Jug Tutorial: Coarse-Level Parallel Programming in Python

Luis Pedro Coelho

lpc@cmu.edu

January 29, 2010

Carnegie Mellon

Problem

Brute force a password.

Assumptions

The crypt module exists with elements:

- decrypt: decrtypt ciphertext given a password.
- isgood: test whether this text could be the plaintext.
- letters: just the letters.

Also, we know that the password is five letters.

Simple Solution

```
import itertools
from crypt import decode, letters, isgood, preprocess

ciphertext = file('secret.msg').read()
ciphertext = preprocess(ciphertext)

for p in itertools.product(letters, repeat=5):
    text = decode(ciphertext, p)
    if isgood(text):
        passwd = "".join(map(chr,p))
        print '%s:%s' % (passwd, text)
```

This solution does not take advantage of multiple processors.

Tasks

```
import itertools
from crypt import decode, letters, isgood, preprocess

ciphertext = file('secret.msg').read()
ciphertext = preprocess(ciphertext)

def decrypt(ciphertext, p):
    text = decode(ciphertext, p)
    if isgood(text):
        passwd = "".join(map(chr,p))
        return (passwd, text)
    # else: return None

results = []
for p in itertools.product(letters, repeat=5):
    results.append(Task(decrypt,ciphertext,p))
```

Python Magic

```
from jug import TaskGenerator
import itertools
from crypt import decode, letters, isgood, preprocess

ciphertext = file('secret.msg').read()
ciphertext = preprocess(ciphertext)

@TaskGenerator
def decrtypt(ciphertext, p):
    text = decode(ciphertext, p)
    if isgood(text):
        passwd = "".join(map(chr,p))
        return (passwd, text)
    # else: return None

results = []
for p in itertools.product(letters, repeat=5):
    results.append(decrypt(ciphertext,p))
```

Enter Jug

You give it the Jugfile, it runs the tasks for you!

Jug Loop

```
while len(tasks) > 0:
    ready = [t for t in tasks if can_run(t)]
    for t in ready:
        if not is_running(t):
            t.run()
        tasks.remove(t)
```

Except jug is much fancier!

Jug Advantages

1. Automatic task-level parallelization with dependency tracking.
2. Remember all intermediate results.
3. Makes writing parallel code look like writing sequential code.

This example is actually not so good.
We have $26^5 \approx 11M$ tasks, all of which run very fast.
As a rule of thumb, your tasks should take at least a couple of seconds.

Solution each task will be:
Given a letter, try all passwords beginning with that letter.
Now, we have 26 tasks. Much better.

```
@TaskGenerator
def decrypt(prefix):
    res = []
    for suffix in product(letters, repeat=5-len(prefix)):
        passwd = np.concatenate([prefix, suffix])
        text = decode(ciphertext, passwd)
        if isgood(text):
            passwd = "".join(map(chr, passwd))
            res.append( (passwd, text) )
    return res

@TaskGenerator
def join(partials):
    return list(chain(*partials))

results = join([decrypt([p]) for p in letters])
```

Let's call jug now.
(Assuming our code is in a file called jugfile.py)

```
$jug status
Task name       Waiting  Ready  Finished Running
-------------------------------------------------
jugfile.join          1      0         0       0
jugfile.decrypt       0     26         0       0
.................................................
Total:                1     26         0       0
```

Some tasks are ready. None are running.

```
$jug execute &
[1] 29501
$jug execute &
[2] 29502
```

Executing in the background...

```
$jug status
Task name       Waiting  Ready  Finished Running
-------------------------------------------------
jugfile.join          1      0         0       0
jugfile.decrypt       0     24         0       2
.................................................
Total:                1     24         0       2
```

Two tasks running. Good.

Wait a few minutes. . .

```
$jug status
Task name       Waiting  Ready  Finished Running
-------------------------------------------------
jugfile.join          1      0         0       0
jugfile.decrypt       0     14        10       2
.................................................
Total:                1     14        10       2
```

Ten tasks have finished.
Notice how the join task must wait for all the others.

A few more minutes. . .

```
$jug status
Task name       Waiting  Ready  Finished Running
-------------------------------------------------
jugfile.join          0      0         1       0
jugfile.decrypt       0      0        26       0
.................................................
Total:                0      0        27       0
```

What Now?

All tasks are finished!
How do I get to the results?

```
import jug
jug.init('jugfile', 'jugdata')
import jugfile
results = jug.task.value(jugfile.results)
for p,t in results:
        print "%s\n\n    Password was '%s'" % (t,p)
```

jug.init is necessary to initialise the jug backend.
Then, the jugfile can be directly imported as a Python module.

For more information

http://luispedro.org/software/jug
http://github.com/luispedro/jug