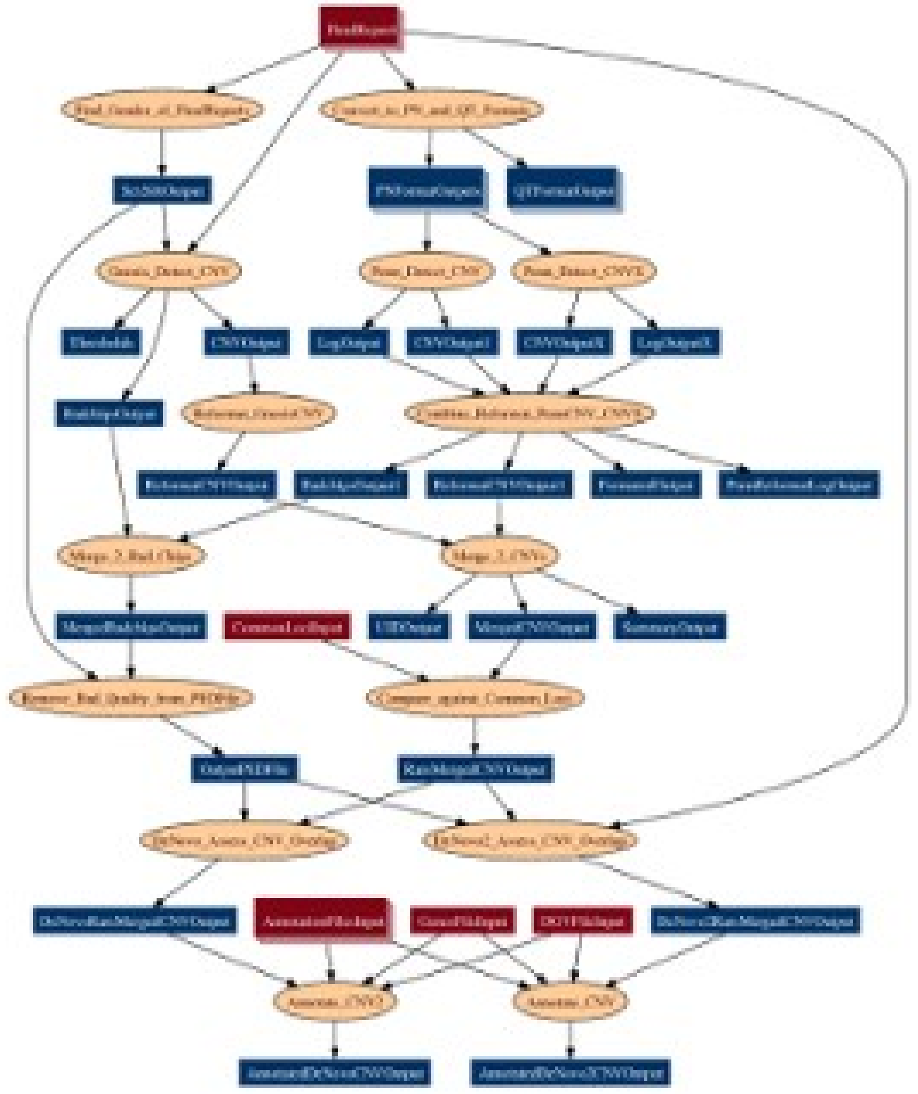
Pegasus as job submission framework for Virgo

About Pegasus

The Pegasus project encompasses a set of technologies that help workflow-based applications execute in a number of different environments including desktops, campus clusters, grids, and clouds. Pegasus bridges the scientific domain and the execution environment by automatically mapping high-level workflow descriptions onto distributed resources. It automatically locates the necessary input data and computational resources necessary for workflow execution.Pegasus enables scientists to construct workflows in abstract terms without worrying about the details of the underlying execution environment or the particulars of the low-level specifications required by the middleware (Condor, Globus, or Amazon EC2). Pegasus also bridges the current cyberinfrastructure by effectively coordinating multiple distributed resources.



