Workflow Management Through Cobalt

Gregor von Laszewski (1,2)
Christopher Grubbs (3),
Matthew Bone (3)
David Angulo (4)

gregor@mcs.anl.gov
http://www.mcs.anl.gov

(1) University of Chicago, Computation Institute, Research Institutes Building, #402, 5640 S. Ellis Ave., Chicago, IL 60637
(2) Argonne National Laboratory, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439
(3) Loyola University Chicago, Department of Computer Science, Lewis Towers, Suite 416, Water Tower Campus, Chicago, Illinois 60611
(3) DePaul University, School of Computer Science, Telecommunications and Information Systems, 243 South Wabash Ave, Chicago, Illinois 60604

http://wiki.cogkit.org/index.php/Java_CoG_Kit_Qstat
Acknowledgments

The REU was supported by NSF REU site grant 0353989. The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.
Introduction

- **Cluster & Batch Queuing System:**
  - enable use of parallel computers/ network of workstations by multiple users (within one administrative)

- **Metacomputer:**
  - integrating multiple clusters

- **Grids:**
  - enhance traditional view while shifting to multiple administrative domains, not only clusters.

- **Cyberinfrastructure for Science:**
  - provide the infrastructure to make scientific advances, not only Grids.
Queueing system revised

- Queueing software and systems (QS)
  - handles job execution on parallel systems,
  - schedules the jobs
  - allocates cluster resources accordingly
  - automation of scheduling process
  - monitoring
  - (some deal with dependencies, workflows)

- Grid interfaces to Queues
  - provides a useful subset of features provided by the QS
  - adaptation to new queuing system is possible
  - integration of custom features possible inspite of loss of generality
  - requires Grid software to be installed

gregor@mcs.anl.gov, http://www.mcs.anl.gov/~gregor
Motivation for the Effort

Scenario A: Can we use advanced features of the queuing system?
- Example: LSF provides dependencies between jobs
- We could integrate this in specialized versions of Grid software.
- Is this a loss of uniformity?
- Need a tool that is general and can integrate with all Grid resources

Scenario B: What is if there is no Grid software installed and can not be installed?
- Could this be simplified by installing Grid software?
- ANL BlueGene uses Cobalt, but does not (yet) run Globus
- NCSU has cluster for exclusive use for thread management
- Some sites have conditions that doe not allow easy installation of Grid software

- We need a tool that integrates Grid and non Grid resources leading to the cyberinfrastructure for scientists
Blue Gene/L at ANL

- **Hardware**
  - 1024 node = 2048 processor IBM BlueGene/L
  - Primarily for scientific computing, application porting
  - System software development
  - scalability studies
  - forerunner to ALCF

- **Software**
  - Cobalt Queuing system
  - Not Grid enabled
Cobalt

Simple Queuing system

Architecture

- Smaller and simpler is better philosophy,
- Trades on feature-richness for agility
- Core implementation is comprised of less than 4000 lines of code, mostly in Python
- Component-based architecture
- Highly adaptable and useful research platform
- Adding new components or rewrite existing ones is possible

Using Cobalt

- Login to BGL via SSH using public key authentication.
- cqstat - like qstat
- cqsub - like qsub (execution time, # of nodes requested, path to the executable, and so on)
- dynamic kernel selection (unique to Cobalt)
CoG Karajan

- One of several workflow tools in the CoG Kit
- Others include
  - APIs for Graphs and workflows
  - cog-shell
  - Replaceable scheduling strategies
- Architecture
  - Workflow Language
  - Workflow Engine
  - Part of a larger architecture framework where engine and language are replacable
CoG Karajan Language

- Functional Programming Language
  - Available as XML spec
  - Available as f( ) spec called “k”
  - originated from GridAnt, GECCO
- Features
  - CSP style programming
  - DAG style programming through library
  - Event based programming through exceptions and integration with CoG APIs
  - Control structure support: Loops, Conditions, ...
Features: CoG Karajan Language

<parallel>
<sequential>
<if>
<while>
Lists, Maps, Numerics, Boolean
Exceptions
XUL like forms
Java object and method bindings
Simple Variables

**Arithmetic**
- math:sum ::= a + b
- math:product ::= a * b
- math:subtraction ::= a - b
- math:quotient ::= a/b
- math:remainder
- math:square ::= a*2
- math:sqrt
- math:equals ::= a =b
- math:gt ::= a > b
- math:lt ::= a < b
- math:ge ::= a >= b
- math:le ::= a <= b

**Booleans**
- and, or, not, true, false

**Lists**
- list:append, list:prepend, list:concat,
- list:first, list:last, list:butFirst, list:butLast,
- list:size, list:isEmpty

