Pegasus Workflows at CHTC

Introduction

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. Some of the advantages of using Pegasus includes:

- **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 CHTC condor pool , 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 auxiliary 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.

Sample Workflow at CHTC

Introduction

The CHTC condor pool has no read/write enabled shared file system across the resources. Jobs are required to either bring inputs along with the job, or as part of the job stage the inputs from a remote location. The following examples highlight how Pegasus can be used to manage workloads in such an environment by providing an abstraction layer around things like data movements and job retries, enabling the users to run larger workloads, spending less time developing job management tool and babysitting their computations.

Pegasus workflows have 4 components:

- **DAX** - Abstract workflow description containing compute steps and dependencies between the steps. This is called abstract because it does not contain data locations and available software. The DAX format is XML, but it is most commonly generated via the provided APIS (documentation). Python, Java and Perl APIs are available.

- **Transformation Catalog** - Specifies locations of software used by the workflow

- **Replica Catalog** - Specifies locations of input data

- **Site Catalog** - Describes the execution environment

However, for simple workflows, the transformation and replica catalog can be contained inside the DAX, and to further simplify the setup, the following examples generate the site catalog on the fly. This means that the user really only has to be concerned about creating the DAX.

For details, please refer to the Pegasus documentation

The example described here for CHTC uses the python DAX API to generate the input workflow description.

Other canonical examples that illustrate how to construct common data processing patterns Pegasus workflows using the Python, Java and Perl DAX API's can be found here .

Wordfreq Workflow

wordfreq is an example application and workflow used to introduce Pegasus tools and concepts. The workflow is available on the CHTC login node submit-1.chtc.wisc.edu

Step 1: create a copy of the Pegasus tutorial and change the working directory to the wordfreq workflow by running the following commands:

```
$ cd $HOME
$ cp ~vahi/wordfreq-workflow.tgz .
$ tar zxvf wordfreq-workflow.tgz
$ cd wordfreq-workflow
$ export PATH=/home/vahi/SOFTWARE/pegasus/default/bin:$PATH
```

In the wordfreq-workflow directory, you will find:

- *inputs/* (directory)
- *dax-generator.py*
- *pegasusrc*
- *submit*
- *wordfreq*

The *inputs/* directory contains 6 public domain ebooks. The application in this example is *wordfreq,* which takes in a text file, does a word frequency analysis, and outputs a file with the frequency table. The task of the workflow is to run *wordfreq* on each book in the*inputs/* directory. A set of independent tasks like this is a common use case on OSG and this workflow can be thought of as template for such problems. For example, instead of wordfreq on ebooks, the application could be protein folding on a set of input structures.

When invoked, the DAX generator (*dax-generator.py*) loops over the *inputs/* directory and creates compute steps for each input. As the wordfreq application only has simple inputs/outputs, and no job dependencies, the dax generator is very simple:

dax-generator.py

```
#!/usr/bin/env python
from Pegasus.DAX3 import *
import sys
import os

base_dir = os.getcwd()

# Create a abstract dag
dax = ADAG("wordfreq-workflow")

# notifcations on state changes for the dax
dax.invoke("all", "/usr/share/pegasus/notification/email")

# Add executables to the DAX-level replica catalog
wordfreq = Executable(name="wordfreq", arch="x86_64", installed=False)
wordfreq.addPFN(PFN("file://" + base_dir + "/wordfreq", "local"))
dax.addExecutable(wordfreq)

# add jobs, one for each input file
inputs_dir = base_dir + "/inputs"
for in_name in os.listdir(inputs_dir):

    # Add input file to the DAX-level replica catalog
    in_file = File(in_name)
    in_file.addPFN(PFN("file://" + inputs_dir + "/" + in_name, "local"))
    dax.addFile(in_file)

    # Add job
    wordfreq_job = Job(name="wordfreq")
    out_file = File(in_name + ".out")
    wordfreq_job.addArguments(in_file, out_file)
    wordfreq_job.uses(in_file, link=Link.INPUT)
    wordfreq_job.uses(out_file, link=Link.OUTPUT)
    dax.addJob(wordfreq_job)

# Write the DAX to stdout
f = open("dax.xml", "w")
dax.writeXML(f)
f.close()
```

Note how the DAX is devoid of data movement and job details. These are added by Pegasus when the DAX is planned to an executable workflow, and provides the higher level abstraction mentioned earlier.