About Pegasus

Portability / Reuse - User created workflows can easily be run in different environments without alteration. Pegasus currently runs workflows on top of Condor, Grid infrastrucutures such as Open Science Grid and TeraGrid, Amazon EC2, Nimbus, and many campus clusters. The same workflow can run on a single system or across a heterogeneous set of resources.

Performance - The Pegasus mapper can reorder, group, and prioritize tasks in order to increase the overall workflow performance.

Scalability - Pegasus can easily scale both the size of the workflow, and the resources that the workflow is distributed over. Pegasus runs workflows ranging from just a few computational tasks up to 1 million. The number of resources involved in executing a workflow can scale as needed without any impediments to performance.

Provenance - By default, all jobs in Pegasus are launched via the kickstart process that captures runtime provenance of the job and helps in debugging. The provenance data is collected in a database, and the data can be summaries with tools such as pegasus-statistics, pegasus-plots, or directly with SQL queries.

Data Management - Pegasus handles replica selection, data transfers and output registrations in data catalogs. These tasks are added to a workflow as auxilliary jobs by the Pegasus planner.

Reliability - Jobs and data transfers are automatically retried in case of failures. Debugging tools such as pegasus-analyzer helps the user to debug the workflow in case of non-recoverable failures.

Error Recovery - When errors occur, Pegasus tries to recover when possible by retrying tasks, by retrying the entire workflow, by providing workflow-level checkpointing, by re-mapping portions of the workflow, by trying alternative data sources for staging data, and, when all else fails, by providing a rescue workflow containing a description of only the work that remains to be done. It cleans up storage as the workflow is executed so that data-intensive workflows have enough space to execute on storage-constrained resource. Pegasus keeps track of what has been done (provenance) including the locations of data used and produced, and which software was used with which parameters.