**Map/Hashtable**
- map:put, map:entry, map:get, map:contains,
- map:delete, map:size
Datatypes

**Assignment**
<set name="var" value="1"/>

**Lists**
<list:append>
  <argument value="1"/>
  <argument value="2"/>
  <argument value="3"/>
</list:append>

**Operators**
<math:sum>
  <argument value="1"/>
  <argument value="2"/>
  <argument value="3"/>
</math:sum>

**Hashtable/map**
<set name="map">
  <map>
    <map:entry key="name" value="John"/>
    <map:entry key="age" value="99"/>
  </map>
</set>
<echo message="Name: {1}, age: {2}">
  <map:get map="{map}" key="name"/>
  <map:get map="{map}" key="age"/>
</echo>
Control Structures

- **Loops**
  
  ```xml
  <for name="iteration" from="1" to="4">
    <echo message="Iteration {iteration}"/>
  </for>
  
  <foreach name="iteration" in="one, two, three, four">
    <echo message="Iteration {iteration}"/>
  </foreach>
  
  <while>
    <condition>
      <false/>
    </condition>
    <echo message="You will never see this message"/>
  </while>
  
  <if>
    <condition>
      <false/>
    </condition>
    <then>
      ...
    </then>
    <else>
      ...
    </else>
  </if>
  ``
<project>
  <include file="cogkit.xml"/>
  <element name="date" arguments="host">
    <execute executable="/bin/date"
      stdout="date-{host}" host="{host}" provider="GT4"/>
    <echo message="Job on {host} completed. Transferring output"/>
    <transfer srchost="{host}" srcfile="date-{host}""
      desthost="localhost" provider="gridftp"/>
    <echo message="Output from {host} transferred"/>
    <set name="date">
      <readFile file="date-{host}"/>
    </set>
    <echo message="The date on {host} is {date}"/>
  </element>
  <parallel>
    <date host="hot.mcs.anl.gov"/>
    <date host="sunny.mcs.anl.gov"/>
  </parallel>
</project>
Explicit CSP Like Parallelism

<sequential>
  <element1/>
  <element2/>
  <element3/>
</sequential>

<parallel>
  <element1/>
  <element2/>
  <element3/>
</parallel>

<parallel>
  <element1/>
  <element2 sync="false"/>
  <element3/>
</parallel>
import("cogkit.k")
import("dag.k")
discard(
    dag("a"
         node([], print("A+"), wait(delay=10000), print("A-"))
         edges("b", "c")
    )
    node("b"
         element([], print("B+"), wait(delay=20000), print("B-"))
         edges("d")
    )
    node("c"
         element([], print("C+"), wait(delay=20000), print("C-"))
         edges("d")
    )
    node("d"
         element([], print("D+"), wait(delay=10000), print("D-"))
    )
    for(i, range(1, 10)
        node(
            i
            element([], print("X+"), wait(delay=10000), print("X-"))
        )
    )
)
On demand XML GUI Gorms

```xml
<form:form title="test" id="form" waitOn="IDOk">
  <form:vbox>
    <form:hbox>
      <form:vbox>
        <form:hbox>
          <form:label text="First name: "/>
          <form:textField id="IDFirst" columns="20"/>
        </form:hbox>
        <form:hbox>
          <form:label text="Last name: "/>
          <form:textField id="IDLast" columns="20"/>
        </form:hbox>
      </form:hbox>
      <form:vbox>
        <form:checkBox caption="Married" id="IDMarried" halign="0"/>
        <form:HSeparator/>
        <form:radioBox caption="Sex" id="IDSex">
          <form:radioButton caption="Male" id="IDMale"/>
          <form:radioButton caption="Female" id="IDFemale"/>
        </form:radioBox>
      </form:vbox>
      <form:button id="IDOk" caption="Ok"/>
    </form:hbox>
  </form:vbox>
</form:form>
```
<onError match="\(.*\text{Expired credentials detected.}.*)\|\(.*\text{Proxy file.*not found.}.*)\)">
  <if>
    <condition> <!-- avoid recursive errors -->
      <math:equals value1="\{errorcount\}" value2="1"/>
    </condition>
    <then>
      <!-- pop up a proxy init window -->
      <echo message="Invalid GSI credentials detected. Executing proxy init..."/>
      <executeJava mainClass="org.globus.cog.karajan.util.ProxyInitWrapper"/>
      <!-- re-execute the element -->
      <echo message="Restarting failed element"/>
      <executeElement element="\{element\}"/>
    </then>
    <else>
      <!-- "Error count > 1" -->
      <!-- error produced by generateError are never intercepted -->
      <generateError message="\{error\}"/>
    </else>
  </if>
</onError>
Executing Jobs on The Grid

```xml
<project>
  <include file="cogkit.xml"/>
  <execute executable="/bin/date"
               stdout="thedate"
               host="hot.mcs.anl.gov" provider="GT2"/>
  <echo message="Job completed. Transferring the output"/>
  <transfer srchost="hot.mcs.anl.gov" srcfile="thedate"
            desthost="localhost" provider="gridftp"/>
  <echo message="Transfer complete"/>
  <set name="date">
    <readFile file="thedate"/>
  </set>
  <echo message="The date is \{date\}"/>
</project>
```
Features: CoG Karajan Uses CoG Grid Abstractions

- Authentication, remote task execution, file transfers,
- Access to CoGs task execution framework which employs the CoG Kit's task/provider model.
- Programmatic interface while the underlying provider (which may be GT2, GT4, SSH, etc.)
- Can be useful for runtime mappings
- Provides high level syntax/abstraction
- Adding new providers is a structured process
- Not easily allows special features because of limitations in the abstractions (same as in Grid software)
Adding Cobalt support to CoG

Karajan

- We employ Karajan's Java binding functionality
- allows limited interfacing with Java classes, objects, and methods.
- We develop Java code that interfaces with the Cobalt queuing system
  - includes authentication through SSH
  - includes history management
  - includes status and submission
  - bypasses the restrictive older CoG abstraction layer while providing even more functionality
Reusing CoG Karajan Java Bindings

- Instatiation of Java Objects
  - <java:new>
  - referred class must be in classpath

- Invoking methods provided by the object
  - <java:invokeMethod>

- Example
Example Java Code Integration

```java
class Example {
    public myMethod (String arg0, int arg1)
}
```

gregor@mcs.anl.gov, http://www.mcs.anl.gov/~gregor
Java2Karajan Example

```
<element arguments="" name="example:initialize">
  <global name="_obj">
    <java:new classname="org.cogkit.q.Example"/>
  </global>
</element>

<element arguments="arg0, arg1" name="example:myMethod">
  <java:invokeMethod method="myMethod" object="{_obj}"
    types="int, java.lang.String">
    <argument value="{arg0}"/>
    <argument value="{arg1}"/>
  </java:invokeMethod>
</element>
```
Java2Karajan

- Example shows limitation
  - Integration of Java code requires some effort

- Solution: Java2Karaja
  - takes compiled Java class file
  - uses Java reflection to construct code automatically
Implementation

- CobaltSubmitter class
  - utilizes the SSH connection, authentication, and command execution functionality found in the open-source J2SSH SSHTools libraries
  - parameters the hostname, the username, the port, and the path to the user's private key.
  - submit()
  - status()

- Using Java2Karajan we derive a Karajan element that we can reuse
  - See next slide ...
Session: Using Cobalt in CoG

Karajan

<cobalt:initialize host="bgl.mcs.anl.gov" user="user"
  port="22"
  privatekeypath="/home/user/.ssh/id_rsa"/>

<parallel>
  <cobalt:execute walltime="30" nodes="32"
    path="/bin/job1" .../>

  <cobalt:execute walltime="60" nodes="64"
    path="/bin/job2" .../>
</parallel>

<cobalt:execute walltime="45" nodes="32"
  path="/bin/job3" .../>
Typical CoG Karajan Cobalt Session

- SSH Initialization
- SSH Authentication
- Open SSH session channel
- Execution of cqsub, returning the job's ID
- Close channel to the remote resource
Qstat Monitor: Observing the workflow

- Backend: We developed the Cobalt submission API
- Frontend: Monitor Table
  - Views status of the cqstat & more in a graphical component
  - Developed as Swing component
  - Distributed as API, SWING stand alone, and JNLP application
- Features:
  - Color-coded (running jobs are green, queued ones are yellow)
  - Allows users to customize which fields they would like to view.
  - Jobs History (not provided by cqstat)
  - Colored blue, providing the user with a record of completed jobs.
- Benefits:
  - Allows to observe the queue's status throughout the various stages of the workflow execution.
- Frontend: Job submission
  - Integrated in Qstat Monitor
  - Form based specification of jobs
Screenshot of the CoG Queue Monitor
Extensibility: Integration of PBS

- Test integration of native queuing system support
- We chose PBS
  - Available on UC/ANL TeraGrid
- Porting activities
  - Develop Java abstraction class for different handling of qsub command and its arguments
  - Creation of pbs:initialize and pbs:execute is simple through Java2Karajan
  - Integration of PBS in CoG Qstat monitor
- Security is handled via SSH
Possible Further Activities

- Integration of Karajan in Qstat Monitor
- Integration or reuse by Workflow editor(s)
- Exploit unique queuing system features such as dependencies
Conclusion

- Demonstrates effective way to extend Karajan
  - Java2Karajan
  - We develop now a CoG Shell based on the lessons learned
- Integrated Cobalt Queuing system
  - No Grid system necessary
  - Standard SSH
- Demonstrates Integration of Grid and non Grid resources
  - Increases diversity
  - Integrates Job History
The Java CoG Kit is available through its homepage at


A Java Web Start release of the Qstat Monitor is available at

http://wiki.cogkit.org/index.php/Java_CoG_Kit_Qstat

Code referred to in this paper can be found in the qstat section of the CoG Kit's Subversion repository, viewable at

http://svn.sourceforge.net/viewvc/cogkit/trunk/five/qstat/

Instructions for downloading the repository to Eclipse, using the Maven build system, can be found at

http://wiki.cogkit.org/index.php/MavenRepository