In the tarball there is also a *submit* script. This is a convenience script written in bash, and it performs three steps: runs the dax generator, generates a site catalog, and plans/submits the workflow for execution. The site catalog does not really have to be created every time we plan/submit a workflow, but in this case we have a workflow which is used by different users, so changing the paths to scratch and output filesystems on the fly makes the workflow easier to share.

submit

```
#!/bin/bash

set -e

TOPDIR=`pwd`

# generate the dax
export PYTHONPATH=`pegasus-config --python`
./dax-generator.py

# create the site catalog
cat >sites.xml <<EOF

<?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-4.0.
xsd" version="4.0">

       <site  handle="local" arch="x86_64" os="LINUX" osrelease="" osversion="" glibc="">
                  <directory  path="/home/$HOME/data/outputs" type="shared-storage" free-size="" total-size="">
                         <file-server  operation="all" url="file:///home/$HOME/data/outputs"/>
                    </directory>

                    <directory  path="/home/$HOME/data/scratch" type="shared-scratch" free-size="" total-size=""
>
                         <file-server  operation="all" url="file:///home/$HOME/data/scratch"/>
                    </directory>
       </site>
       <site  handle="chtc" arch="x86_64" os="LINUX" osrelease="" osversion="" glibc="">
            <profile namespace="condor" key="+ProjectName" >"con-train"</profile>
                <profile namespace="condor" key="universe" >vanilla</profile>
                 <profile namespace="pegasus" key="style" >condor</profile>
       </site>
</sitecatalog>
EOF




# plan and submit the  workflow
pegasus-plan \
    --conf pegasusrc \
    --sites chtc \
    --dir /home/$USER/data/workflows \
    --output-site local \
    --dax dax.xml \
    --cluster horizontal \
    --submit
```

Step 2: Submit the workflow by executing the submit command.

Note that when Pegasus plans/submits a workflow, a work directory is created and presented in the output. This directory is the handle to the workflow instance and used by Pegasus command line tools.

Sample submit output

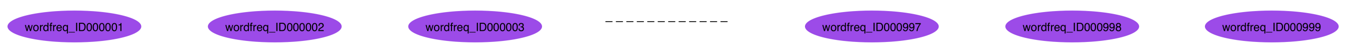
```
[vahi@submit-1 wordfreq-workflow]$ ./submit
2014.05.22 16:11:49.552 CDT:   Submitting job(s).
2014.05.22 16:11:49.560 CDT:   1 job(s) submitted to cluster 45647956.
2014.05.22 16:11:49.567 CDT:
2014.05.22 16:11:49.573 CDT:   -----------------------------------------------------------------------
2014.05.22 16:11:49.580 CDT:   File for submitting this DAG to Condor           : wordfreq-workflow-0.dag.
condor.sub
2014.05.22 16:11:49.587 CDT:   Log of DAGMan debugging messages                 : wordfreq-workflow-0.dag.
dagman.out
2014.05.22 16:11:49.594 CDT:   Log of Condor library output                     : wordfreq-workflow-0.dag.lib.
out
2014.05.22 16:11:49.601 CDT:   Log of Condor library error messages             : wordfreq-workflow-0.dag.lib.
err
2014.05.22 16:11:49.608 CDT:   Log of the life of condor_dagman itself          : wordfreq-workflow-0.dag.
dagman.log
2014.05.22 16:11:49.615 CDT:
2014.05.22 16:11:49.622 CDT:   -----------------------------------------------------------------------
2014.05.22 16:11:49.629 CDT:
2014.05.22 16:11:49.636 CDT:   Your workflow has been started and is running in the base directory:
2014.05.22 16:11:49.643 CDT:
2014.05.22 16:11:49.650 CDT:     /home/vahi/wordfreq-workflow/dags/vahi/pegasus/wordfreq-workflow/run0001
2014.05.22 16:20:27.515 CDT:
2014.05.22 16:11:49.657 CDT:
2014.05.22 16:11:49.664 CDT:   *** To monitor the workflow you can run ***
2014.05.22 16:11:49.671 CDT:
2014.05.22 16:11:49.678 CDT:     pegasus-status -l /home/vahi/wordfreq-workflow/dags/vahi/pegasus/wordfreq-
workflow/run0001
2014.05.22 16:20:27.515 CDT:
2014.05.22 16:11:49.685 CDT:
2014.05.22 16:11:49.692 CDT:   *** To remove your workflow run ***
2014.05.22 16:11:49.699 CDT:
2014.05.22 16:11:49.706 CDT:     pegasus-remove /home/vahi/wordfreq-workflow/dags/vahi/pegasus/wordfreq-workflow
/run0001
2014.05.22 16:20:27.515 CDT:
2014.05.22 16:11:49.713 CDT:
2014.05.22 16:11:50.227 CDT:   Time taken to execute is 10.814 seconds
```

Note: The wfdir is the one that you see on your terminal. The one above is for example purposes only.

