Complex Workloads on HUBzero – Pegasus Workflow Management System

Karan Vahi

Science Automation Technologies Group

USC Information Sciences Institute

USC Viterbi School of Engineering

Information Sciences Institute

HubZero

- **A valuable platform for scientific researchers**
  - For building analysis tools and sharing with researchers and educators.
  - Made available to the community via a web browser

- **Supports interfaces for**
  - Designing Analysis tools using the Rappture Toolkit
  - Uploading and creating inputs
  - Visualizing and plotting generated outputs

- **Supports hundreds of analysis tools and thousands of users.**

Hubzero - Scalability

- **Execution of the analysis tools for all users cannot be managed on the HubZero instance**
- **Need to decouple the analysis composition and user interaction layer from backend execution resources**
- **Scalability requires a need to support multiple types of execution backends**
    - Local Campus Cluster
    - DiaGrid
    - Distributed Computational Grids such as Open Science Grid
    - Computational Clouds like Amazon EC2

Distributing Analysis - Challenges

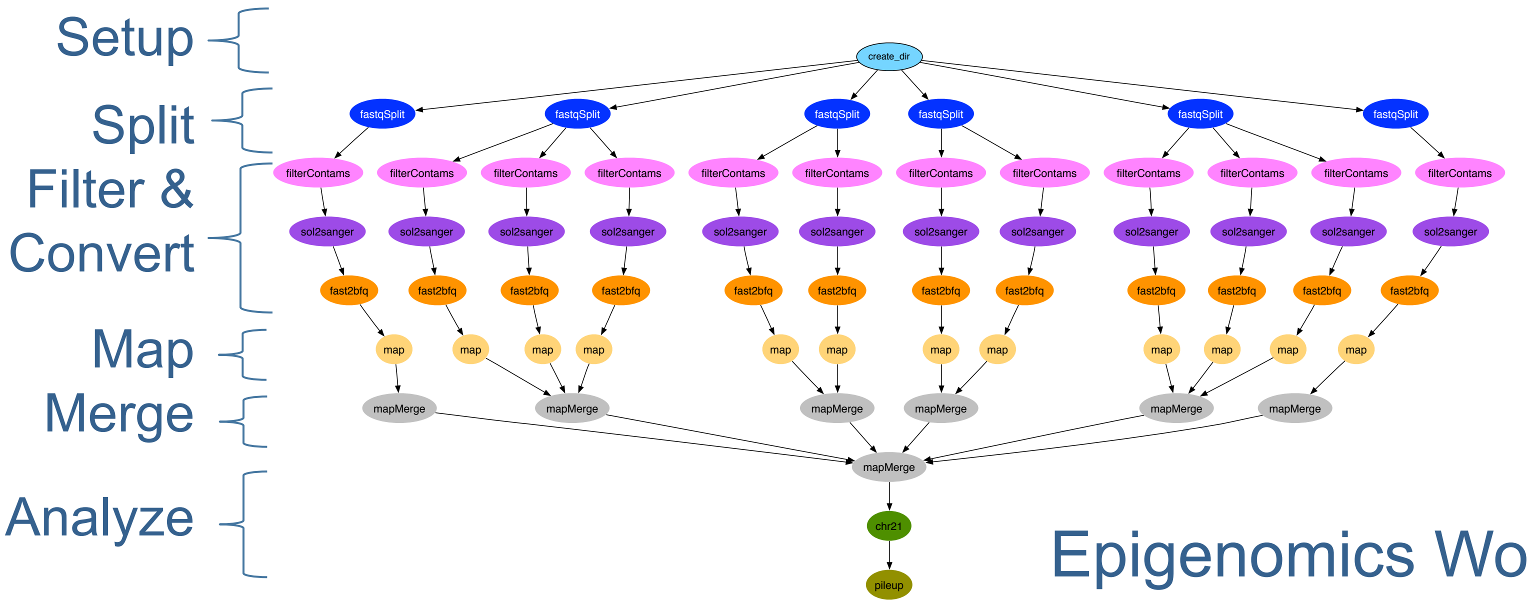
- **Portability**
  - Some Hubs are tied to local clusters. Others are connected to distributed computational grids. How do we get the analysis tool to run on local PBS cluster one day and OSG the next, or run across them.
- **Data Management**
  - How do you ship in the small/large amounts data required by the analysis tool?
  - You upload inputs via the web browser, but the analysis runs on a node in a cluster.
  - Different protocols for different sites: Can I use SRM? How about GridFTP? HTTP and Squid proxies?
- **Debug and Monitor Computations**
  - Users need automated tools to go through the log files
  - Need to correlate data across lots of log files
  - Need to know what host a job ran on and how it was invoked
- **Restructure Analysis Steps for Improved Performance**
  - Short running tasks or tightly coupled tasks
    - Run on local cluster a hub is connected to.
  - Data placement?

HubZero – Separation of concerns

- Focus on user interface and provide users
  - means to design, launch analysis steps and inspect and visualize outputs
- Model analysis tools as scientific workflows
- Use a Workflow Management System to manage computation across varied execution resources.

Scientific Workflows

- Orchestrate complex, multi-stage scientific computations
- Often expressed as directed acyclic graphs (DAGs)
- Capture analysis pipelines for sharing and reuse
- Can execute in parallel on distributed resources



Epigenomics Workflow

Why Scientific Workflows?

- Automate complex processing pipelines
- Support parallel, distributed computations
- Use existing codes, no rewrites
- Relatively simple to construct
- Reusable, aid reproducibility
- Can be shared with others
- Capture provenance of data

Pegasus Workflow Management System (WMS)

- **Under development since 2001**
- **A collaboration between USC/ISI and the Condor Team at UW Madison**
  - USC/ISI develops Pegasus
  - UW Madison develops DAGMan and Condor
- **Maps abstract workflows to diverse computing infrastructure**
  - Desktop, Condor Pool, HPC Cluster, Grid, Cloud
- **Actively used by many applications in a variety of domains**
  - Earth science, physics, astronomy, bioinformatics

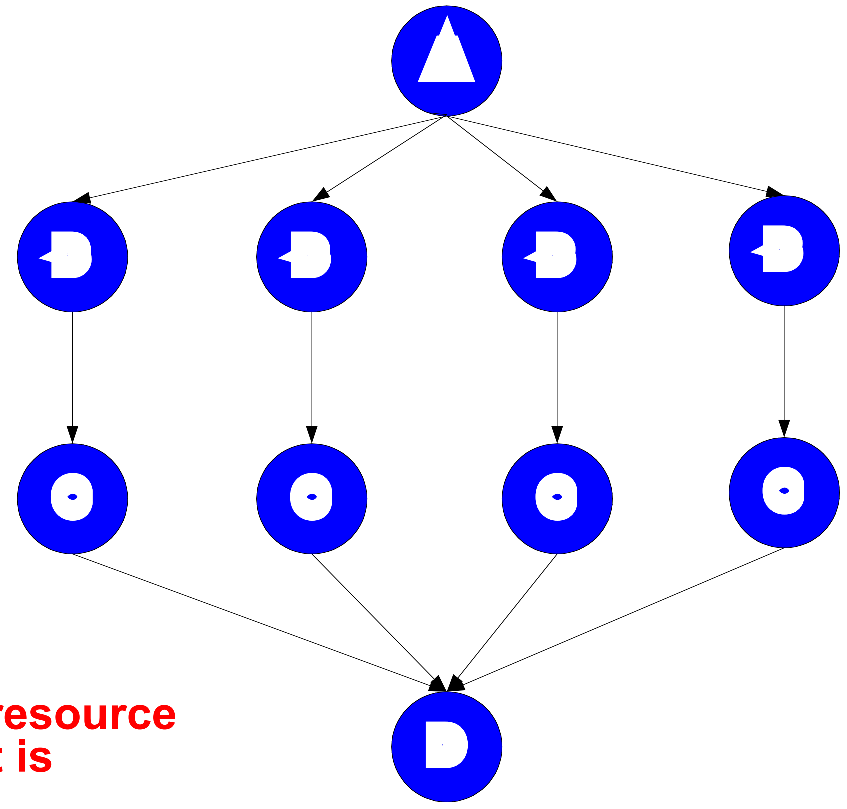
Benefits of workflows in the Hub

- **Clean separations for users/developers/operator**
  - **User**: Nice high level interface via Rappture
  - **Tool developer**: Only has to build/provide a description of the workflow (DAX)
  - **Hub operator**: Ties the Hub to an existing distributed computing infrastructure (DiaGrid, OSG, …)

- **The Hub Submit and Pegasus handle low level details**
  - Job scheduling to various execution environments
  - Data staging in a distributed environment
  - Job retries
  - Workflow analysis
  - Support for large workflows

