P*: A Model of Pilot-Abstractions

Andre Luckow, Mark Santcroos, Ole Weidner, Andre Merzky, Pradeep Mantha, Shantenu Jha

IEEE eScience 2012
Chicago, 11 October 2012
Towards a Common Model for Pilot-Jobs

Andre Luckow, Ole Weidner, Andre Merzky, Sharath Maddineni
Center for Computation and Technology
Louisiana State University
Baton Rouge, LA
{aluckow, oweidern, amerzky, smaddini}@cct.lsu.edu

Mark Santcroos
Bioinformatics Laboratory
Academic Medical Center
University of Amsterdam
Amsterdam, The Netherlands
m.a.santcroos@amc.uva.nl

Shantenu Jha
CAC/ECE
Rutgers University
Piscataway, NJ
shantenu.jha@rutgers.edu
IEEE eScience 2012

P*: A Model of Pilot-Abstractions

Andre Luckow\textsuperscript{1}, Mark Santcroos\textsuperscript{2,1}, Ole Weidner\textsuperscript{1}, Andre Merzky\textsuperscript{1}, Pradeep Mantha\textsuperscript{1}, Shantenu Jha\textsuperscript{3,1*}

\textsuperscript{1}Center for Computation & Technology, Louisiana State University, USA
\textsuperscript{2}Bioinformatics Laboratory, Academic Medical Center, University of Amsterdam, The Netherlands
\textsuperscript{3}Rutgers University, Piscataway, NJ 08854, USA

*Contact Author: shantenu.jha@rutgers.edu
NECS12 Workshop

Pilot Abstractions for Compute, Data, and Network

Mark Santcroos
Academic Medical Center
University of Amsterdam
The Netherlands
m.a.santcroos@amc.uva.nl

Silvia Delgado Olabriaga
Academic Medical Center
University of Amsterdam
The Netherlands
s.d.olabriaga@amc.uva.nl

Daniel S. Katz
University of Chicago &
Argonne National Laboratory
USA
d.katz@ieee.org

Shantenu Jha
Rutgers University
USA
shantenu.jha@rutgers.edu
Landscape of DCI and Applications: Empirical Assertions

- DCI is complex: Heterogeneous software, access-layers, policy..
- The space of possible Distributed Applications (DA) is large (and rich), but the number of effective and extensible DA small
- There are missing abstractions and poor implementations
  - Many local solutions, lack of end-to-solutions, especially tools
  - Missing conceptual frameworks and systems approach to tools
- Accept, if not embrace “distributedness”
  - Manage it, but also exploit it
  - The ability to reason: distributed performance, decomposition or aggregation!
Context

- Scientific computing applications demand (computer) resources to execute
Application

• Can be a simple script calling multiple instances of a program
• Can be a complex workflow management system that dynamically executes tasks based on a workflow description
• Resource requirements may vary over time
  – Internal dynamics of an application
  – Different solvers
  – Adaptive algorithms
Compute Resources

- Heterogeneous
  - Local clusters
  - High Throughput Computing Grids
  - High Performance Computing Clusters
  - Desktop Grids
  - IaaS (clouds)

- Capacity is fluctuating
  - Growing, shrinking
  - Changing in load and capabilities
Storage / Data Resources

- Local storage systems / data sources
- Different protocols / access mechanisms
- Distributed experiment data
- Data may be replicated
- Persistent / volatile
- Dynamic
Decoupling

Applications that are capable of using tools, abstractions and services that decouple payload management and resource assignment have been more successful at efficiently utilizing distributed resources.
Pilot Jobs

• A Pilot-Job is an abstraction that uses a placeholder job as a container for a set of compute tasks

• An instance of that placeholder job is commonly referred to as Pilot-Job or pilot
Pick your flavor …

- DIANE
- DIRAC
- CONDOR Glide-In
- Swift Coasters
- ToPoS
- Falkon
- PanDA
- Nimrod/G
- BigJob
- …
P*: a model

- A model for Pilot Abstraction
- Minimal but Complete
- Establish terminology
- Provide a framework for comparison
Positioning of P*

- Applications
- Tools
- Pilot-MapReduce
- Pilot-API
- P*
  - Pilot-Jobs
  - Pilot-Data
- SAGA
- Resource (Network, Compute & Data) Access Layer
- Physical Infrastructure
  - e.g. XSEDE, EGI, OSG, ESNet, Clouds
Pilot-Abstraction for Dynamic Distributed Data

