Fun with Grid Engine XML

Background

- This is a 15 minute talk - I’ll speak fast
- Talking about work published here:
  - http://xml-qstat.org
  - Simple web based SGE status dashboard
  - Not rocket science:
    - Transforms SGE qstat XML into useful XHTML

Demo

XML is good

- Initial misconception:
  - “*Structured output solves the parsing problem only*”
- The real benefit of Grid Engine XML:
  - *Transform* XML into *anything* you need
    - Textfiles, Complex PDFs, Speadsheets, SMS alerts, manpages, web pages, …
  - Can also search, parse, manipulate, etc.

Transforming XML

- Works like this
  1. Take any XML source
  2. Apply a XSL stylesheet
  3. Run both through a XSLT Engine
  4. Profit!

Buzzword Overkill

- XML
  - "*Extensible Markup Language*"
- XSLT
  - "*Extensible Stylesheet Language Transformation*"
- XPath
  - "*XML Path Language*"
- Web stuff
  - XHTML / DHTML
  - CSS
  - AJAX

XSLT & XPath

- Your source XML is transformed according to rules laid out in a XSLT stylesheet
- XPath “language”
  - Traverse XML document trees, parse data and run simple functions
- XPath statements are used within XSLT stylesheets

XSLT Example: Source

```
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family_name>Smith</family_name>
  </person>
  <person username="MI1">
    <name>Morka</name>
   <family_name>Ismincius</family_name>
  </person>
</persons>
```

Source: Wikipedia XSLT article

XSLT Example: XSL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="/persons">
        <html xmlns="http://www.w3.org/1999/xhtml">
        <head> <title>Testing XML Example</title> </head>
        <body>
                <h1>Persons</h1>
                <ul>
                <xsl:apply-templates select="person">
                        <xsl:sort select="family_name" />
                </xsl:apply-templates>
                </ul>
        </body>
        </html>
</xsl:template>

<xsl:template match="person">
        <li>
                <xsl:value-of select="family_name"/>,
                <xsl:value-of select="name"/>
        </li>
</xsl:template>

</xsl:stylesheet>
```

XSLT Example: XHTML Result

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head> <title>Testing XML Example</title> </head>
<body>
        <h1>Persons</h1>
        <ul>
           <li>Ismincius, Morka</li>
           <li>Smith, John</li>
        </ul>
</body>
</html>
```

Source: Wikipedia XSLT article

Grid Engine XSLT Examples

- Count number of pending jobs:

```
<xsl:variable
  name="pending_job_count"
  select="count(//job_info/job_list[@state='pending']/>
```

Grid Engine XSLT Examples

- Cluster wide slot utilization

```
<xsl:variable name="slotsUsed"
select="sum(//Queue-List/slots_used)"/>

<xsl:variable name="slotsTotal"
select="sum(//Queue-List/slots_total)"/>

  <xsl:variable name="slotsPercent"
select="($slotsUsed div $slotsTotal)*100" />
```

Grid Engine XSLT Examples

- Nested conditionals

```
<xsl:choose>
  <xsl:when test="//job_list[@state='pending']">
         <!-- HANDLE PENDING JOBS HERE -->
  </xsl:when>
  <xsl:otherwise>
         <!-- DO OTHER STUFF HERE -->
  </xsl:otherwise>
</xsl:choose>
```

Grid Engine XSLT Examples

- Better conditional example
  - Problem: queues in state “au” have no reported load_average. This was screwing up a HTML table
  - Embedded HTML shown in BLUE

```
<xsl:choose>
    <xsl:when test="load_avg">
     <!-- When load_avg is reported, output as usual -->
     <td class="boldcode">
         <xsl:value-of select="load_avg"/>
     </td>
    </xsl:when>
    <xsl:otherwise>
     <!-- otherwise, add a placeholder -->
     <td class="alarmcode">unknown</td>
    </xsl:otherwise>
</xsl:choose>
```

Rolling your own …

- Good online and paper references for XML, XSLT and XPath are all available
- There are *many* XSLT engine implementations …
  - Xalan (C++, Java)
  - XML:LibXSLT XML:LibXML (Perl)
  - Gnome libxml2 and libxslt (C)

Apache Cocoon

- http://cocoon.apache.org
    - xml-qstat (finally!) standardized on the Cocoon XML web publishing framework
    - Why Cocoon?
        - It just works
        - Most cross-platform solution I've found
            - Sole requirement: Java
        - Handles "web stuff" I don't want to care about
            - Cookies, URL handling, parameter passing, etc.