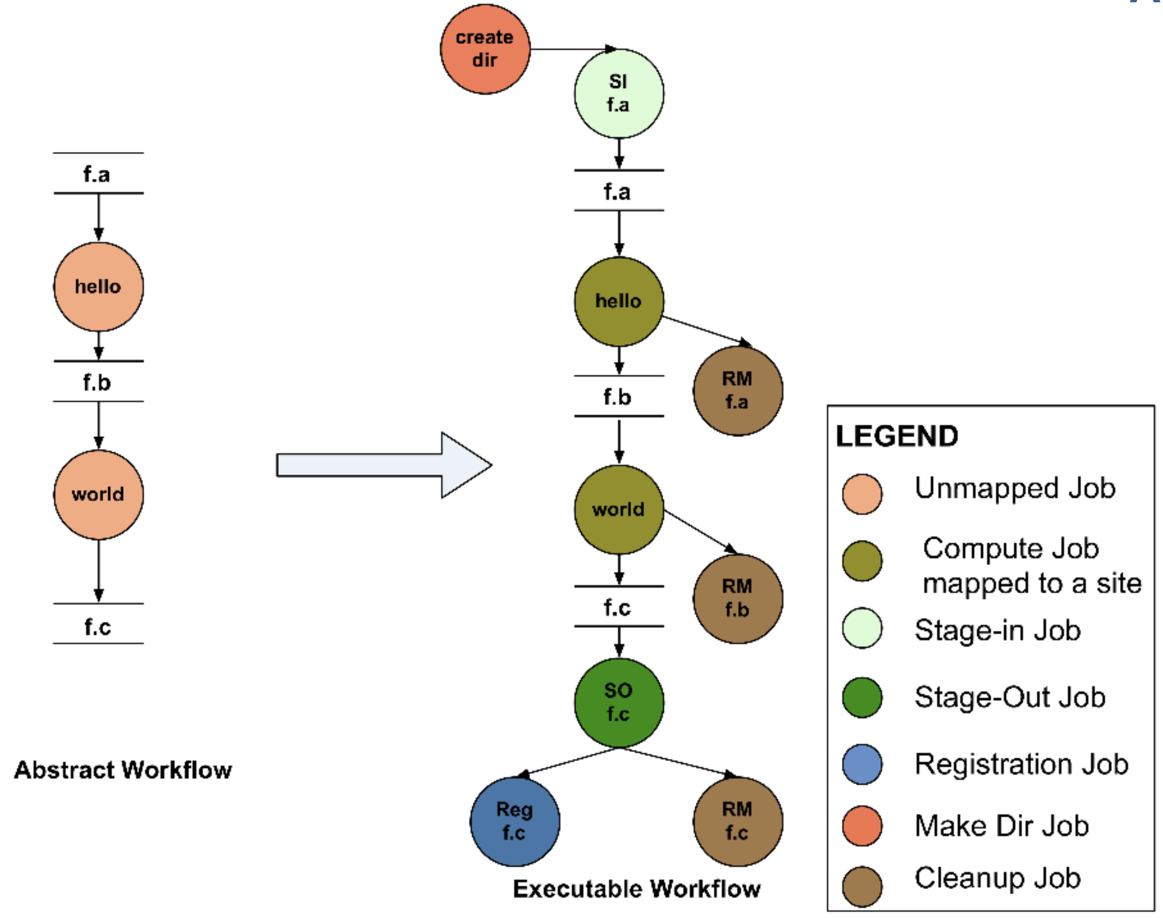
Pegasus Workflows are Directed Acyclic Graphs

- **Nodes are tasks**
  - Typically, executables with arguments.
  - Each executable identified by a unique logical identifier e.g. fft , date, fast_split
  - Nodes can also be other workflows
- **File Aware**
  - With each node you specify specify the input and output files referred to by logical identifiers.
- **Edges are dependencies**
  - Represent data flow
  - Can also be control dependencies
  - Pegasus can infer edges from data use
- **No loops, no branches**
  - Recursion is possible
  - Can generate workflows in a workflow
  - Can conditionally skip tasks with wrapper
- **Captures computational recipe, devoid of resource descriptions, devoid of data locations, that is portable and can be easily shared.**



Abstract to Executable Workflow Mapping

Pegasus compiles the Abstract Workflow to an Executable Workflow that can be executed on varied distributed execution environments



Abstraction provides

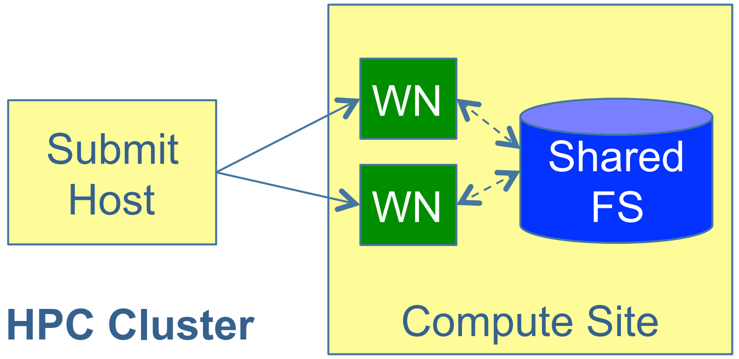
- **Ease of Use** (do not need to worry about low-level execution details)
- **Portability** (can use the same workflow description to run on a number of resources and/or across them)
- **Gives opportunities for optimization and fault tolerance**
  - automatically restructure the workflow
  - automatically provide fault recovery (retry, choose different resource)

Pegasus Guarantee - Wherever and whenever a job runs it's inputs will be in the directory where it is launched.

Supported Data Staging Approaches - I

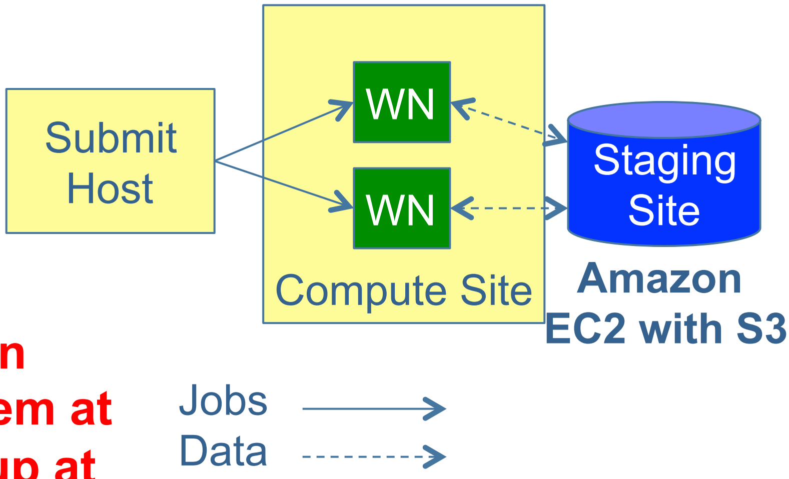
Shared Filesystem setup (typical of XSEDE and HPC sites)

- Worker nodes and the head node have a shared filesystem, usually a parallel filesystem with great I/O characteristics
- Can leverage symlinking against existing datasets
- Staging site is the shared-fs.



Non-shared filesystem setup with staging site (typical of OSG and EC 2)

- Worker nodes don't share a filesystem.
- Data is pulled from / pushed to the existing storage element.
- A separate staging site such as S3.

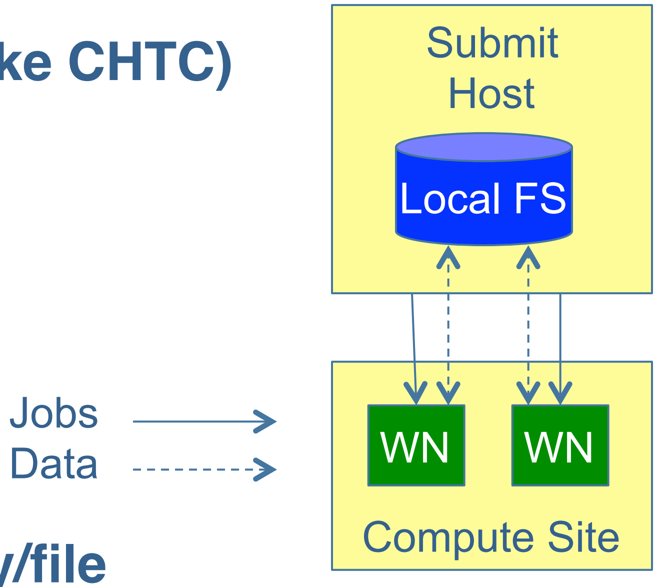


HubZero uses Pegasus to run a single application worklow across sites, leveraging shared filesystem at local PBS cluster and non shared filesystem setup at OSG!

Supported Data Staging Approaches - II

Condor IO (Typical of large Condor Pools like CHTC)

- Worker nodes don't share a filesystem
- Symlink against datasets available locally
- Data is pulled from / pushed to the submit host via Condor file transfers
- Staging site is the submit host.



Supported Transfer Protocols – for directory/file creation and removal, file transfers

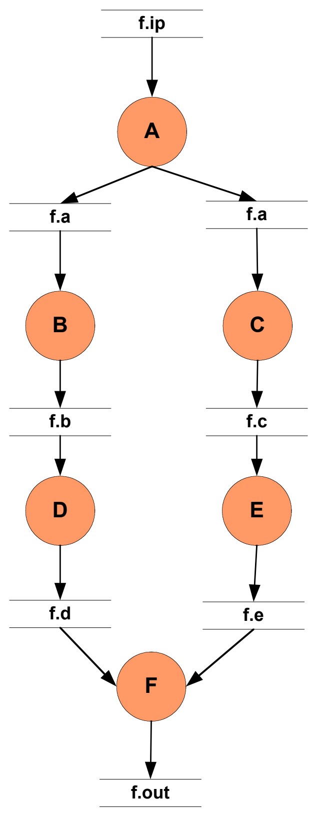
- HTTP
- SCP
- GridFTP
- IRODS
- S3 / Google Cloud Storage
- Condor File IO
- File Copy
- OSG Stash

Using Pegasus allows you to move from one deployment to another without changing the workflow description!

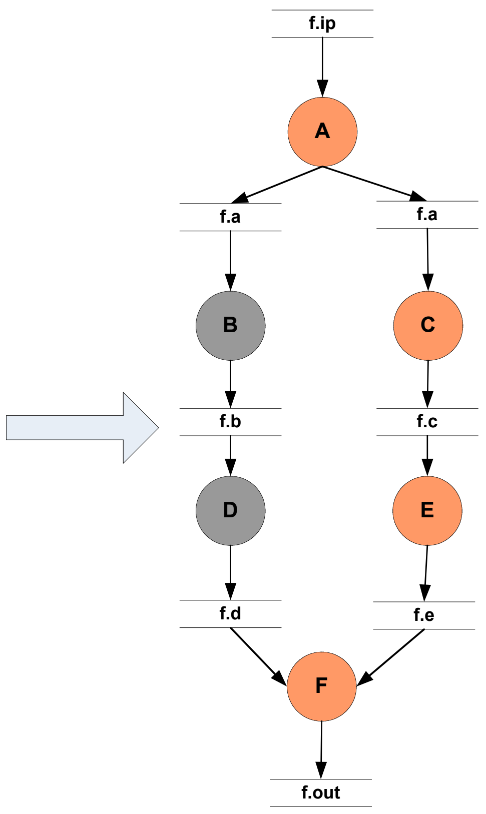
Pegasus Data Management Tools

pegasus-transfer, pegasus-create-dir, pegasus-cleanup support client discovery, parallel transfers, retries, and many other things to improve transfer performance and reliability

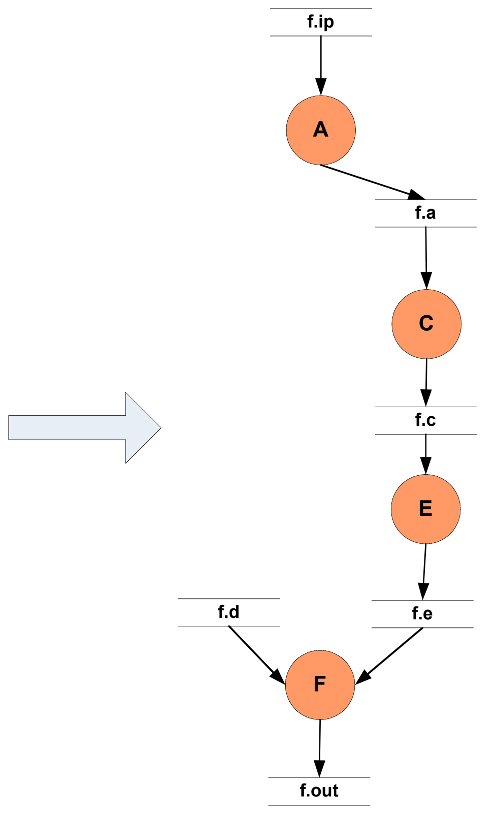
Workflow Reduction (Data Reuse)



Abstract Workflow



File f.d exists somewhere.
Reuse it.
Mark Jobs D and B to delete



Delete Job D and Job B

Useful when you have done a part of computation and then realize the need to change the structure.

File cleanup

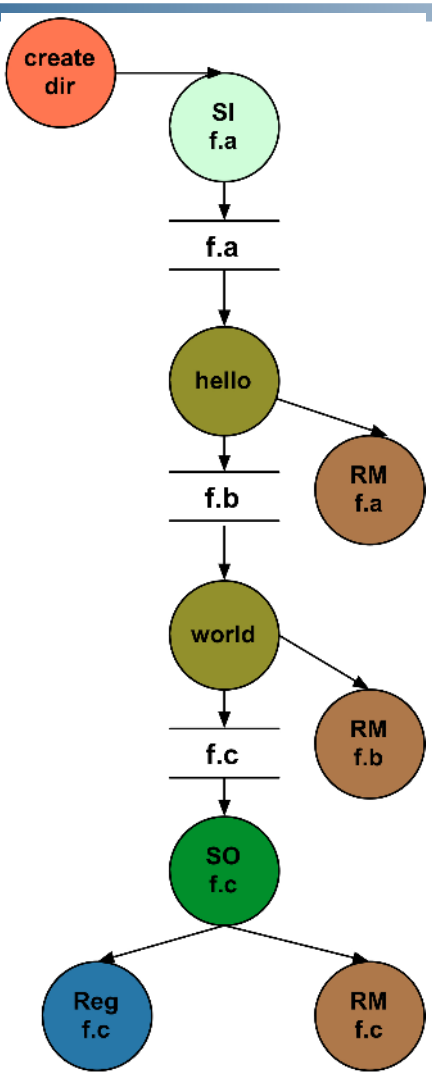
- **Problem: Running out of disk space during workflow execution**

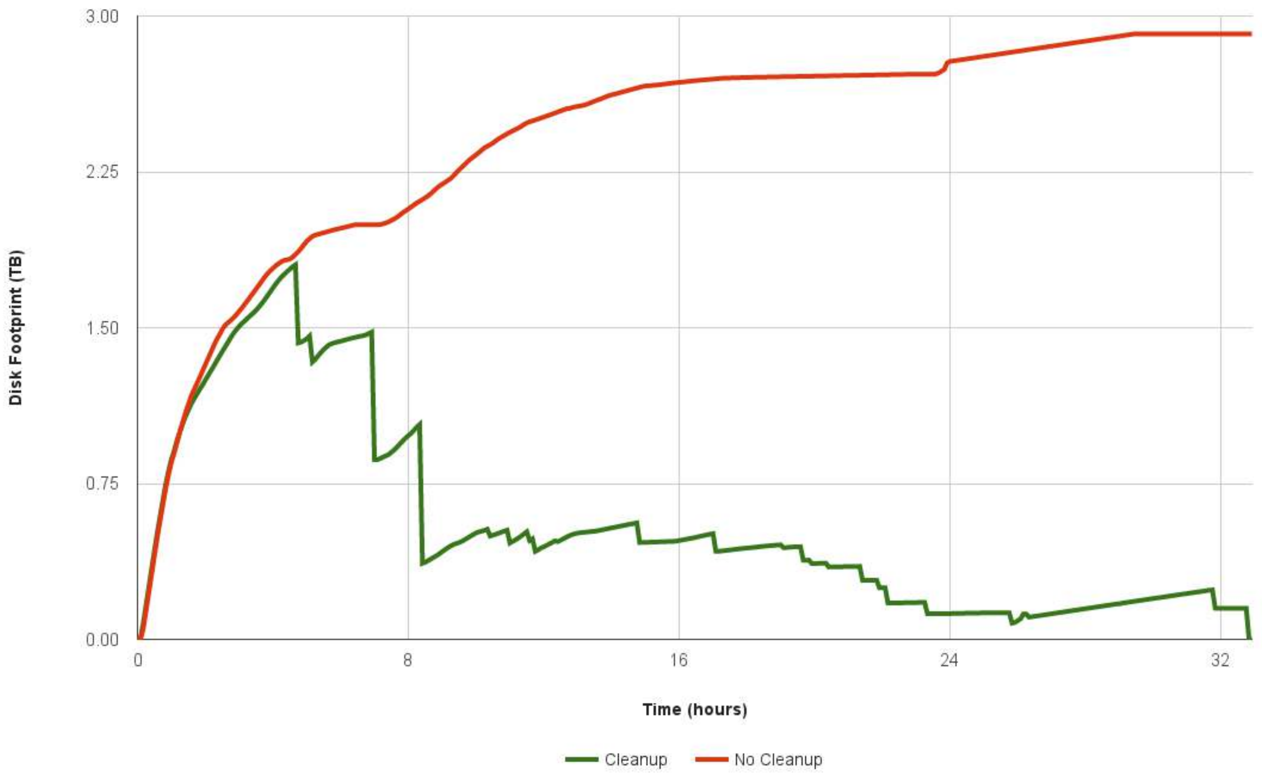
- **Why does it occur**
  - Workflows could bring in huge amounts of data
  - Data is generated during workflow execution
  - Users don't worry about cleaning up after they are done

- **Solution**
  - **Do cleanup after workflows finish**
    - Add a leaf Cleanup Job
  - **Interleave cleanup automatically during workflow execution.**
    - Requires an analysis of the workflow to determine, when a file is no longer required
  - **Cluster the cleanup jobs by level for large workflows**
  - In 4.6 release, users should be able to specify maximum disk space that should not be exceeded. Pegasus will restructure the workflow accordingly.

Real Life Example: Used by a UCLA genomics researcher to delete TB's of data automatically for long running workflows!!

