Everest: A Cloud Platform for Computational Web Services

Oleg Sukhoroslov, Alexander Afanasiev, Anton Rubtsov, Sergey Volkov

Institute for Information Transmission Problems (Moscow, Russia)
Agenda

- Scientific Application as a Service
- Service Model and Unified Interface
- Everest Architecture and Implementation
Scientific Application as a Service
Related Approaches

• Computational Grids
  – Globus Toolkit, gLite, UNICORE...
  – Generic web service interfaces to computing resources
  – Low-level, hard to use for unskilled researchers

• Scientific Portals
  – P-GRADE, HubZero, Galaxy...
  – Convenient web user interfaces to applications and computing resources
  – Do not expose applications as services or provide programming interfaces

• Web Service Toolkits
  – GEMLCA, Opal, MathCloud...
  – Tools for exposing scientific applications as web services
  – Ad-hoc, no common practices, require an infrastructure for hosting services
Everest

- Platform as a Service for development and hosting of scientific web services
  - Accessible via web browser
  - Supports multiple users
  - Enables users to create, share and access services
  - Integrates with external computing resources

- Combination of existing approaches + PaaS
  - Uniform service interface
  - Web UI for service development
  - Automatic generation of web UI for service invocation
Everest: A Cloud Platform for Computational Web Services (CLOSER 2014)
Service Model

```
{ "in1": ..., "in2": ..., "in3": ... }
```

```
{ "out1": ..., "out2": ... }
```

```
{ "name": ..., "inputs": { ... }, "outputs": { ... } }
```
# POV-Ray

## Inputs

<table>
<thead>
<tr>
<th>Title</th>
<th>Name</th>
<th>Type</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene file</td>
<td>scene</td>
<td>string</td>
<td><img src="http://bit.ly/PeI8kW" alt="Open" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INI file</td>
<td>ini</td>
<td>string</td>
<td><img src="http://bit.ly/PeI8kW" alt="Open" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Include files</td>
<td>includes</td>
<td>array[string]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output format</td>
<td>format</td>
<td>string</td>
<td><img src="#" alt="Available options" /></td>
<td>N8</td>
<td></td>
</tr>
<tr>
<td>Image width</td>
<td>width</td>
<td>integer</td>
<td>[1, 2048]</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>Image height</td>
<td>height</td>
<td>integer</td>
<td>[1, 2048]</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>Image quality</td>
<td>quality</td>
<td>integer</td>
<td>[0, 9]</td>
<td>9</td>
<td>0 = rough, 9 = full</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Title</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output image</td>
<td>image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>CPU utilization histogram</td>
<td>cpuHist</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>POV-Ray log</td>
<td>log</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
### Uniform Interface (REST API)

<table>
<thead>
<tr>
<th></th>
<th>GET</th>
<th>POST</th>
<th>PUT</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE</td>
<td>Get service description</td>
<td>Invoke service</td>
<td>Modify service</td>
<td>Delete service</td>
</tr>
<tr>
<td>JOB</td>
<td>Get job state/results</td>
<td>N/A</td>
<td>Modify job</td>
<td>Delete job</td>
</tr>
<tr>
<td>FILE</td>
<td>Download file</td>
<td>N/A</td>
<td>N/A</td>
<td>Delete file</td>
</tr>
<tr>
<td>SERVICES</td>
<td>List services</td>
<td>Create service</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FILES</td>
<td>List files</td>
<td>Upload file</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
POV-Ray

- **Scene file**: http://bit.ly/Pel8kW
- **Output format**: PNG 8 bits/color
- **Image width**: 320
- **Image height**: 240
- **Image quality**: 9

Request JSON

```json
{
    "inputs": {
        "scene": "http://bit.ly/Pel8kW",
        "includes": [],
        "format": "N16",
        "width": 320,
        "height": 240,
        "quality": 9
    }
}
```
Service Implementation

Authenticate and authorize client

Parse and validate input parameters

Translate input parameters to computational tasks

Run tasks on computing resources (stage-in, submit, monitor, stage-out)

Translate task results to output parameters

Computing Resources

Request

```
{ "in1": ..., "in2": ..., "in3": ... }
```

Task

```
executable arguments
input files
output files
resources
```

Request

```
{ "out1": ..., "out2": ... }
```

in1

in2

in3

out1

out2
**POV-Ray**

### Command

```
povray +Iscene.pov +F$\{format\} +W$\{width\} +H$\{height\} +Q$\{quality\} -D +HTN -Oimage
```

Refer to input values as `$\{param\}$

### Input Mappings

<table>
<thead>
<tr>
<th>Input</th>
<th>File</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>scene</td>
<td>scene.pov</td>
<td></td>
</tr>
<tr>
<td>ini</td>
<td>povray.ini</td>
<td></td>
</tr>
</tbody>
</table>

### Output Mappings

<table>
<thead>
<tr>
<th>Output</th>
<th>File</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>image.*</td>
<td></td>
</tr>
<tr>
<td>cpuHist</td>
<td>histgram.png</td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>stderr</td>
<td></td>
</tr>
</tbody>
</table>
**Everest Architecture**

- **Web UI**
- **Clients**
  - HTTPS + auth. token
  - Job request

**REST API**
authentication, /users, /groups, /services, /resources, /files, /jobs...

**Compute Bridge**
generate job tasks and process task results
- Single Task
- Parametric
- ???

**Data Storage**
store entities and files
- MongoDB

**Job Manager**
- Scheduling
- Task state

**Data Manager**
- Task data

**Scheduler**
- Task state

**Resource Manager**
- Tasks

**Computing Resources**
Integration with Computing Resources

Everest

- Agent Connection
- Data Manager

WebSocket
message exchange

HTTP(S)
data transfer

Compute Agent

- Agent Protocol Client
- Task Handlers
  - Local
  - TORQUE
  - SLURM
  - ???

Cluster
Mapping Services to Resources

- **Service A**
  - Connected to **Resource 1**
  - Connected to **Resource 2**
  - Connected to **Resource 3**

- **Service B**
  - Connected to **Resource 2**
  - Connected to **Resource 3**

Override by user
Python API

```python
session = everest.Session(
    'https://mc2.distcomp.org',
    user = '...',
    password = '...' )

serviceA = everest.Service('52b1d2d13b...', session)
serviceB = everest.Service('...', session)
serviceC = everest.Service('...', session)
serviceD = everest.Service('...', session)

jobA = serviceA.run({'a': '...'})
jobB = serviceB.run({'b': jobA.output('out1')})
jobC = serviceC.run({'c': jobA.output('out2')})
jobD = serviceD.run({'d1': jobB.output('out'), 'd2': jobC.output('out')})

print(jobD.result())
```
Applications

• Personal services
  – Ubiquitous access to applications + resources
  – Automate repetitive tasks

• Sharing services with colleagues
  – Collaborative workflows
  – Publication of results
  – Reproducibility

• Education
Conclusion

- Cloud Platform for Computational Web Services
  - Uniform service interface
  - Use of PaaS model (Web UI + REST API)
  - Flexible mapping of services to external computing resources
  - Pilot deployment

- Future work
  - Experimental evaluation, application case studies
  - Task scheduling across multiple resources
  - Integration with grid infrastructures
  - Integration with scientific workflow systems
  - Optimization of data transfer
Thank you for your attention!

https://mc2.distcomp.org/ demo:demo
oleg.sukhoroslov@gmail.com