About Pegasus

1) Define the input and output files

2) Define the executables

3) Define the jobs building up the workflow

4) Create an abstract workflow by setting up the dependencies among the jobs

5) Define the sites to which the abstract workflow will be mapped

6) Generate the execution modell

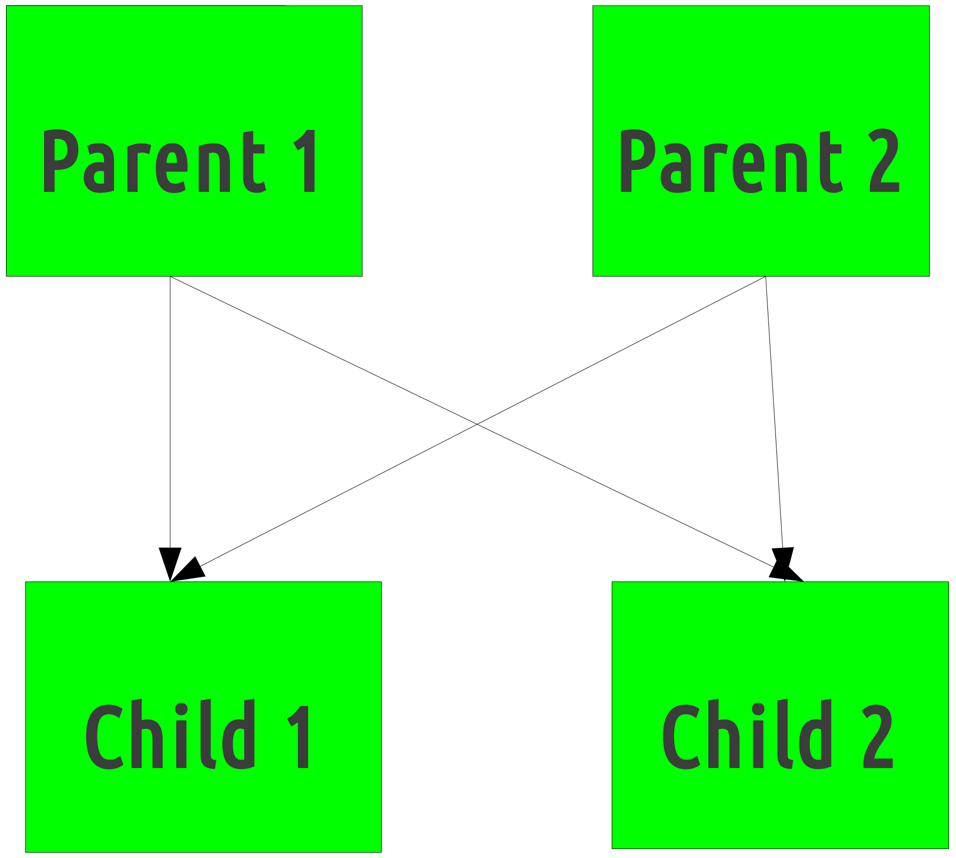
7) Execute

8) Monitor

9) Create the statistics

An abstract workflow can be mapped simultaneosuly to many different sites.

Example workflow



Example workflow:

Each of the child jobs needs the output of each of the parent jobs.

For example a hierarchical search

Example

```
#!/usr/bin/env python

# Import pegasus modules
from Pegasus.DAX3 import *
import os
import sys


#Create a DAX
dax =ADAG("pipeline")

# Add input file to the DAX-level replica catalog
parent_input = File("input.gwf")
parent_input.addPFN(PFN("file://" + "/tmp/pegasus-test/input.gwf", "local"))
dax.addFile(parent_input)

# Define files that willbe used / produced by the workflow
parent1out = File("parent1.out")
parent2out = File("parent2.out")
child1out  = File("child1.out")
child2out  = File("child2.out")

# Define the executables, their properties and locations
e_parent1 = Executable(namespace="pipeline", name="parent1", version="2.0", os="linux", arch="x86", installed=False)
e_parent1.addPFN(PFN("file://" + "/tmp/pegasus-test/parent1.sh", "local"))
dax.addExecutable(e_parent1)

e_parent2 = Executable(namespace="pipeline", name="parent2", version="2.0", os="linux", arch="x86", installed=False)
e_parent2.addPFN(PFN("file://" + "/tmp/pegasus-test/parent2.sh", "local"))
dax.addExecutable(e_parent2)

e_child1 = Executable(namespace="pipeline", name="child1", version="2.0", os="linux", arch="x86", installed=False)
e_child1.addPFN(PFN("file://" + "/tmp/pegasus-test/child1.sh", "local"))
dax.addExecutable(e_child1)
```

Example

```
e_child2 = Executable(namespace="pipeline", name="child2", version="2.0", os="linux", arch="x86", installed=False)
e_child2.addPFN(PFN("file://" + "/tmp/pegasus-test/child2.sh", "local"))
dax.addExecutable(e_child2)

# Define the jobs and assign the executable to the jobs
parent1 = Job(e_parent1)
parent1.addArguments("test ", "argument")
parent1.setStdout(parent1out)
parent1.uses(parent_input, link=Link.INPUT, transfer=True, register=False)
parent1.uses(parent1out, link=Link.OUTPUT, transfer=True, register=False)
dax.addJob(parent1)

parent2 = Job(e_parent2)
parent2.addArguments("test ", "argument")
parent2.setStdout(parent2out)
parent2.uses(parent_input, link=Link.INPUT, transfer=True, register=False)
parent2.uses(parent2out, link=Link.OUTPUT, transfer=True, register=False)
dax.addJob(parent2)

child1 = Job(e_child1)
child1.addArguments("test ", "argument")
child1.setStdout(child1out)
child1.uses(parent1out, link=Link.INPUT, transfer=True, register=False)
child1.uses(child1out,  link=Link.OUTPUT, transfer=True, register=False)
dax.addJob(child1)

child2 = Job(e_child2)
child2.addArguments("test ", "argument")
child2.setStdout(child2out)
child2.uses(parent2out, link=Link.INPUT,  transfer=True, register=False)
child2.uses(child2out,  link=Link.OUTPUT, transfer=True, register=False)
dax.addJob(child2)
```

