Some real-world examples of parallel processing on LOTUS

Example 2: extract spatial subsets from CMIP5 experiments (1)

Requirement:

Extract spatial subsets from CMIP5 a set of experiments.

Details:

- For each model:
  - For each variable (hus, ps, ta, ua & va):
    - Extract a spatial subset (80° to 140° Longitude; -30° to 40° Latitude)
    - Where:
      - Frequency: 6hr
      - Realm: atmosphere
      - Ensemble: r1i1p1

Example 2: extract spatial subsets from CMIP5 experiments (2)

Basic Implementation:

Script 1 (bash):

- For each variable (hus, ps, ta, ua & va):
  - Make output directory
  - Find all relevant input NetCDF files (using a glob pattern)
  - Call Python script; extract spatial subset; write output

Script 2 (Python):

  - Read input file; extract spatial subset for variable; write output file.
  - Main code used: cf-python library

Extract from: **extract_cmip5_subset.py**

```
import cf
f = cf.read(infile)
subset = f[2].subspace(latitude=cf.wi(bb.south, bb.north), longitude=cf.wi(bb.west, bb.east))
cf.write(subset, outfile)
```

Example 2: extract spatial subsets from CMIP5 experiments (3)

Parallel Implementation using LOTUS:

Script 1 (bash):

- For each variable (hus, ps, ta, ua & va):
  - Make output directory
  - Find all relevant input NetCDF files
  - Submit a job to the LOTUS scheduler that will call the Python script using the "**bsub**" command:

```
bsub -q lotus -o $outdir/`date +%s`.txt ~/extract_cmip5_subset.py $nc_file $this_dir $var
```

Why use this approach?

- Because you can submit 200 jobs in one go.
- Lotus executes jobs when resource becomes available
- They will all run and complete in parallel

Example 3: chaining tasks with Jug: calculating monthly means (1)

Requirement:

- Calculate a monthly mean from 6-hourly model data

Details:

- For each date in the month:
  - Extract data for 00Z, 06Z, 12Z & 18Z
  - Calculate daily average from them
- Gather all daily averages
- Calculate monthly average
- Write monthly average to netCDF file

Example 3: chaining tasks with Jug: calculating monthly means (2)

Basic Implementation in Python (PART 1):

```
import cdms2

def calcDailyMean(date, var = "U10"):
    f = cdms2.open("/badc/ecmwf-era-interim/metadata/cdml/era-interim_ggas.xml")
    v = f(var, time = ("%s 00:00" % date, "%s 23:59" % date))
    total = sum([v[i] for i in range(4))
    f.close()
    return total

def calcMonthlyMean(data, date):
    total = sum([data[i] for i in range(len(data))])
    return total

def writeMonthlyMean(data, fname):
    f = cdms2.open(fname, "w")
    f.write(data)
    f.close()
```

Example 3: chaining tasks with Jug: calculating monthly means (3)

Basic Implementation in Python (PART 2):

```
(...continued...)

year = 2001
month = 1
all_days = []

for day in range(1, 32):
    d = "%.4d-%.2d-%.2d" % (year, month, day)
    all_days.append(calcDailyMean(d))

monthly_mean = calcMonthlyMean(all_days, "%.4d-%.2d-15" % (year, month))

output_path = "/group_workspace/jasmin/megalon/output.nc”
writeMonthlyMean(monthly_mean, output_path)
```

Run sequentially: `$ python2.7 monthly_mean.py`

Example 3: chaining tasks with Jug: calculating monthly means (4)

Convert to Jug Tasks by adding:

```
from jug import TaskGenerator, barrier
import cdms2

@TaskGenerator
def calcDailyMean(date, var = "U10"):
    f = cdms2.open("/badc/ecmwf-era-interim/metadata/cdml/era-interim_ggas.xml")
    v = f(var, time = ("%s 00:00" % date, "%s 23:59" % date))
    total = sum([v[i] for i in range(4))
    f.close()
    return total

@TaskGenerator
def calcMonthlyMean(data, date):
    total = sum([data[i] for i in range(len(data))])
    return total

@TaskGenerator
def writeMonthlyMean(data, fname):
```

Example 3: chaining tasks with Jug: calculating monthly means (5)

Add Jug "barriers" to wait for dependent tasks to complete:

```
(...continued...)

year = 2001
month = 1
all_days = []

for day in range(1, 32):
    d = "%.4d-%.2d-%.2d" % (year, month, day)
    all_days.append(calcDailyMean(d))

barrier()
monthly_mean = calcMonthlyMean(all_days, "%.4d-%.2d-15" % (year, month))

barrier()
output_path = "/group_workspace/jasmin/megalon/output.nc"
writeMonthlyMean(monthly_mean, output_path)
```

Example 3: chaining tasks with Jug: calculating monthly means (6)

Now we can run with Jug, locally (on 4 processors):

```
$ jug execute monthly_mean.py &
$ jug execute monthly_mean.py &
$ jug execute monthly_mean.py &
$ jug execute monthly_mean.py &


$ jug status monthly_mean.py    # will report on status
```

Example 3: chaining tasks with Jug: calculating monthly means (7)

Or run on LOTUS:

On LOTUS we can use the "Job Array" submission feature to submit the same Jug script multiple times to the cluster, e.g. Run 40 times (on 40 processors):

```
$ dir=/group_workspace/jasmin/megalon
$ bsub -e $dr/%J-%I.err -o $dr/%J-%I.out -J "jug[1-40]" jug execute monthly_mean.py
```

Each process will output to:

- ${dir}/${job_id}-${array_count}.out

We can still monitor progress using “jug status”

```
$ jug status monthly_mean.py    # will report on status
```

So why did we use Jug?

- Jug makes it trivial to **convert existing functions into parallel "Tasks"**.
- Jug allows simple management of *race conditions* where dependencies exist between tasks.
- Jug gives us a simple way of farming out our workflows onto processing clusters (such as LOTUS).

But clearly you could write it yourself...if you want.

Further information

Jug:

http://pythonhosted.org/Jug/