mpi4py import MPI
from math   import pi as PI
from numpy  import array

def get_n():
    prompt  = "Enter the number of intervals: (0 quits) "
    try:
        n = int(raw_input(prompt));
        if n < 0: n = 0
    except:
        n = 0
    return n

def comp_pi(n, myrank=0, nprocs=1):
    h = 1.0 / n;
    s = 0.0;
    for i in xrange(myrank + 1, n + 1, nprocs):
        x = h * (i - 0.5);
        s += 4.0 / (1.0 + x**2);
    return s * h

def prn_pi(pi, PI):
    message = "pi is approximately %.16f, error is %.16f"
--More--(57%)
```

Demo: run mpi4py code

```
[lamuser@ip-10-251-167-15 ~]$ mpiexec -n 2 python /home/beowulf/mpi4py-0.5.0/tests/cpi-buf.py
Enter the number of intervals: (0 quits) 20
pi is approximately 3.1418009868930934, error is 0.0002083333033003
Enter the number of intervals: (0 quits) 0
[lamuser@ip-10-251-167-15 ~]$ mpiexec -n 32 python /home/beowulf/mpi4py-0.5.0/tests/cpi-buf.py
Enter the number of intervals: (0 quits) 20
pi is approximately 3.1418009868930938, error is 0.0002083333033007
Enter the number of intervals: (0 quits) 0
[lamuser@ip-10-251-167-15 ~]$ more /home/beowulf/mpi4py-0.5.0/tests/cpi-buf.py
```

Running Ganglia



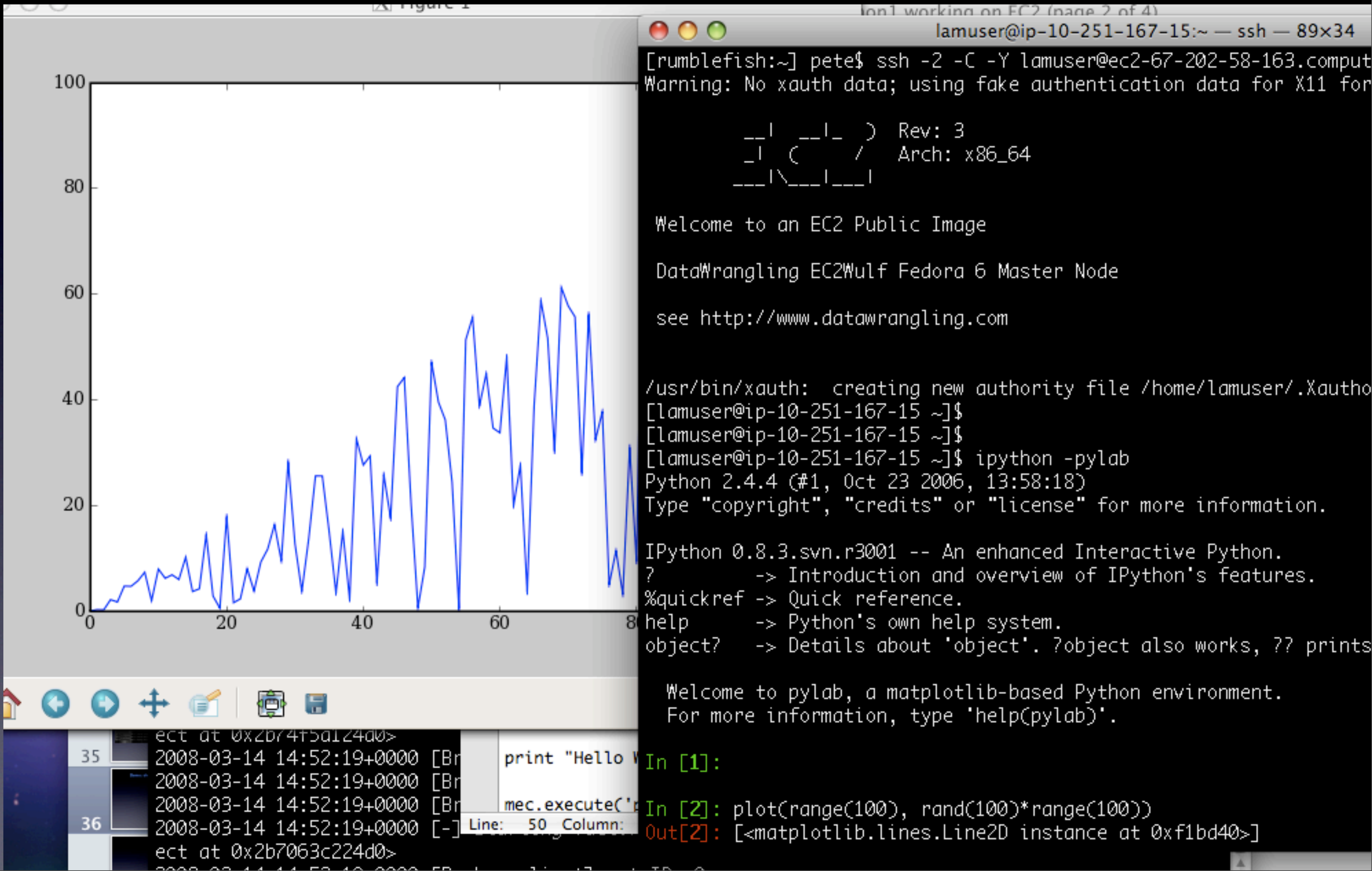
Demo: start ipython1 controller

```
[lamuser@ip-10-251-167-15 ~]$
[lamuser@ip-10-251-167-15 ~]$ ipcontroller &
[1] 3090
[lamuser@ip-10-251-167-15 ~]$ 2008-03-14 14:50:51+0000 [-] Log opened.
2008-03-14 14:50:51+0000 [-] ipython1.kernel.enginepb.PBEngineServerFactory starting on 10201
2008-03-14 14:50:51+0000 [-] Starting factory <ipython1.kernel.enginepb.PBEngineServerFactory ins
tance at 0x2b0d56a90a28>
2008-03-14 14:50:51+0000 [-] Starting controller interface: multiengine
2008-03-14 14:50:51+0000 [-] Starting controller network interface (multiengine): xmlrpc::10105
2008-03-14 14:50:51+0000 [-] Adapting: <ipython1.kernel.multiengine.MultiEngine object at 0x2b0d5
6a91390>
2008-03-14 14:50:51+0000 [-] ipython1.external.twisted.web2.channel.http.HTTPFactory starting on
10105
2008-03-14 14:50:51+0000 [-] Starting factory <ipython1.external.twisted.web2.channel.http.HTTPFa
ctory instance at 0x2b0d56a90dd0>
2008-03-14 14:50:51+0000 [-] Starting controller interface: task
2008-03-14 14:50:51+0000 [-] Starting controller network interface (task): xmlrpc::10113
2008-03-14 14:50:51+0000 [-] ipython1.external.twisted.web2.channel.http.HTTPFactory starting on
10113
2008-03-14 14:50:51+0000 [-] Starting factory <ipython1.external.twisted.web2.channel.http.HTTPFa
ctory instance at 0x2b0d56b0a758>

[lamuser@ip-10-251-167-15 ~]$
```

Demo: start engines with mpirun

```
[lamuser@ip-10-251-167-15 ~]$ mpirun -n 32 ipengine --controller-ip=master
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Starting factory <ipython1.kernel.enginepb.PBEngineClientFactory obj
ect at 0x2b704a6874d0>
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Log opened.
2008-03-14 14:52:19+0000 [-] Starting factory <ipython1.kernel.enginepb.PBEngineClientFactory obj
ect at 0x2b7c370ae4d0>
2008-03-14 14:52:19+0000 [-] Starting factory <ipython1.kernel.enginepb.PBEngineClientFactory obj
ect at 0x2ba2522e14d0>
2008-03-14 14:52:19+0000 [-] Starting factory <ipython1.kernel.enginepb.PBEngineClientFactory obj
ect at 0x2b346b0994d0>
2008-03-14 14:52:19+0000 [-] Starting factory <ipython1.kernel.enginepb.PBEngineClientFactory obj
ect at 0x2b08a48a04d0>
2008-03-14 14:52:19+0000 [-] Starting factory <ipython1.kernel.enginepb.PBEngineClientFactory obj
```

Demo: parallel ipython1



Demo: parallel ipython1

```
In [5]: from ipython1.kernel import client

In [6]: mec = client.MultiEngineClient(('127.0.0.1',10105))

In [7]: mec.get_ids()
Out[7]:
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
```

Scaling KNN on ElasticWulf

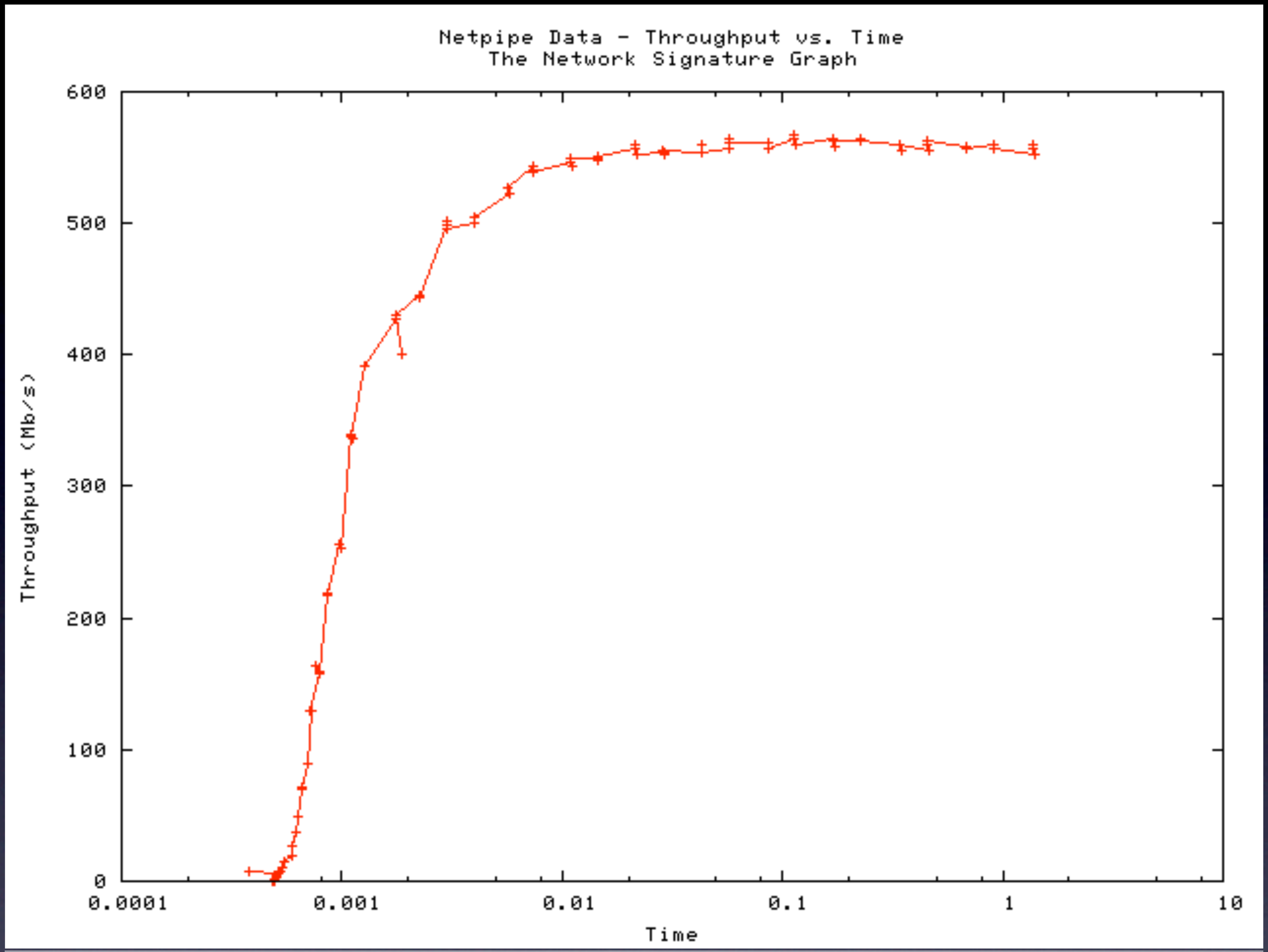
Run pearson correlation between every pair of movies...
Use Numpy & mpi4py to scatter the movie ratings vectors
Spend $2.40 for 24 cpu hours, trade $ for time

```
One Flew Over the Cuckoo's Nest (1975)

10 Nearest Neighbors:

To Kill a Mockingbird (1962)
Shawshank Redemption, The (1994)
Fargo (1996)
Godfather, The (1972)
Saving Private Ryan (1998)
Casablanca (1942)
Rear Window (1954)
Silence of the Lambs, The (1991)
Amadeus (1984)
Schindler's List (1993)
Wizard of Oz, The (1939)
```

```
Star Wars: Episode VI - Return of the Jedi (1983)

10 Nearest Neighbors:

Star Wars: Episode V - The Empire Strikes Back (1980)
Star Wars: Episode IV - A New Hope (1977)
Star Wars: Episode I - The Phantom Menace (1999)
Raiders of the Lost Ark (1981)
Indiana Jones and the Last Crusade (1989)
Ghostbusters (1984)
Back to the Future (1985)
Jurassic Park (1993)
Batman (1989)
E.T. the Extra-Terrestrial (1982)
Jurassic Park (1993)
```

Performance

Latency: 0.000492 (microseconds)



Netpipe Benchmark

Amazon EC2 - Beowulf Performance Suite Results

Generated on 2008-03-06 at 21:22 by the Beowulf Performance Suite

Suite Information is available here.

This report was generated by the Cluster Monkey FC6 rpm.

The benchmarks were run on a virtual Amazon EC2 cluster which consisted of 2 Extra Large Instances running Fedora Core 6.

Machine Data:

Machine Name	ip-10-251-139-31
Kernel	2.6.16.33-xenU
CPU 1 Model	Dual Core AMD Opteron(tm) Processor 270
CPU 2 Model	Dual Core AMD Opteron(tm) Processor 270
CPU 3 Model	Dual Core AMD Opteron(tm) Processor 270
CPU 4 Model	Dual Core AMD Opteron(tm) Processor 270
CPU 1 MHz	2004.544
CPU 2 MHz	2004.544
CPU 3 MHz	2004.544
CPU 4 MHz	2004.544

Benchmarks:

- NAS Parallel:npb.gnu4.mpich2.A.16 results
- NAS Parallel:npb.gnu4.mpich2.A.4 results
- NAS Parallel:npb.gnu4.mpich2.B.16 results
- NAS Parallel:npb.gnu4.mpich2.B.4 results
- netperf results
- netpipe results
- stream results

EC2 “gotchas”

EC2 is still in Beta: instances die, freeze, be prepared

Relatively high latency

Dynamic subnet

ec2-upload-bundle --retry

/etc/fstab overwritten by default in bundle-vol

more: http://del.icio.us/pskomoroch/ec2+gotchas

Questions?