- Similar heterogeneity for data infrastructure
- Application level capabilities on logical level and not on file level
- Fundamentally task placement is independent of data placement
- Dynamic decision for data
- Other considerations (varying data sources, data rates, etc)
Elements of the P* model

- Pilot-Compute (PC)
- Pilot-Data (PD)
- Compute Unit (CU)
- Data Unit (DU)
- Scheduling Unit (SU)
- Pilot-Manager (PM)
P* Model of Pilot Abstractions

1) submit pilot description
2) submit pilot
3) start pilot
4) submit CU
5) schedule SU to pilot
6) manage and schedule execution of SU

Application

Pilot-Manager

Resource Manager

Resource

Pilot

SU
CU
CU

SU
CU

SU
CU

Application

P*
Pilot-API

- Python implementation of P* model
- Exhibits the decoupling of Workload and Resources
NGS Workflow

- csFasta1
- csFasta2
- qual2
- qual1
- newTitle
- gridSplitDirectory1
- gridSplitDirectory2
- linesPerFile
- solid-to-fastqPE
- split-fastq1
- split-fastq2
- split-bean1
- split-bean2
- ReferenceTarGz
- ParametersTxt
- griddir-bam-files
- bwa-short-paired-reads
- ReferenceFastaGz
- merge-bam
- MergedBed
- MergedPileup
- MergedBai
- MergedFinalPileup
- MergedBam
Element: Pilot Compute(PC)

• The placeholder entity that gets submitted to a resource
• Often has the role of an agent:
  – collects information
  – manages the resources allocated
  – exchanges data
• Executes application code
# Instantiate a PilotComputeService running on localhost

```python
pilot_compute_service = PilotComputeService('localhost')
```

# Define a PilotComputeDescription

```python
pilot_compute_desc = {
    'service_url': 'cream://gb-ce-amc.amc.nl/cream-pbs-medium'
}
```

# Create a PilotCompute through the PilotComputeService

```python
pilot_compute_service.create_pilot(pilot_compute_desc)
```
Element: Pilot Data (PD)

• The placeholder entity that represents a storage resource (reservation)
• Can have the role of an agent:
  – collects information
  – manages the resources allocated
• Physically stores the data
PilotData Creation

# Instantiate a PilotDataService

```python
pilot_data_service = PilotDataService()
```

# Define a PilotDataDescription

```python
pilot_data_description = {
    'service_url': 'srm://tbn18.nikhef.nl/home/mark/pilotdata/'
}
```

# Create a PilotData

```python
pilot_data_service.create_pilot(pilot_data_description)
```
Element: Compute Unit (CU)

- Is defined by the application
- Encapsulates a self-contained piece of work that is submitted to the PJ framework
- E.g.:
  - task, job, rpc, web service call, etc.
# Define ComputeUnitDescription for solid-to-fastqPE

```python
compute_unit_description = {
    'executable': 'solid-to-fastqPE',
    'arguments': [<...APPLICATION_ARGUMENTS...>],
    'input_data': [solid_du],
    'output_data': [fwd_fastq_du, rev_fastq_du],
}
```

# Submitting solid-to-fastqPE ComputeUnit

```python
cd_service.submit_compute_unit
    (compute_unit_description)
```
Element: Data Unit (DU)

• Is defined by the application
• Encapsulates a self-contained piece of logical data that is submitted to the PJ framework
• E.g.:
  – file, chunk, database, etc.
Input DataUnit

# Define DataUnitDescription for short read files
solid_dud = {'file_urls': [
    'file:///Test42/Test42_F3.csfasta.bz2',
    'file:///Test42/Test42_F3_QV.qual.bz2',
    'file:///Test42/Test42_R3.csfasta.bz2',
    'file:///Test42/Test42_R3_QV.qual.bz2']}

# Submit the DataUnit, this will initiate a transfer.
solid_du = cd_service.submit_data_unit(solid_dud)
Output DataUnit

# DataUnit for forward fastq output file
data_unit_description = {
    'file_urls': ['Test42_fwd_read.fastq.gz']
}

fwd_fastq_du = DataUnit(data_unit_description)
Element: Scheduling Unit (SU)

• Represents a DataUnit / ComputeUnit internal to the runtime system
• Allows for aggregation and division of application level payload
# Mapping

