PiCloud:
A simple approach to cloud computing

Ken Younge
Krannert School of Management
Purdue University

2013 New Directions in Text as Data Workshop
London School of Economics, September 27-29, 2013
Overview

• Amazon AWS + PiCloud.com

• Example projects
  - Patent-to-patent similarity  7 million patents
  - USPTO office actions  5 million rulings

• Example code
  - 1 job  One VM, 64GB RAM, a lot of storage
  - 1,000 jobs  A repetitive task, split up into 1,000 blocks
  - 100 jobs x 7 threads x multiple steps  ‘Queues’ to implement a pipeline
Amazon AWS

- Amazon sells access to virtual machines (VMs) running on the cloud.
- The scale and rate of expansion of AWS drive down costs.
- AWS requires considerable setup, configuration, and system admin.
- AWS offers high-performance-computing (HPC) clusters, but you need HPC-coded solutions (e.g., molecular modeling, genome analysis, ...).
- AWS doesn’t provide an easy way to parallelize the kinds of custom, ad-hoc, cobbled-together programs we often run in text analysis.
PiCloud

- PiCloud manages AWS for you
  - they provide a unified control panel to monitor execution across multiple computers
  - you can remotely monitor stdout, stderr, CPU, RAM, disk, swap, memalloc, etc...
    (but if you don’t know how to do any of that - that’s OK too).

- PiCloud provides a simple API to transfer data, execute code, and collect your results
  - Your core functions can be written in Python, R, C, C++, Java, MATLAB, Fortran, etc...
  - You can customize Linux environments to run anything that runs in Linux x86-64
  - You can call the API from Python or from a command line interface
  - There are many types of VMs (large/small RAM, fast/slow CPU, large/small disk, ...)

Ken Younge - Purdue University
Capacity

- You can scale up to thousands of machines and unlimited storage
- You can reserve capacity when you know you need it
  ... but excess capacity is usually available

Cost

- AWS + PiCloud is cheaper than all of the small-, mid-, and super-computing-sized clusters I’ve reviewed
- You pay for what you use
- You can enter a hard-stop dollar limit to cover your downside
**Example Project:** Patent-to-patent vector space model

- Scrape content from the US Patent and Trademark Office
- Build a vocabulary space
- Vectorize every patent
- Save a sparse vector for each patent (small file)
- Calculate patent-to-patent cosine similarities as needed
Example Project: USPTO ‘office actions’

- Remotely mount 5 million ZIP archives
- OCR (tesseract) particular files from the archive
- Python-nltk the text
- Identify events in the text that we care about
- Build a directed graph of events
Example Code: Run one job

- Maybe you need more RAM
- Maybe you need your computer for something else
- Run it on the cloud:

```python
import cloud
import nltk

def calculate():
    # insert code here that you want to run on the cloud
    # you can save results to cloud storage for download
    # or return the result(s) directly from the function
    return "All Done"

job_id = cloud.call(calculate, _type='m1', _cores=8) # m1 with 64 GB RAM
```
Example Code: Run 1,000 jobs

• Maybe you have a simple job but you’re in a hurry
• Chop the problem into 1,000 batches and run 1,000 jobs

```python
import cloud
import Levenshtein

def calculate(batch_no):
    # insert code here that you want to run on the cloud

    # limit execution to the "batch" of operations represented
    # by the batch_no parameter passed to the function

    # you can save results to cloud storage for download
    # or return the result(s) directly from the function

    return "Results from batch no " + str(batch_no)

batch_nos = [i for i in range(1000)]
job_ids = cloud.map(calculate, batch_nos, _type='c1', _env='younge')
```
Example Code: Scrape website from unique IP addresses

```python
import cloud
import urllib2
import myfuncs

def scrape(patno):
    # scrape
    url = "http://patft.uspto.gov/netacgi/nph-Parser?S1=" + str(patno) + ".PN."
    cnn = urllib2.urlopen(url)
    content = cnn.read()
    cnn.close()

    # save
    fname = str(patno) + ".html"
    cloud.bucket.putf(content, fname)
    cloud.bucket.make_public(fname)

    # move patno to the 'DONE' queue
    return patno

def main():
    # initialize queues
    q_todo = cloud.queue.get('ToDo')
    q_done = cloud.queue.get('Done')
    q_err = cloud.queue.get('Error')

    # push list of patent numbers onto the starting queue
    patnos = load_list("patnos.txt")
    q_todo.push(patnos)

    # start execution
    q_todo.attach(scrape, q_done,
                  on_error=[{Exception: ['queue': q_err, 'delay': 0]},
                             retry_on=[urllib2.HTTPError, urllib2.URLError],
                             retry_delay=10, max_retries=3, max_parallel_jobs=1000,
                             readers_per_job=7, _type="s1")
    
    if __name__ == "__main__": exit(main())
```
Example: Control Panel showing the queuing system
### Displaying jobs

<table>
<thead>
<tr>
<th>id</th>
<th>parent</th>
<th>key</th>
<th>hostname</th>
<th>function</th>
<th>label</th>
<th>created</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>184782</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 05:32:24</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>184781</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 05:32:24</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>184780</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 05:32:24</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>184779</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 05:32:24</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>184778</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 05:32:24</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>184777</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 01:25:04</td>
<td>✗️</td>
<td></td>
</tr>
<tr>
<td>184776</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 01:25:03</td>
<td>✗️</td>
<td></td>
</tr>
<tr>
<td>184775</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 01:25:03</td>
<td>✗️</td>
<td></td>
</tr>
<tr>
<td>184774</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 01:25:03</td>
<td>✗️</td>
<td></td>
</tr>
<tr>
<td>184773</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-10 01:25:03</td>
<td>✗️</td>
<td></td>
</tr>
<tr>
<td>184772</td>
<td>5682</td>
<td>Kens-MacBook-Pro-2.local</td>
<td>__main__match at sample.py:315</td>
<td>queue-dd_firmnos</td>
<td>2013-09-04 14:45:10</td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>
Summary

• Cloud computing can be easy and cheap
• I’m happy to chat more down at the pub!

Thank You
## Machine Types

<table>
<thead>
<tr>
<th>Core Type</th>
<th>Compute Units</th>
<th>Memory</th>
<th>Disk</th>
<th>Max Multicore</th>
<th>Price/Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1 (default)</td>
<td>1</td>
<td>300 MB</td>
<td>15 GB</td>
<td>1</td>
<td>$0.05</td>
</tr>
<tr>
<td>c2</td>
<td>2.5</td>
<td>800 MB</td>
<td>30 GB</td>
<td>8</td>
<td>$0.13</td>
</tr>
<tr>
<td>f2</td>
<td>5.5 w/ HT</td>
<td>3.7 GB</td>
<td>100 GB</td>
<td>16</td>
<td>$0.22</td>
</tr>
<tr>
<td>m1</td>
<td>3.25</td>
<td>8 GB</td>
<td>140 GB</td>
<td>8</td>
<td>$0.30</td>
</tr>
<tr>
<td>s1</td>
<td>0.5 to 2</td>
<td>300 MB</td>
<td>4 GB</td>
<td>1</td>
<td>$0.04</td>
</tr>
</tbody>
</table>