File cleanup (cont)

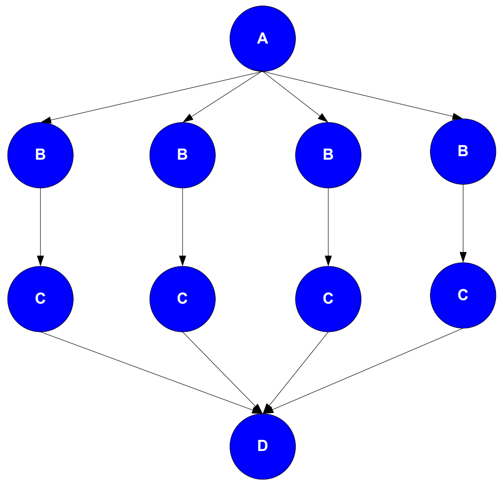


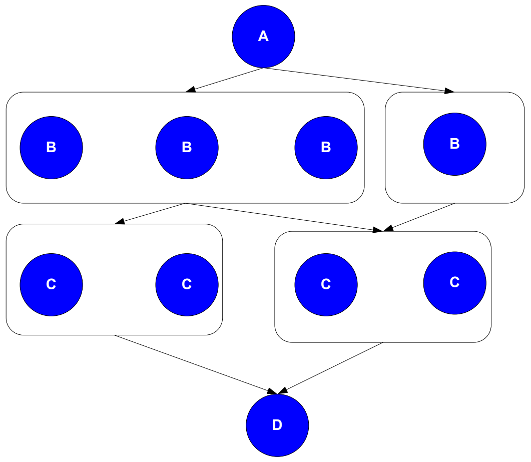


Single SoyKB NGS Pegasus Workflow with 10 input reads.

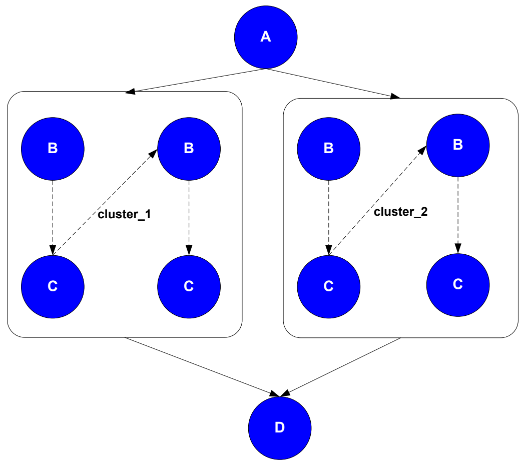
Workflow Restructuring to improve application performance

- **Cluster small running jobs together to achieve better performance**
- **Why?**
  - Each job has scheduling overhead – need to make this overhead worthwhile
  - Ideally users should run a job on the grid that takes at least 10/30/60/? minutes to execute
  - Clustered tasks can reuse common input data – less data transfers





Horizontal clustering



Label-based clustering

Workflow Monitoring - Stampede

- **Leverage Stampede Monitoring framework with DB backend**
  - Populates data at runtime. A background daemon monitors the logs files and populates information about the workflow to a database
  - Stores workflow structure, and runtime stats for each task.

- **Tools for querying the monitoring framework**
  - **pegasus-status**
    - Status of the workflow
  - **pegasus-statistics**
    - Detailed statistics about your finished workflow
  - **Integrated into Hub infrastructure via the submit integration**

```
------------------------------------------------------------------------------
Type           Succeeded Failed  Incomplete  Total     Retries   Total+Retries
Tasks          135002    0       0           135002    0         135002
Jobs           4529      0       0           4529      0         4529
Sub-Workflows  2         0       0           2         0         2
------------------------------------------------------------------------------

Workflow wall time                                : 13 hrs, 2 mins, (46973 secs)
Workflow cumulative job wall time                 : 384 days, 5 hrs, (33195705 secs)
Cumulative job walltime as seen from submit side : 384 days, 18 hrs, (33243709 secs)
```

Workflow Debugging Through Pegasus

- **After a workflow has completed, we can run pegasus-analyzer to analyze the workflow and provide a summary of the run**

- **pegasus-analyzer's output contains**
  - **a brief summary section**
    - showing how many jobs have succeeded
    - and how many have failed.
  - **For each failed job**
    - showing its last known state
    - exitcode
    - working directory
    - the location of its submit, output, and error files.
    - any stdout and stderr from the job.

Integrated with Submit . Alleviates the need for searching through lots of logs files and integrated with submit command.

Different Types of Analysis tools on HubZero

- **Parameter Sweep Analysis**
  - **Execute independent analysis steps on a set of input data, and merge the data and inspect it**
- **Sequential Analysis**
  - Execute analysis steps sequentially one after the other, with each step taking in the input of the previous steps.
- **DAG based analysis**
  - **Multistage analysis where each stage can be a single job or multiple stages**

Hubzero Pegasus Integration

How to generate the workflow

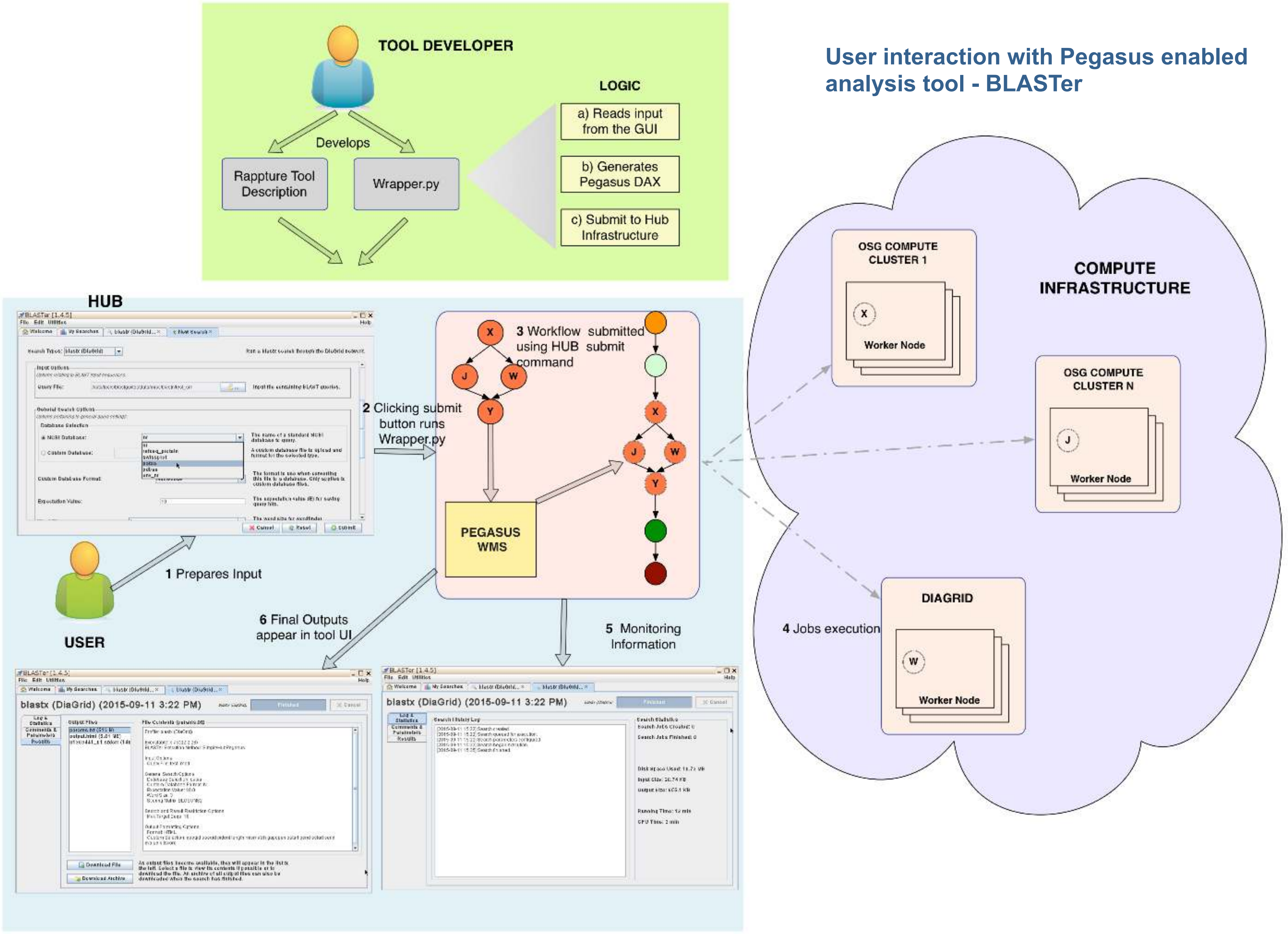
- **Tool description file tool.xml specifies a python run script (driver file) called by the Rappture interface when you hit the submit button**

  Two options on how to generate the DAX in python run script

  1. **Use the Pegasus Python DAX API to generate the DAX in the run script**
  2. **Call out to an external application-specific DAX generator**

