NOAO SDM Document PL019

Transforming NHPPS Pipeline Applications into Grid Applications

F. