Introduction to Python Grid Tools

Keith R. Jackson
David E. Konerding
Lawrence Berkeley National Lab

DANSE Kickoff Meeting 2006
Outline

- Overview
- Pre-WS Grid Tools
- Web Service Tools
- Python WS Core
- Python Command Line Legacy Tool Wrapper
- Scientific Workflow
- Conclusions
Overview

- Why Python tools?
  - Simple, high-level language that is easy to learn
  - Good support for interacting with C, C++, Java, .Net, and Fortran codes
    - Many automated tools to make this process less tedious
  - Widely used in the scientific computing world
    - Python bindings already exist for a large number of common scientific packages
The Python CoG Kit provides a mapping between Python and the Globus Toolkit®. It extends the use of Globus by enabling to access advanced Python features such as exceptions and objects for Grid programming.

The Python CoG Kit is implemented as a series of Python extension modules that wrap the Globus C code.

Uses SWIG (http://www.swig.org) to help generate the interfaces.
Why Use pyGlobus?

- Provides a full interface to the pre-WS Globus Toolkit components
- Little/no changes are involved in switching between different versions of the Globus Toolkit
- High level language allows for easier Grid programming
- Supports rapid prototyping of Grid services/applications
- Many automated tools exist for exposing legacy C/C++ or Fortran codes as Python objects
Python Web Service Toolkit

• ZSI is an open-source toolkit for building web services and clients in Python
  – Added the support to ZSI that is needed for implementing the WSRF and WS-N families of specifications

• Supports the automatic generation of Python code from the WSDL/XML Schema interface definition using “wsdl2py”
  – Automatically maps XML Schema types to Python types
WS Tools (cont.)

- Provides a lightweight, asynchronous hosting environment, based on Twisted
- Extended code generation
  - Client calls look like methods
  - Schema data members look like instance properties
- Some security support
  - TLS for transport layer security
  - WS-Security Password Profile
  - Basic HTTP Authentication
  - Supports a standard authorization callout
Python WS Core

- Python WS Core, or, pyGridWare, implements the WSRF and WS-N family of specifications used to build stateful resources and services
- Provides APIs and tools for building WS-Resources (Web services that operate on stateful resources)
- Designed to be interoperable with current WSRF implementations
  - GT4 Java and C implementations
  - WSRF.NET
  - WSRF::Lite
Python WS Core

• Builds Python bindings from WSDL using ZSI for Web Services support
• Lifetime management
  – defines how and when services expire
• Security
  – Transport layer (https)
  – Secure Conversation
  – Secure Messaging
  – Authorization callout
• Interfaces to facilitate resource state persistence
• Notifications provide asynchronous messaging
pyGridWare Container

Client

Security Handler

Grid Service Container

Notification Producer

User-implemented service

Web container context

Lifetime Management

create

destroy

delegate

notify

request

response

persist

Persistent Resource

RP changes

DANSE Kickoff Meeting 2006
- We wanted to build a tool that exposed legacy applications as Grid services
- We targeted command-line applications initially
- Client tool is “just like” running command locally but run on a Grid server
- The client and server can be delivered as single executables on many platforms for easy deployment
- Based on ZSI & pyGridWare- code can be used as a learning example for building a Grid Service
PyCLST TODO

• Delegation - so service can do 3\textsuperscript{rd} party transfers
• Flexible, configurable input/output file transfer management, declared in the config file. User can override
• Flexible callout to auth modules (gridmap, SAML, etc)
PyCLST usage

- Developer writes a simple config file and runs PyCLST to build container, service app, and client app
- When invoked, the client app parses command line, wraps request to run command in a SOAP message
- The grid service executes the application; stdout and stderr are sent back immediately via WS-Notifications; client writes to user’s console
- Stdin, stdout, stderr and files sent in base64 encoded- GridFTP and other formats to be supported
### This example exposes “BLAST”

```python
[main]
name=blastpgp
executable=/usr/local/bin/blastpgp

## optionarguments start with one or two
## dashes, and may have an extra value
## argument
[optionarguments]
arg1option=-i
arg1desc=input
arg1hasvalue=True
arg1optional=False

arg2option=-d
arg2desc=database
arg2hasvalue=True
arg2optional=False

### “BLAST” takes no positional arguments

% python setup.py
```

Grid service template files ➔ BLAST CL-specific service
BLAST example timeline

Server:
% ./blastpgp-server.sh

Server runs BLAST

create

start

notify

notify

destroy

Client:
% ./blastpgp.sh -i {input.fa} -d db

Return code

stdout

stderr

DANSE Kickoff Meeting 2006
Visual Workflow Design and Execution

- Scientific workflows are becoming more complex, require more resource discovery, more data management
- Workflows have a strong parallel to visual programming systems
- We have created a Visual Computing Environment designed to make it easy to run complex workflows on the grid easily

DANSE Kickoff Meeting 2006
Logical separation of workflow functionality into a stack of services

**GUI** outputs abstract workflow

**ViCE**

**Planner**

**Condor DAGman, VDS, CBEI**

**Execution Manager**

**CBEI**

**Execution Context**

**CBEI, Condor, PBS, LSF, SGE, GT2.4 GRAM, GT4 GRAM, ssh**

**Physical resources**

**Planner converts abstract workflow into concrete execution plan**

**Execution manager submits jobs from the concrete execution plan according to their dependencies**

**Execution context manages job submission (or script execution) from birth to death and provides feedback to GUI and planner**

**Some planners can modify the execution plan once it has been started (adaptive optimization & fault recovery)**

DANSE Kickoff Meeting 2006
ViCE- design of a simple BLAST network
Conclusion

• The Python Web Service work is funded by the U.S. Department of Energy Office of Science
• ZSI
  – http://pywebsvcs.sourceforge.net/zsi.html
• pyGridWare
  – http://dsd.lbl.gov/gtg/projects/pyGridWare
• ViCE
• PythonCLServiceTool
  – http://dsd.lbl.gov/gtg/projects/PythonCLServiceTool
• Thanks to Michel Sanner for the idea of using a visual programming environment for workflow.