Benchmarking OPeNDAP services for modern ESM data workloads

Stephen Pascoe (Stephen.Pascoe@stfc.ac.uk)
Richard Wilkinson (Tessella plc. Abingdon, UK)
Phil Kershaw (Philip.Kershaw@stfc.ac.uk)
BADC: British Atmospheric Data Centre
NEODC: Near-Earth Observation Data Centre
IPCC-DDC: IPCC Data Distribution Centre
Opportunities for the “NetCDF/OPeNDAP Platform”
Key Questions

• What performance can we expect?
• Can OPeNDAP servers cope?
• How can we improve?
dapbench
OPeNDAP Test Framework

http://github.com/stephenpascoe/dapbench

- Intercept and records OPeNDAP requests from clients
- Load-test OPeNDAP servers
- Jython NetCDF abstraction layer
- Verifying ESG security constraints

- Using: Python, Jython, Grinder, Pandas, Matplotlib, Nmon
Client Tests

Test Clients

- CDAT (cdat-lite 6.0-alpha-3)
- CDO 1.4.7

Test Dataset

- 4D temperature field
  - shape [120,4,144,192] time/level/lat/lon
  - 51MB NetCDF file.

Test Requests

1. Take a 45°x45° subset of entire field
2. Regrid entire field to 100x100 lat/lon resolution
## Client Results

<table>
<thead>
<tr>
<th>Test</th>
<th>Tool</th>
<th>DODS Requests</th>
<th>Download Size (Mb)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset</td>
<td>CDO</td>
<td>481</td>
<td>50</td>
<td>1 + time*level requests</td>
</tr>
<tr>
<td></td>
<td>CDAT</td>
<td>17281</td>
<td>1.5</td>
<td>1 + time<em>level</em>lat requests</td>
</tr>
<tr>
<td>Regrid</td>
<td>CDO</td>
<td>481</td>
<td>50</td>
<td>1 + time*level requests</td>
</tr>
<tr>
<td></td>
<td>CDAT</td>
<td>69121</td>
<td>50</td>
<td>1 + time<em>level</em>lat requests</td>
</tr>
</tbody>
</table>

CDO downloads too much and with sub-optimal number of requests. CDAT downloads just enough but with very sub-optimal number of requests. CDAT iterates over all outer dimensions.
ESGF Security testing

$ dapbench-thredds <cmd> <options>

• Detect OPeNDAP URLs from THREDDS URL or URL list
• Test access to all responses
  • .das, .dods, .dds, .ascii
• Confirms authz redirect correctness
• Useful for C.I. Testing
Server Tests

<table>
<thead>
<tr>
<th>Test Framework</th>
<th>Test Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Grinder load-testing framework</td>
<td>• 30 files of 4D temperature field</td>
</tr>
<tr>
<td>• NetCDF-Java API</td>
<td>• shape ([120,4,144,192]) time/level/lat/lon</td>
</tr>
<tr>
<td>• Jython NetCDF abstraction layer</td>
<td>• 600MB per NetCDF file.</td>
</tr>
<tr>
<td>• Nmon server monitoring</td>
<td></td>
</tr>
</tbody>
</table>

**Test Requests**

1. Download a single file (non-dap response)
2. Request all of a random file in \(n\) slices, \(n = [15, 30, 60, 120, 240, 720, 1440]\)
3. Continuously request random subsets in \(m\) threads, \(m = [1, .. 40]\)
Test System

Measured Bandwidth: 930Mbps (iperf)

- DELL optiplex 980.
- Intel i5 4-cores @ 3.2GHz
- 1Gbps NIC
- NetCDF 4.1.2-beta2, HDF5-1.8.4-patch1

- Xen Virtual Machine
- Host: DELL PowerEdge 2950 III
  - 2x quad-core CPU, 24GB RAM
- Guest VM: Paravirtualised
  OpenSuSE 11.0
  - 2 CPU, 8GB RAM, 4GB swap

**Test Servers**

- THREDDS Data Server: 3.17.3.1
- Pydap: 3.0.rc.15. netCDF4-python-0.9.3.

**Platforms**

- 64-bit Java SDK 1.6.0-13, Tomcat-6.0.20.
  JAVA_OPTS=-Xms1000M -Xmn500M -Xmx1500M
- Apache-2.2.8 (prefork), mod_wsgi-3.3
Simulating WAN performance

$ tc qdisc add dev eth0 root netem delay <n>ms

<table>
<thead>
<tr>
<th>Host</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEDA</td>
<td>&lt;0.5</td>
</tr>
<tr>
<td>PCMDI</td>
<td>160</td>
</tr>
<tr>
<td>DKRZ</td>
<td>40</td>
</tr>
<tr>
<td>EGU</td>
<td>&gt;1000</td>
</tr>
</tbody>
</table>

Ha ha ha!
Results: ramp slices

![Ramp slices test graph]

- Time to download (s)
- Number of slices
- Lines represent different values of ramp slices:
  - (0, 'pydap')
  - (5, 'pydap')
  - (10, 'pydap')
  - (15, 'pydap')
  - (20, 'pydap')
  - (40, 'pydap')
  - (0, 'tds')
  - (5, 'tds')
  - (10, 'tds')
  - (15, 'tds')
  - (20, 'tds')
  - (40, 'tds')
Results: Sequential Download

Basic Download test

- 60 slices
- 30 slices
- 15 slices
- Direct download
  Including apache

Download time (s) vs. Server delay (ms)
Results: parallel requests

- Select Random File
- Select random time slices
- 4 Processes
- 60 Threads/Process
Results: Server load

0ms

MEM

CPU_ALL

NET

20ms

MEM

CPU_ALL

NET
How do we save clients from themselves?
DAP Response tiling / chunking

WMS web-clients use tiling to improve client & server performance
Conclusions

- High-level clients need optimising to use OPeNDAP effectively
  - or we need intelligent proxies / caches

- OPeNDAP servers can compete with whole-file download
  - If you minimise the number of requests

- Server implementations broadly competitive

- Effective caching needs to include clients and/or client-side proxies
Thanks

Stephen.Pascoe@stfc.ac.uk
Twitter: @spascoe
G+: Stephen Pascoe

http://github.com/stephenpascoe/dapbench