Generating, mapping, launching

```
# Define the dependencies among the jobs
dax.depends(child1, parent1)
dax.depends(child1, parent2)
dax.depends(child2, parent1)
dax.depends(child2, parent2)

# Writing out the DAX
dax.writeXML(sys.stdout)
```

After this one can execute the script to generate a DAX:

```
./pipeline.py > ./pipeline.dax
```

The pegasus planner than maps the abstrax DAX workflow to a specific execution environment:

```
pegasus-plan --conf pegasusrc --sites local --output local --dir work --dax pipeline.dax
```

Then the pipeline can be run or submitted:

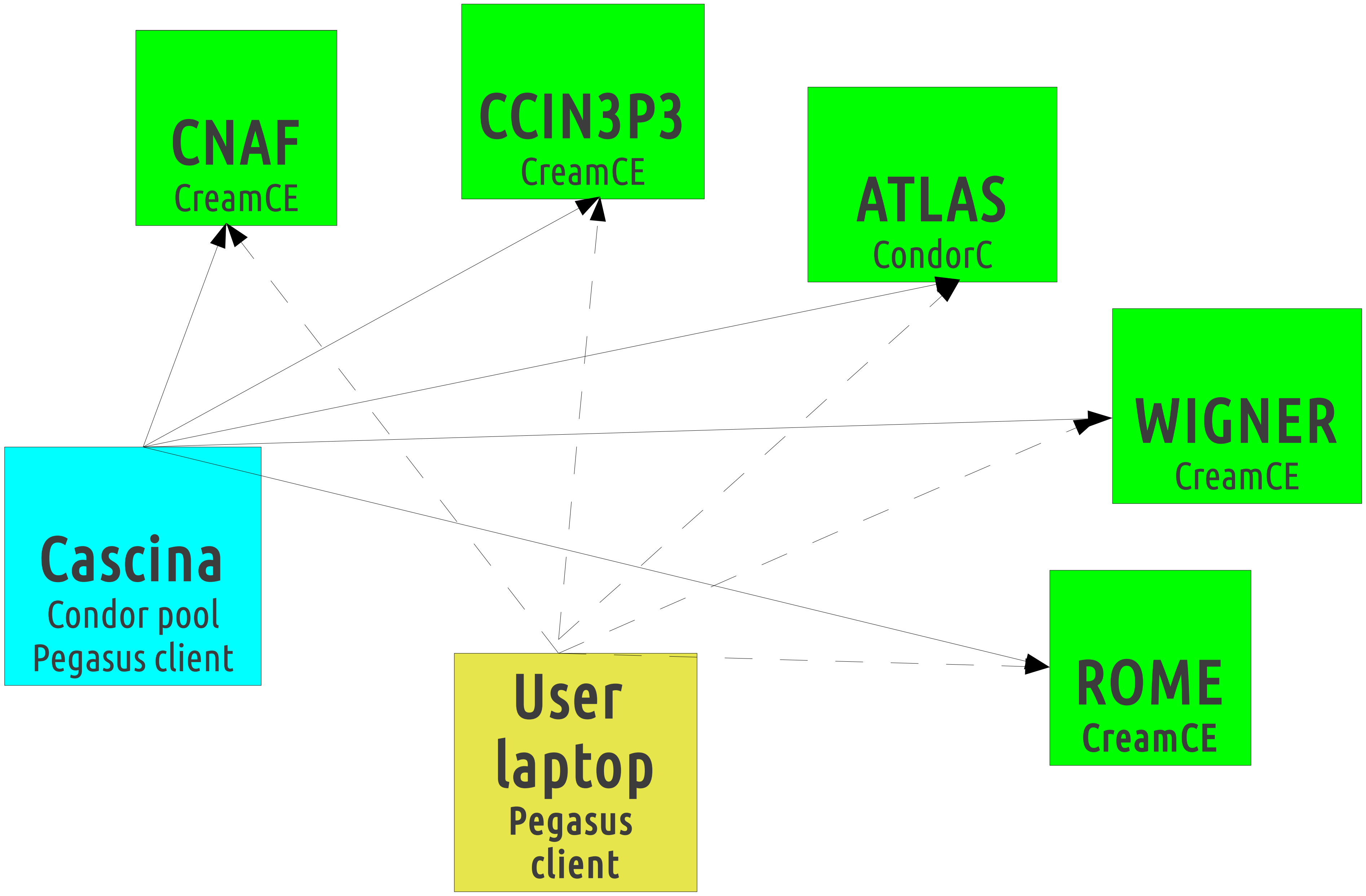
```
pegasus-run
/home/gdebrecz/Arbeits/Virgo/Pegasus/ex2/pipeline/work/gdebrecz/pegasus/pipeline/20140223
T230320+0100
```

