Labs Next Week

Does this work for you?

```bash
ssh student.ssh.inf.ed.ac.uk
```

then from there

```bash
ssh scutter0$((RANDOM%7+1))
```
If that didn’t work, ask for access to the Hadoop Cluster:

http://www.inf.ed.ac.uk/systems/support/form/
Lab Allocation

Go to the lab you picked on Doodle. Ignore the official assignments.

Everybody have a non-clashing lab?
Extreme Computing
Let’s implement MapReduce!
On the exam

- Understand how MapReduce works
- Pseudocode for mappers/reducers
- Performance considerations

Not on the exam (but generally useful)

- Command line programs
- This implementation
- Python
- C++
Goal: Word Count

We’ll take a text file and collect the count of each word.
```python
#!/usr/bin/python
import sys
for line in sys.stdin:
    for word in line.split():
        print(word + "\t1")
```

Text → Mapped

```
this is toy
this 1
is 1
toy 1
toy 1
is 1
small 1
```
```python
#!/usr/bin/python
import sys

for line in sys.stdin:
    for word in line.split():
        print(word + '\t1')
```

This code snippet demonstrates a simple text processing pipeline. It takes a text file as input, processes it, and outputs the word frequencies. Here's a breakdown of the process:

1. **Text Input**: The input text is `this is toy
toy is small`.
2. **Mapped Output**: The program splits the text into words and prints each word followed by a tab and the number 1. The output looks like:
   - `this 1`
   - `is 1`
   - `toy 1`
   - `is 1`
   - `toy 1`
   - `is 1`
   - `small 1`
3. **Sorted Output**: The words are sorted alphabetically, and their counts are aggregated:
   - `is 1`
   - `is 1`
   - `small 1`
   - `this 1`
   - `toy 1`
   - `toy 1`

This is a basic example of how the MapReduce framework can be implemented in Python.
import fileinput
key, count = None, 0
for line in fileinput.input():
    key2, count2 = line.strip().split('	')
    count2 = int(count2)
    if key2!=key:
        if key:
            print(key, count, sep='\t')
        key, count = key2, count2
    else:
        count += count2
if key:
    print(key, count, sep='\t')
```python
#!/usr/bin/python
import sys
for line in sys.stdin:
    for word in line.split():
        print(word + "\t1")
```

Text
---
this is toy
toy is small

Mapped
---
this 1
is 1
toy 1
is 1
toy 1
is 1
small 1

Sorted
---
is 1
is 1
small 1
toy 1
this 1

Reduced
---
is 2
small 1
toy 2
Measuring Performance

```
pv big.txt >/dev/null
```

9.09MiB 0:00:02 [2.94MiB/s] [> ] 0% ETA 0:06:40

- **pv**: Print a file with a progress bar.
- **big.txt**: A text file I made for you.
- **>/dev/null**: Discard the output
Let’s Watch

`pv -c -N map medium.txt |./map.py |sort | \`n
`pv -c -N reduce |./reduce.py >/dev/null`n

    `pv`  Make a progress bar.
    `-c`  Do not mess up the terminal, please.
    `-N map`  Name the progress bar.
    `\`  Continue on the next line.
`>/dev/null`  Discard the output.
What we have now

- One mapper
- One sort
- One reducer

Faster?
GNU Parallel

```
pv big.txt |./map.py >/dev/null
  95.5MiB 0:00:06 [15.7MiB/s] [> ] 5% ETA 0:01:48

pv big.txt |parallel --pipe ./map.py >/dev/null
  639MiB 0:00:15 [38.1MiB/s] [====> ] 34% ETA 0:00:27
```

```
parallel Powerful parallelization tool
  --pipe Split stdin, run jobs on multiple cores
```
Sorting is a bottleneck

pv big.txt | parallel --pipe ./map.py | sort > /dev/null
81.1 MiB 0:00:27 [3.26 MiB/s] [>] 4% ETA 0:09:41

Way slower 😞
Can we parallelize this?
#!/usr/bin/python
#Usage: ./shard.py mapper shards
import sys
shards = [open(str(p) + "-from-" + sys.argv[1], "w")
    for p in range(int(sys.argv[2]))]
for l in sys.stdin:
    key = l.split("	")[0]
    shard = hash(key) % len(shards)
    shards[shard].write(l)
Toy Sharding

./map.py <toy.txt | ./shard.py 0 2
pv 0-from-* | sort | ./reduce.py
pv 1-from-* | sort | ./reduce.py
Toy Sharding

```
./map.py <toy.txt | ./shard.py 0 2
pv 0-from-* | sort | ./reduce.py
pv 1-from-* | sort | ./reduce.py
```

Parallel Mapping and Sharding

```
pv medium.txt | parallel --pipe ./map.py \| ./shard.py {#} 2
pv 0-from-* | sort | ./reduce.py
pv 1-from-* | sort | ./reduce.py
```

\| Escape the | character so sharding is part of the parallel command

{#} Mapper number
Parallel Map and Reduce

```
pv medium.txt |parallel --pipe ./map.py \| ./shard.py {} 2
parallel cat {}-from-* \| sort \| ./reduce.py ::: 0 1

  {}  Substitute argument (reducer number) here.
::: 0 1  Arguments to substitute are 0 and 1 (for two reducers).
```
Command Line MapReduce

- Parallel map and reduce
- Single machine\(^1\)
- Limited fault tolerance

\(^1\)GNU parallel can SSH (awesome!), but data still passes through one machine