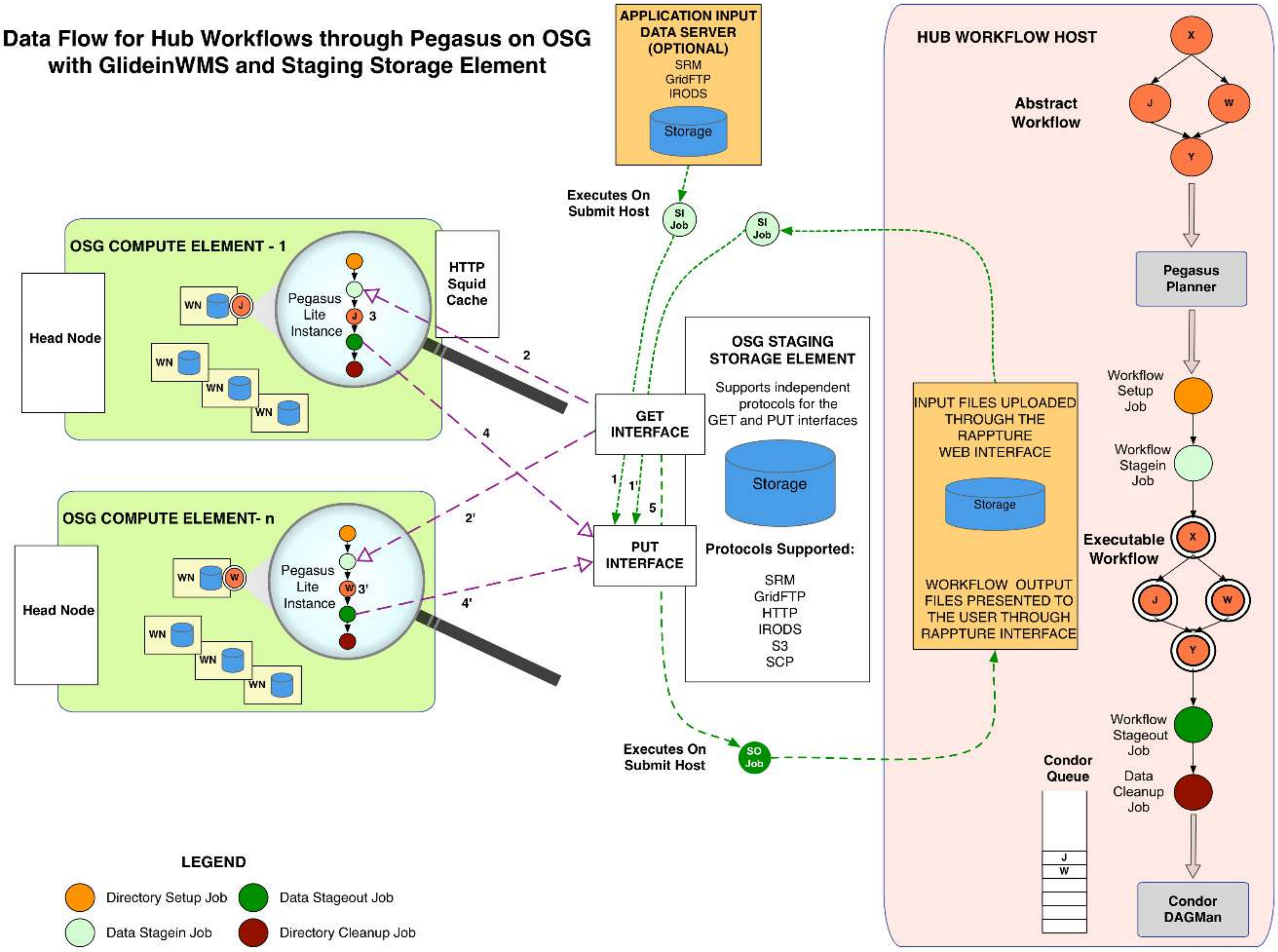
- **Compose the workflow programmatically and launch it using the Hub command line interfaces (Workspaces + submit).**

User interaction with Pegasus enabled analysis tool - BLASTer


TOOL DEVELOPER
Develops
Rappture Tool Description
Wrapper.py
LOGIC
a) Reads input from the GUI
b) Generates Pegasus DAX
c) Submit to Hub Infrastructure
HUB
1 Prepares Input
USER
2 Clicking submit button runs Wrapper.py
3 Workflow submitted using HUB submit command
PEGASUS WMS
X
J
W
Y
4 Jobs execution
5 Monitoring Information
6 Final Outputs appear in tool UI
blastx (DiaGrid) (2015-09-11 3:22 PM)
COMPUTE INFRASTRUCTURE
OSG COMPUTE CLUSTER 1
OSG COMPUTE CLUSTER N
DIAGRID
Worker Node

Acknowledgements: Steven Clark , Chris Thomson and Derrick Kearney, Purdue University

Data Flow for Hub Workflows through Pegasus on OSG with GlideinWMS and Staging Storage Element



Pegasus Tutorial tool now available in HUBZero

https://hubzero.org/tools/pegtut

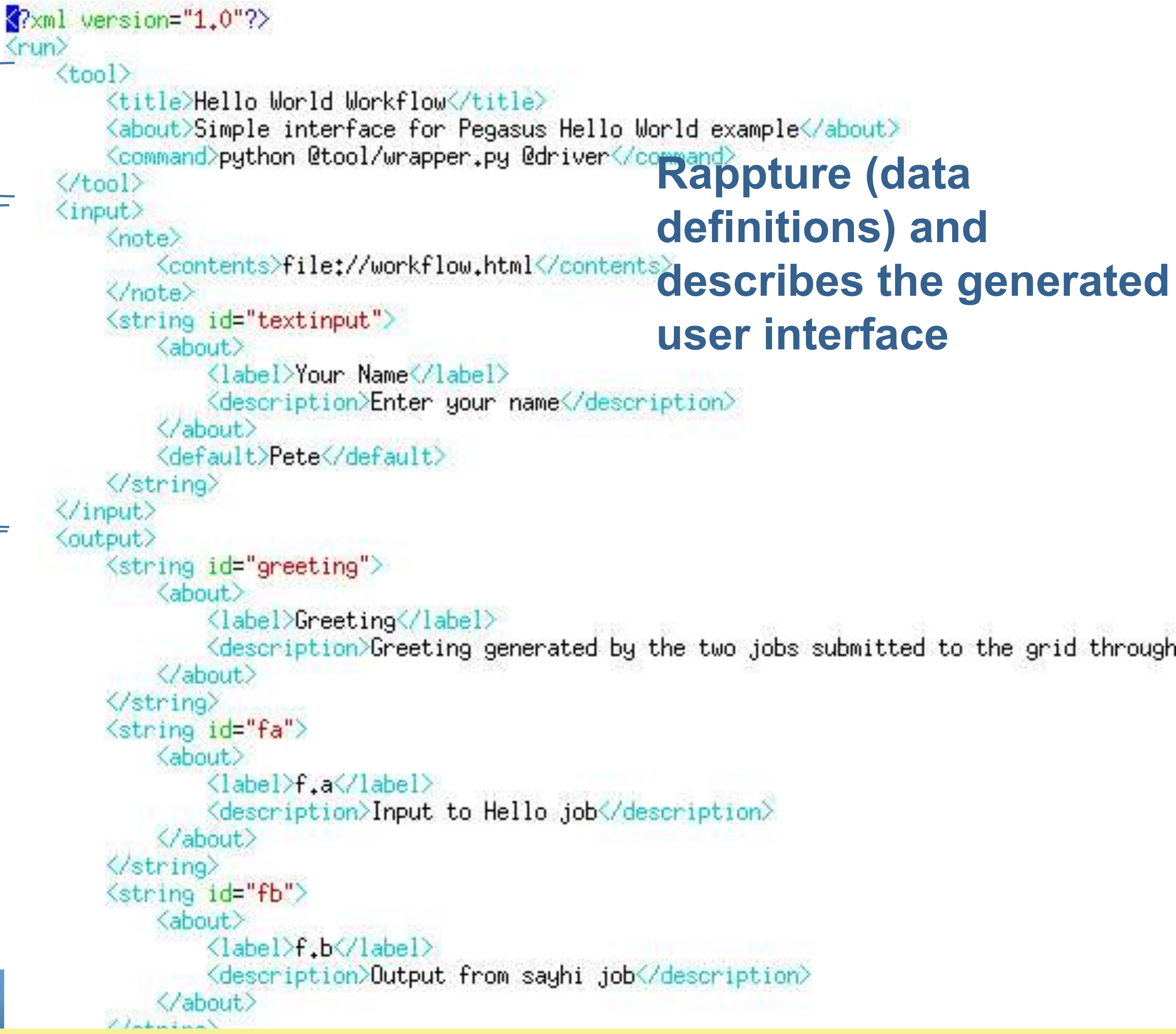
Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Tool description

Inputs

Outputs

Rappture (data definitions) and describes the generated user interface

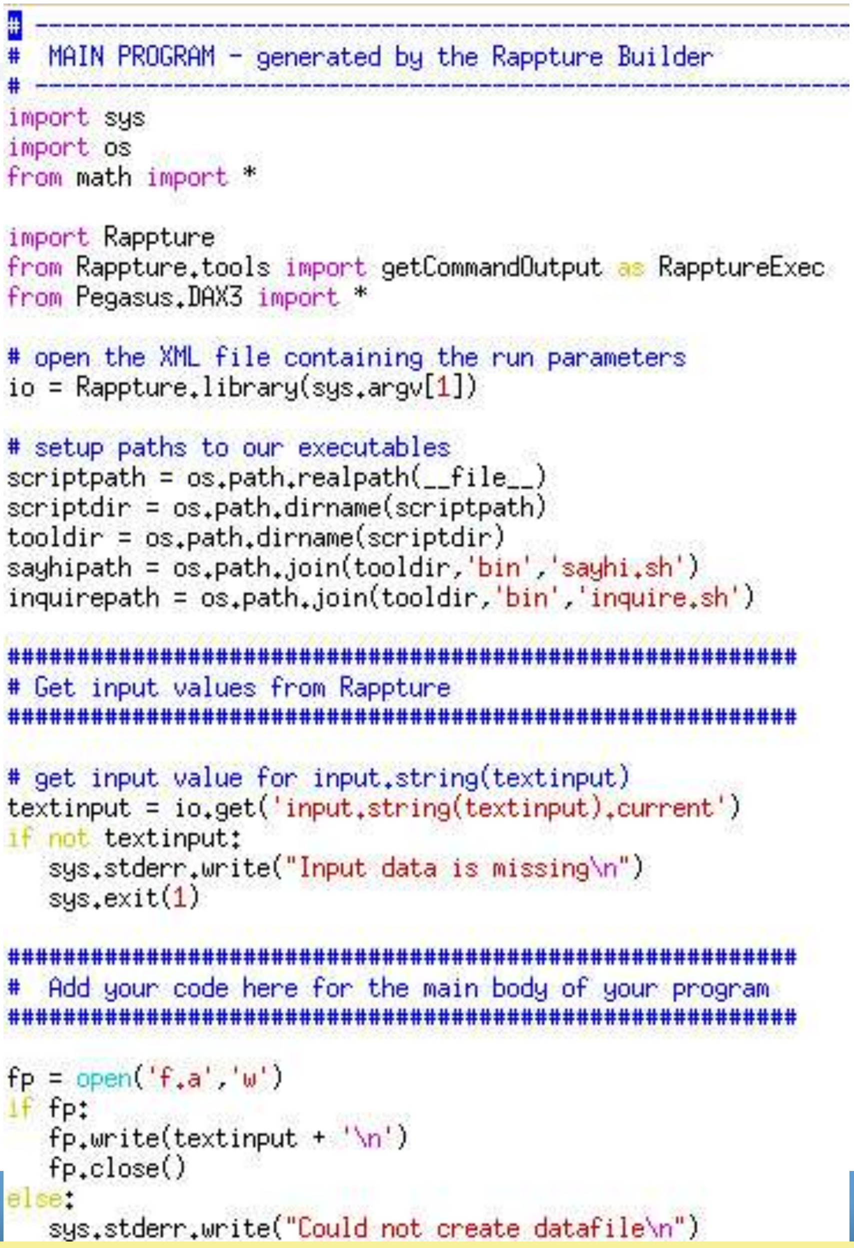


```
<?xml version="1.0"?>
<run>
    <tool>
        <title>Hello World Workflow</title>
        <about>Simple interface for Pegasus Hello World example</about>
        <command>python @tool/wrapper.py @driver</command>
    </tool>
    <input>
        <note>
            <contents>file://workflow.html</contents>
        </note>
        <string id="textinput">
            <about>
                <label>Your Name</label>
                <description>Enter your name</description>
            </about>
            <default>Pete</default>
        </string>
    </input>
    <output>
        <string id="greeting">
            <about>
                <label>Greeting</label>
                <description>Greeting generated by the two jobs submitted to the grid through
            </about>
        </string>
        <string id="fa">
            <about>
                <label>f.a</label>
                <description>Input to Hello job</description>
            </about>
        </string>
        <string id="fb">
            <about>
                <label>f.b</label>
                <description>Output from sayhi job</description>
            </about>
        </string>
```

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Wrapper.py

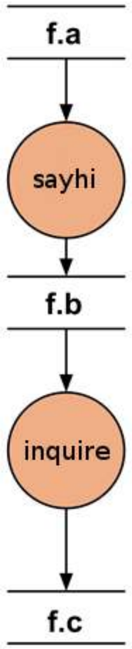
- Python script that is the glue between Rappture and user
- Collects the data from the Rappture interface
- Generates the DAX
- Runs the workflow
- Presents the outputs to Rappture



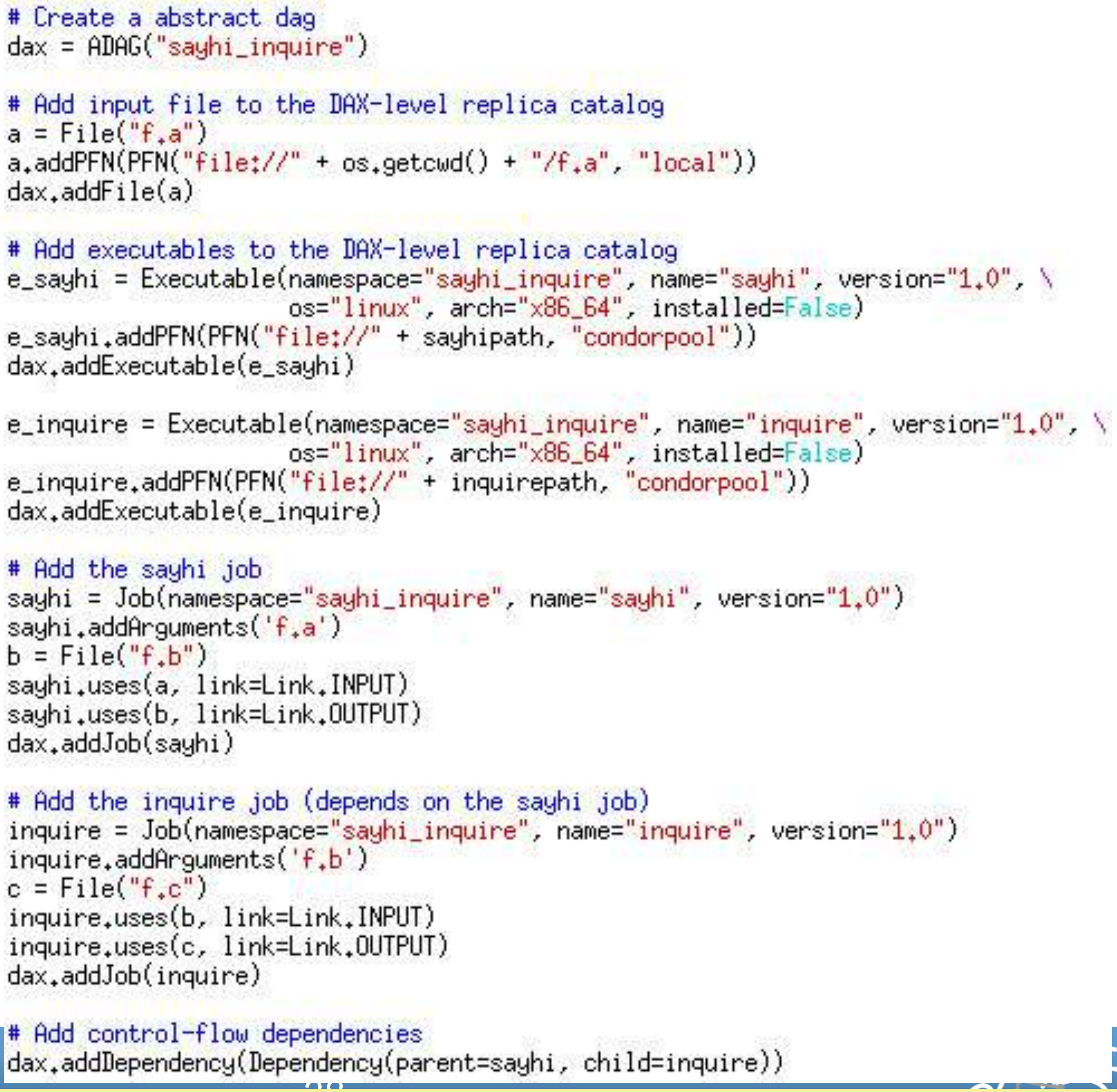
```
# ----------------------------------------------------------------------
#  MAIN PROGRAM - generated by the Rappture Builder
# ----------------------------------------------------------------------
import sys
import os
from math import *

import Rappture
from Rappture.tools import getCommandOutput as RapptureExec
from Pegasus.DAX3 import *

# open the XML file containing the run parameters
io = Rappture.library(sys.argv[1])

# setup paths to our executables
scriptpath = os.path.realpath(__file__)
scriptdir = os.path.dirname(scriptpath)
tooldir = os.path.dirname(scriptdir)
sayhipath = os.path.join(tooldir,'bin','sayhi.sh')
inquirepath = os.path.join(tooldir,'bin','inquire.sh')

#########################################################
# Get input values from Rappture
#########################################################

# get input value for input.string(textinput)
textinput = io.get('input.string(textinput).current')
if not textinput:
   sys.stderr.write("Input data is missing\n")
   sys.exit(1)

#########################################################
#  Add your code here for the main body of your program
#########################################################

fp = open('f.a','w')
if fp:
   fp.write(textinput + '\n')
   fp.close()
else:
   sys.stderr.write("Could not create datafile\n")
```