<table>
<thead>
<tr>
<th>P* Element</th>
<th>BigJob</th>
<th>DIANE</th>
<th>Condor-G/ Glide-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pilot-Manager</td>
<td>BigJob Manager</td>
<td>RunMaster</td>
<td>condor_master</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>condor_collector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>condor_negotiator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>condor_schedd</td>
</tr>
<tr>
<td>Pilot</td>
<td>BigJob Agent</td>
<td>Worker Agent</td>
<td>condor_master</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>condor_startd</td>
</tr>
<tr>
<td>Compute Unit (CU)</td>
<td>Task</td>
<td>Task</td>
<td>Job</td>
</tr>
<tr>
<td>Scheduling (SU)</td>
<td>Unit</td>
<td>Sub-Job</td>
<td>Job</td>
</tr>
</tbody>
</table>
Three services

- **PilotComputeService**
  - Manages a pool of compute resources

- **PilotDataService**
  - Manages a pool of storage resources

- **ComputeDataService**
  - Manages the application’s workload
PilotComputeService

- **PilotComputeService** is responsible for creating and managing the PilotComputes.
- It is the application's interface to the Pilot-Manager in the P* Model.
- Created from **PilotComputeDescription**.
- Creates a **PilotCompute**.
PilotDataService

- **PilotDataService** is responsible for creating and managing the PilotDatas
- It is the application's interface to the Pilot-Manager in the P* Model
- Created from **PilotDataDescription**
- Creates a **PilotData**
The ComputeUnitDescription is a task description based on SAGA Job Description. It offers the application to describe a ComputeUnit in an abstract way that is dealt with by the Pilot-Manager.
# Instantiate a ComputeDataService

```python
cd_service = ComputeDataService()
```

# Connect PilotComputeService to ComputeDataService

```python
cd_service.add_pilot_compute_service(pilot_compute_service)
```

# Connect PilotDataService to ComputeDataService

```python
cd_service.add_pilot_data_service(pilot_data_service)
```

# Submitting solid-to-fastqPE ComputeUnit

```python
cd_service.submit_compute_unit(compute_unit_description)
```
## Characteristics and other properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>BigJob</th>
<th>DIANE</th>
<th>Condor-G/ Glide-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordination</td>
<td>M/W</td>
<td>M/W</td>
<td>M/W</td>
</tr>
<tr>
<td>Communication</td>
<td>Advert Service</td>
<td>CORBA</td>
<td>TCP</td>
</tr>
<tr>
<td>Scheduling</td>
<td>FIFO, custom</td>
<td>FIFO, custom</td>
<td>Matchmaking, priority-based scheduler</td>
</tr>
<tr>
<td>Agent Submission</td>
<td>API</td>
<td>GANGA Submission Script</td>
<td>Condor CLI</td>
</tr>
<tr>
<td>End User Environment</td>
<td>API</td>
<td>API and M/W Framework</td>
<td>CLI Tools</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Error propagation</td>
<td>Error propagation, retries</td>
<td>Error propagation, retries</td>
</tr>
<tr>
<td>Resource Abstraction</td>
<td>SAGA</td>
<td>GANGA/SAGA</td>
<td>Globus</td>
</tr>
<tr>
<td>Security</td>
<td>Multiple (GSI, Pass.)</td>
<td>Multiple (GSI)</td>
<td>Multiple (GSI, Kerberos)</td>
</tr>
</tbody>
</table>
PJ Framework Interoperability

- XSEDE:Trestles/FG (C1)
- EGI/FG (C2)
- XSEDE:QB/OSG (C3)

Runtime (in min)
Conclusions

• Established P* as an abstraction for supporting dynamic execution
• Common framework for comparisons
• Well defined Pilot-API for application developers
Discussion

• Applicability to workflow paradigm

• Extension with networking

• Clouds are an important and useful development but not a panacea
Acknowledgements

This work is funded by NSF CHE-1125332 (Cyber-enabled Discovery and Innovation), HPCOPS NSF-OCI 0710874 award, NSF-ExTENCI (OCI-1007115) and NIH Grant Number P20RR016456 from the NIH National Center For Research Resources
SJ acknowledges the e-Science Institute, Edinburgh for supporting the research theme. “Distributed Programming Abstractions” & 3DPAS
MS is sponsored by the program of BiG Grid, the Dutch e-Science Grid, which is financially supported by the Netherlands Organisation for Scientific Research, NWO
This work has also been made possible thanks to computer resources provided by TeraGrid TRAC award TG-MCB090174 (Jha) and BiG Grid
This document was developed with support from the US NSF under Grant No. 0910812 to Indiana University for “FutureGrid: An Experimental, High-Performance Grid Test-bed”