Sites XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sitecatalog xmlns="http://pegasus.isi.edu/schema/sitecatalog"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://pegasus.isi.edu/schema/sitecatalog http://pegasus.isi.edu/schema/sc-3.0.xsd"
             version="3.0">
    <site  handle="local" arch="x86" os="LINUX">
        <grid  type="gt2" contact="localhost/jobmanager-fork" scheduler="Fork" jobtype="auxillary"/>
        <grid  type="gt2" contact="localhost/jobmanager-fork" scheduler="Fork" jobtype="compute"/>
        <head-fs>
            <scratch>
                <shared>
                    <file-server protocol="file" url="file://" mount-point="/tmp/pegasus-test"/>
                    <internal-mount-point mount-point="/tmp/pegasus-test" free-size="100G" total-size="30G"/>
                </shared>
            </scratch>
            <storage>
                <shared>
                    <file-server protocol="file" url="file://" mount-point="/tmp/pegasus-test"/>
                    <internal-mount-point mount-point="/tmp/pegasus-test" free-size="100G" total-size="30G"/>
                </shared>
            </storage>
        </head-fs>
        <replica-catalog  type="LRC" url="rlsn://dummyValue.url.edu" />
        <profile namespace="env" key="PEGASUS_HOME" >/usr</profile>
    </site>
   <site  handle="creamce" arch="x86" os="LINUX">
        <grid type="cream" contact="https://ce01-lcg.cr.cnaf.infn.it:8443/ce-cream/services/CREAM2" scheduler="LSF" jobtype="compute" />
        <grid type="cream" contact="https://ce01-lcg.cr.cnaf.infn.it:8443/ce-cream/services/CREAM2" scheduler="LSF" jobtype="auxillary" />

        <directory type="shared-scratch" path="/home/virgo034">
            <file-server operation="all" url="gsiftp://ce01-lcg.cr.cnaf.infn.it/home/virgo034"/>
        </directory>
                                                                                                $
        <profile namespace="pegasus" key="style">cream</profile>
        <profile namespace="globus" key="queue">virgo</profile>
    </site>
    <site  handle="condorpool" arch="x86" os="LINUX">
        <head-fs>
            <scratch />
            <storage />
        </head-fs>
        <profile namespace="pegasus" key="style" >condor</profile>
        <profile namespace="condor" key="universe" >vanilla</profile>
    </site>
</sitecatalog>
```

Possible submission scenario



Features, considerations

Some good stuff:

1) Pegasus can be interfaced with Globus, Condor, Condor-C, GRAM, CreamCE, XEDE, etc...
2) LIGO pipelines are mainly using Pegasus
3) If we want the computing situation to be more symmetric, i.e. to run LIGO pipelines on Virgo resources, the only way is to use Pegasus...because of manpower.
4) Programmable, generative modell, many API
5) Many I/O protocol is supported
6) Failure tolerant
7) Easy to port / run the pipeline on differetn execution environment
8) Data transfer decoupled, any file catalog can be used by the job itself

To clarify:

9) Long term EMI / CreamCE support
10)More helpful error messages