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Rappture (workflow definition)



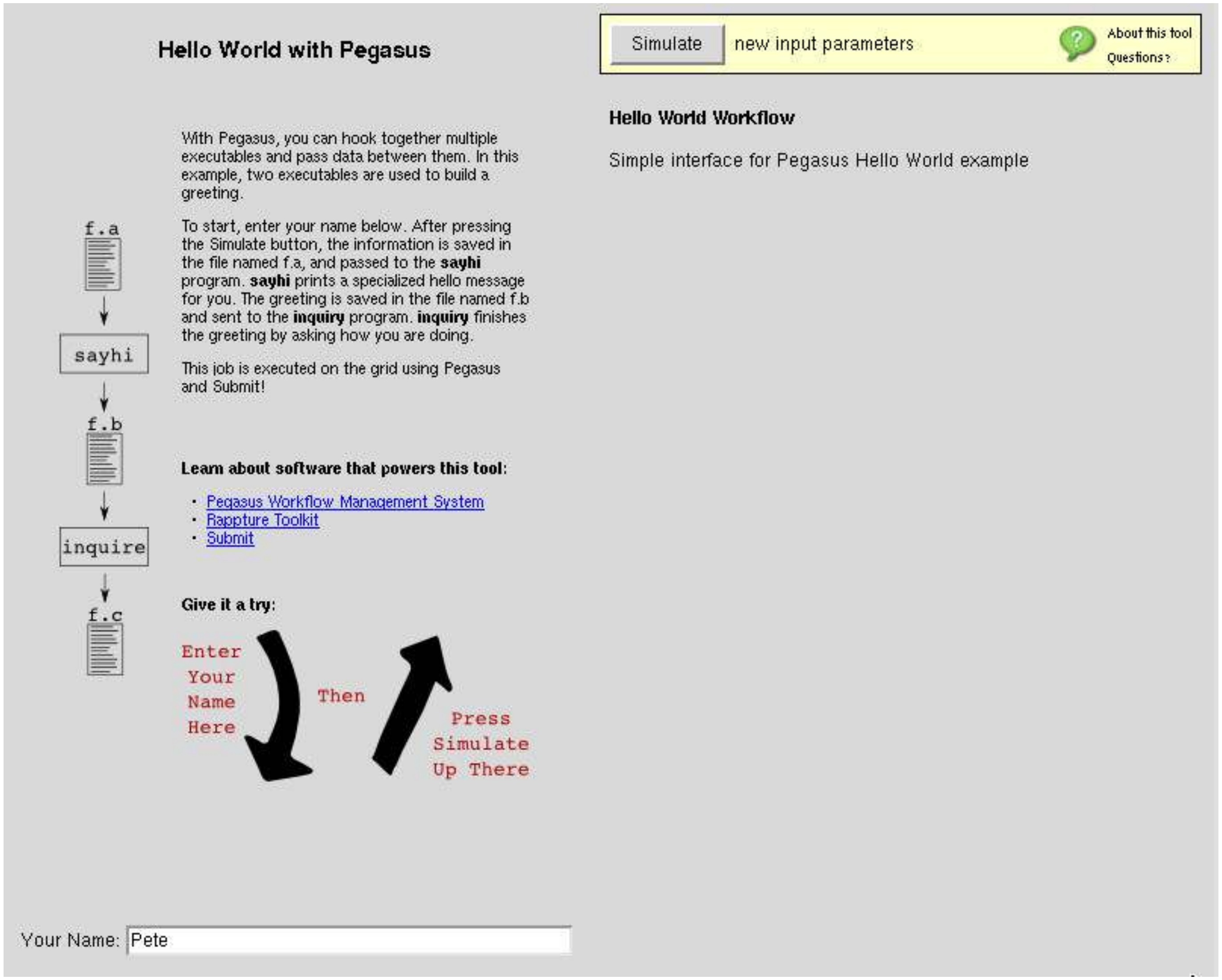
Abstract Workflow



```
# Create a abstract dag
dax = ADAG("sayhi_inquire")

# Add input file to the DAX-level replica catalog
a = File("f.a")
a.addPFN(PFN("file://" + os.getcwd() + "/f.a", "local"))
dax.addFile(a)

# Add executables to the DAX-level replica catalog
e_sayhi = Executable(namespace="sayhi_inquire", name="sayhi", version="1.0", \
                     os="linux", arch="x86_64", installed=False)
e_sayhi.addPFN(PFN("file://" + sayhipath, "condorpool"))
dax.addExecutable(e_sayhi)

e_inquire = Executable(namespace="sayhi_inquire", name="inquire", version="1.0", \
                     os="linux", arch="x86_64", installed=False)
e_inquire.addPFN(PFN("file://" + inquirepath, "condorpool"))
dax.addExecutable(e_inquire)

# Add the sayhi job
sayhi = Job(namespace="sayhi_inquire", name="sayhi", version="1.0")
sayhi.addArguments('f.a')
b = File("f.b")
sayhi.uses(a, link=Link.INPUT)
sayhi.uses(b, link=Link.OUTPUT)
dax.addJob(sayhi)

# Add the inquire job (depends on the sayhi job)
inquire = Job(namespace="sayhi_inquire", name="inquire", version="1.0")
inquire.addArguments('f.b')
c = File("f.c")
inquire.uses(b, link=Link.INPUT)
inquire.uses(c, link=Link.OUTPUT)
dax.addJob(inquire)

# Add control-flow dependencies
dax.addDependency(Dependency(parent=sayhi, child=inquire))
```

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

User provides inputs to the workflow and clicks the “Submit” button



Hello World with Pegasus

With Pegasus, you can hook together multiple executables and pass data between them. In this example, two executables are used to build a greeting.

To start, enter your name below. After pressing the Simulate button, the information is saved in the file named f.a, and passed to the **sayhi** program. **sayhi** prints a specialized hello message for you. The greeting is saved in the file named f.b and sent to the **inquiry** program. **inquiry** finishes the greeting by asking how you are doing.

This job is executed on the grid using Pegasus and Submit!

Learn about software that powers this tool:

- Pegasus Workflow Management System
- Rappture Toolkit
- Submit

Give it a try:

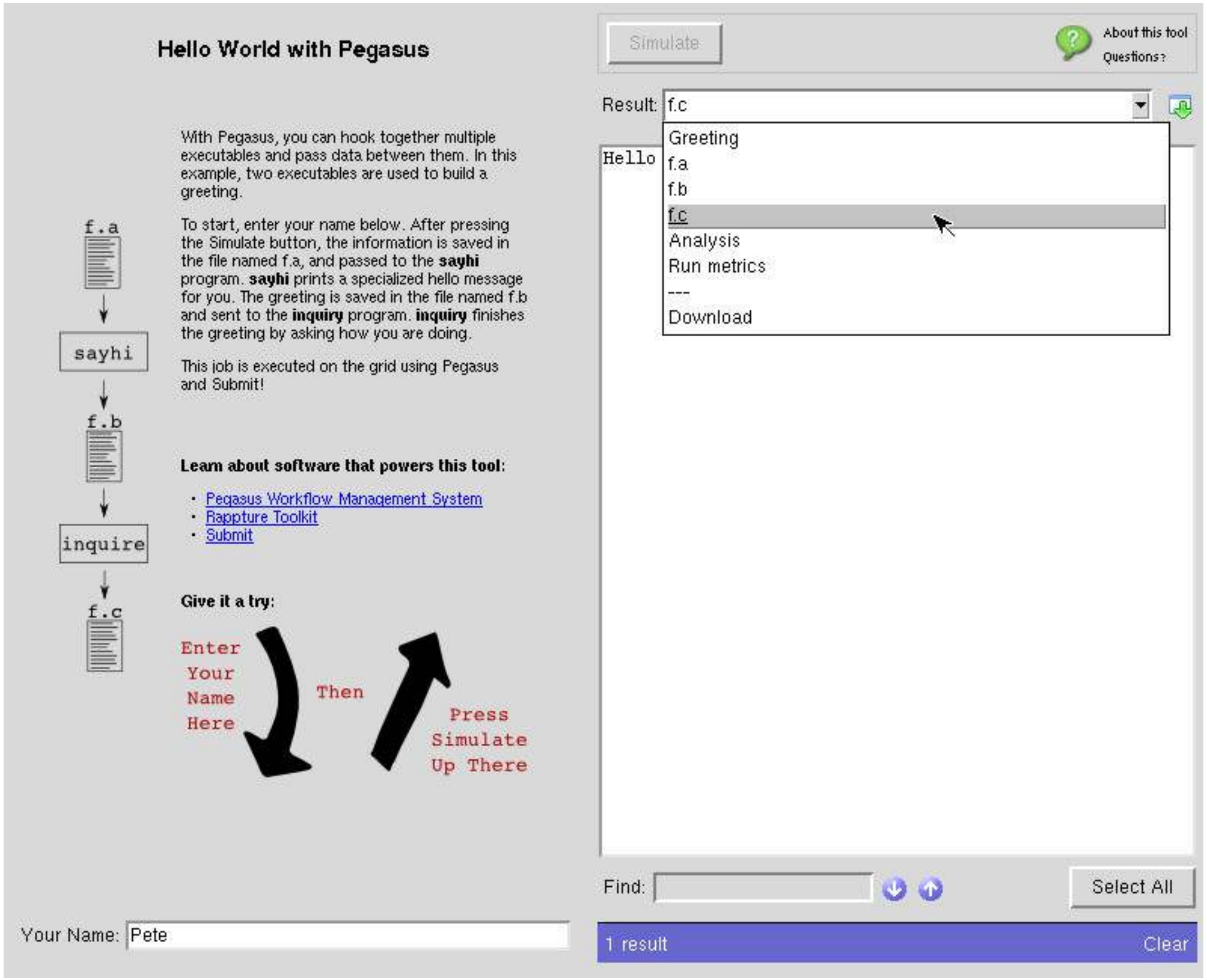
Your Name: Pete

Hello World Workflow

Simple interface for Pegasus Hello World example

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Workflow has completed. Outputs are available for browsing/downloading



Hello World with Pegasus

With Pegasus, you can hook together multiple executables and pass data between them. In this example, two executables are used to build a greeting.