Some useful tools to know about:

- ***pegasus-status -v [wfdir]***
  Provides status on a currently running workflow. (more)
- ***pegasus-analyzer [wfdir]***
  Provides debugging clues why a workflow failed. Run this after a workflow has failed. (more)
- ***pegasus-statistics [wfdir]***
  Provides statistics, such as walltimes, on a workflow after it has completed. (more)
- ***pegasus-remove [wfdir]***
  Removes a workflow from the system. (more)

During the workflow planning, Pegasus transforms the workflow to make it work well in the target execution environment. Our DAX had1000 independent tasks defined:



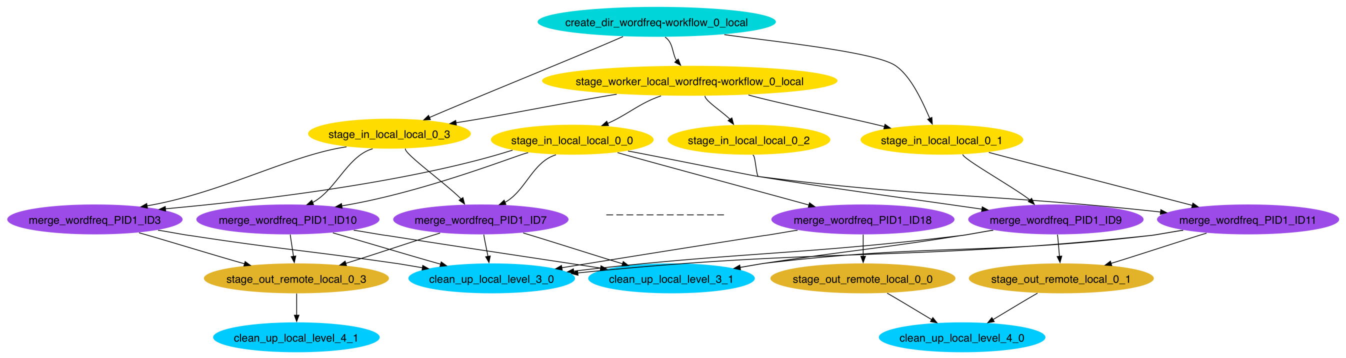
As these tasks are really short, let's tell Pegasus to cluster multiple tasks together into jobs. If you do not do this step, the jobs will still run, but not very efficiently. This is because every job has a small scheduling overhead. For short jobs, the overhead is obvious. If we make the jobs longer, the scheduling overhead becomes negligible. Hence for this example, we have the task clustering feature of Pegasus enabled

In the dax-generator.py script we have

```
wordfreq.addProfile(Profile(Namespace.PEGASUS, "clusters.size", 50))
```

This informs Pegasus that it is ok to cluster up to 50 of the wordfreq tasks in each job.

The executable workflow has also has a set of additional tasks added by Pegasus: create scratch dir, data staging in and out, and data cleanup jobs



Step 3: Check the status of the workflow:

```
$ pegasus-status [wfdir]
```

You can keep checking the status periodically to see that the workflow is making progress.

Step 4: Examine a submit file and the *.dag.dagman.out files. Do these look familiar to you from DAGMan descriptions? Pegasus is based on HTCondor and DAGMan.

```
$ cd [wfdir]
$ cat wordfreq_ID0000001.sub
...
$ cat *.dag.dagman.out
...
```

Step 5: Keep checking progress with pegasus-status. Once the workflow is done, display statistics with pegasus-statistics:

```
$ pegasus-status [wfdir]
$ pegasus-statistics [wfdir]
...
```

Step 6: cd to the output directory and look at the outputs. Which is the most common word used in the 6 books? Hint:

```
$ cd ~/data/outputs/
$ head -n 3 *.out
```

Step 8: Cleanup. If you want to remove the jobs for a particular workflow you can use the pegasus-remove command

```
$ pegasus-remove [wfdir]
```