Apache Cocoon

- All in one XML publishing framework
  - XML and XSLT transformations handled internally
  - Just point it at your XML and your XSL stylesheet and it does the rest
- Easily drops behind Apache or Tomcat
- Web apps guided by a sensible configuration file
  - “sitemap.xmap”

Cocoon & SGE XML: qstat.html

- From the sitemap.xmap file ...

```
<!-- Standard View; CACHED Grid Engine XML data -->
<!-- URL will be "/xmlqstat/qstat.html"        -->

<map:match pattern="qstat.html">
  <map:generate src="xml/cached-sge-status.xml" />
      <map:transform src="xsl/qstat-xhtmlCSS-standard-v2.xsl">
          <map:parameter name="showQtable"  value="{cookie:displayQtable}" />
          <map:parameter name="enableResourceQueries" value="no" />
          <map:parameter name="real" value="yes" />
          <map:parameter name="perUserJobSort" value="no" />
       </map:transform>
   <map:serialize type="xhtml" />
</map:match>
```

Cocoon & SGE XML: RSS Feeds

- From the sitemap.xmap file …

```
<!-- ATOM XML FEED -->
<!-- URL pattern example:  /feed/overview   -->

<map:match pattern="feed/*">
   <map:generate src="xml/cached-sge-status.xml" />
     <map:transform src="xsl/feed-atom-{1}.xsl">
       <map:parameter name="ts1" value="{date:yyyy-MM-DD}"/>
       <map:parameter name="ts2” value="{date:HH:mm:ss}"/>
     </map:transform>
   <map:serialize type="atom-xml" />
</map:match>
```

Cocoon & SGE XML: Job view

- From the sitemap.xmap file

```
<!-- SGE JOB rendering for url path /job/<jobID>.html  -->

<map:match pattern="job/*.html">
<map:generate src="http://127.0.0.1:8889/xmldata/sgeJob?{1}" />
   <map:transform src="xsl/job-to-xhtml.xsl">
     <map:parameter name="jobID" value="{1}" />
   </map:transform>
 <map:serialize type="xhtml" />
</map:match>
```

Cocoon & SGE XML: Help!

- Look how we get “qstat -j <jobID>” data …

```
<map:generate src="http://127.0.0.1:8889/xmldata/sgeJob?{1}" />
```

- This is a lame way to get XML source data!
  - We have to access a CGI script located at URL “/xmldata/sgeJob” in order to get per-job data from qstat
  - Method works but is not elegant
  - We could do better …

Wanna help?

- I don’t know Java well at all …
- Apache Cocoon has:
  - “Cocoon Generators”
    - Java classes within Cocoon for natively generating/obtaining XML source
- Directly call “qstat” from within Cocoon
  - Dumb perl CGI’s no longer required!
- What I would love to have:
  - Custom Java Cocoon Generator for qstat calls
  - Capable of running the command:
    - “qstat -f -F -xml -j <jobID>”

Next steps for xml-qstat

- 1.0 release will occur
  - When Issue 2335 is fixed :)
  - Load average is not reported for “-xml”
- Modest changes will appear
  - Full XHTML Transitional validation
  - Full CSS validation
  - Start/stop scripts for Cocoon and XML cacher
- Possible extras
  - iPhone / Safari iPod detection and custom rendering

Lessons learned

- Handling Grid Engine XML is pretty easy
- XSLT and XPath have a somewhat steep learning curve
- I was wrong about xml-qstat
  - The XML processing is the easy bit
  - 90% of the hard work involves pure, traditional web design
    - XHTML, CSS, DHTML, JavaScript, etc.
    - I’m a geek, not a web designer

2 Off topic Announcements

ARCo on RAILS?

- xml-qstat largely a product of XML related night classes at Harvard Extention School
- I’m now enrolled in a “Web development with Ruby and Rails” class
  - Possible project idea:
    - Ruby-on-Rails web front end for ARCo databases
- Contact me if this interests you …

SGE on Amazon EC2

- BioTeam “skunkworks” project
  - Led by new hire ninja Mike Cariaso
    - an expert on Amazon EC2 “Elastic Compute Cloud”
- We already have MPI apps running inside the Amazon storage and compute clouds
  - Linux virtual server containers spun up on demand
  - Python and Windows GUI interfaces!
  - Real “on-demand” dynamic computing
  - Billing model allows people to make money via EC2
    - Example:
      - $.10/hr for Amazon, $.02/hr for BioTeam, $.02/hr for developer
      - Amazon handles all transactions

SGE on Amazon EC2

- Getting MPI and MPIBlast working in the cloud was non-trivial
  - Original MPICH2-EC2 work done by Peter Skomorch
  - Mike Cariaso built upon this work by porting MPI apps to EC2
- More info: http://runblast.com
- Next logical step
  - Develop EC2 images and scripts for dynamic creation and provisioning of arbitrary-sized SGE clusters within the Amazon compute and storage clouds
- That is all I can say at the moment :)
  - Expect whitepapers or announcements in 2008

End;

- Thanks!

- Questions / Contact
  - Chris Dagdigian
    - Personal: dag@sonsorol.org
    - Corporate: chris@bioteam.net