To start, enter your name below. After pressing the Simulate button, the information is saved in the file named f.a, and passed to the **sayhi** program. **sayhi** prints a specialized hello message for you. The greeting is saved in the file named f.b and sent to the **inquiry** program. **inquiry** finishes the greeting by asking how you are doing.

This job is executed on the grid using Pegasus and Submit!

Learn about software that powers this tool:

- Pegasus Workflow Management System
- Rappture Toolkit
- Submit

Give it a try:

Your Name: Pete

Result: f.c

```
Hello
```

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Submit Command

- Used by Rappture interface to submit the workflow
- Submits the workflow through Pegasus to
  - OSG
  - DIAGRID
  - Local Cluster
- Prepares the site catalog and other configuration files for Pegasus
- Uses pegasus-status to track the workflow
- Generates statistics and report about job failures using pegasus tools.
  - Error reports in file pegasus.analysis
  - Job statistics and status pegasusjobstats.csv pegasusstatus.txt

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Examples of Pegasus Usage in Hubs

Pegasus on Hub Platforms

- **Using Workspaces**
  - Pegasus installed on all hubs
  - Users can directly submit and compose workflows through Pegasus and execute on OSG, DiaGrid.

- **Integrated into published tools on various Hubs**
  - Various tools in **NanoHub, DiaGrid, NEESHub**

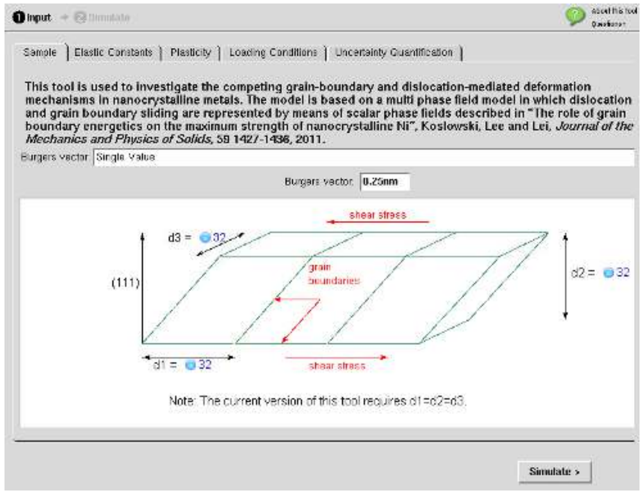
Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Pegasus on Hub Platforms

- **Using Workspaces**
  - Pegasus and submit installed on all hubs
  - Users can directly submit and compose workflows through Pegasus and execute on OSG, Diagrid.
- **Nanohub**
  - Perform parameter sweep on variables (CNTFET Lab, nanoFET, NanoPlasiticity)
  - Jobs are executed on Open Science Grid and DiaGrid
  - **CNTFET Lab -** Simulates using carbon nanotubes as field effect transistors
  - **nanoFET -** Simulates 2D mosfet devices.
  - **NanoPlasticity -** Investigates how nano-crystalline materials deform, includes uncertainty quantification.
- **DiaGrid**
  - **BLASTer** - online tool to run BLAST (Basic Local Alignment Search Tool) on the DiaGrid Hub.
  - **Cryo-EM** - Electron cryo-microscopy (cryo-EM) for 3-D structure of large macromolecular machines
  - **SubmitR** - Run R scripts on high-performance computing resources.
- **NEESHub**
  - **OpenSEES –** suite of simulation tools for submitting NEES scripts and for education and outreach.

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Pegasus on Hub Platforms



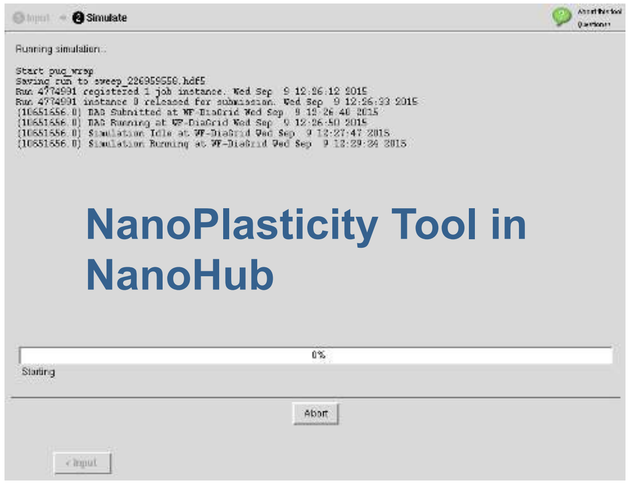
This tool is used to investigate the competing grain-boundary and dislocation-mediated deformation mechanisms in nanocrystalline metals. The model is based on a multi phase field model in which dislocation and grain boundary sliding are represented by means of scalar phase fields described in "The role of grain boundary energetics on the maximum strength of nanocrystalline Ni", Koslowski, Lee and Lei, *Journal of the Mechanics and Physics of Solids*, 59 1427-1436, 2011.

Burgers vector: Single Value

Burgers vector: 0.25nm

Note: The current version of this tool requires d1=d2=d3.

a) Specify and configure simulation in the tool UI

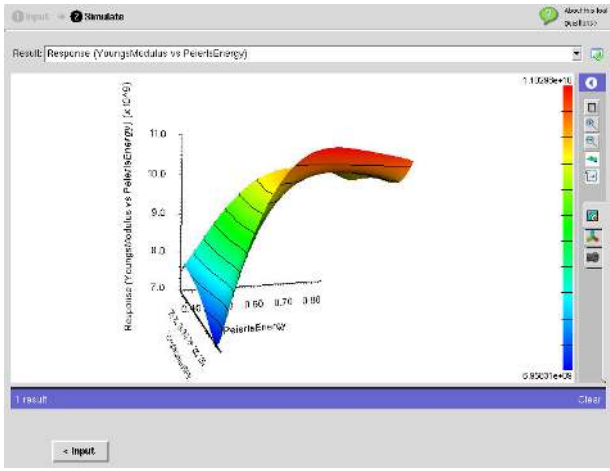


Running simulation...

```
Start puq_wrap
Saving run to sweep_226959558.hdf5
Run 4774991 registered 1 job instance. Wed Sep  9 12:26:12 2015
Run 4774991 instance 0 released for submission. Wed Sep  9 12:26:33 2015
[10651656.0] DAG Submitted at WF-DiaGrid Wed Sep  9 12:26:40 2015
[10651656.0] DAG Running at WF-DiaGrid Wed Sep  9 12:26:50 2015
[10651656.0] Simulation Idle at WF-DiaGrid Wed Sep  9 12:27:47 2015
[10651656.0] Simulation Running at WF-DiaGrid Wed Sep  9 12:29:24 2015
```

NanoPlasticity Tool in NanoHub

b) Workflow Executing through Pegasus



c) Inspect outputs in tool UI

- **Rappture Based Tools**
  - Tools built using the Rappture GUI Builder
  - Users configure the simulation, prepare inputs using Rappture tool UI
  - Tool execution results in workflows launched through Submit/Pegasus
  - Jobs execute on DiaGrid and Open Science Grid, and outputs staged back to Hub
  - Users visualize the outputs in the tool UI
  - Tools in NanoHub, DiaGrid, NEESHub

Acknowledgements: Steven Clark and Derrick Kearney, Purdue University

Relevant Links

- **Pegasus: http://pegasus.isi.edu**
  - Tutorial and documentation: http://pegasus.isi.edu/wms/docs/latest/
  - Support: pegasus-users@isi.edu pegasus-support@isi.edu
- **Rappture & Submit:**
  - https://hubzero.org/documentation/2.0.0/tooldevs/grid.rappture_submit
- **Submit Command**
  - https://hubzero.org/documentation/2.0.0/tooldevs/grid.submitcmd
- **Pegasus Workflows on HubZero**
  - https://hubzero.org/documentation/2.0.0/tooldevs/grid.pegasuswf
  - Tutorial https://hubzero.org/tools/pegtut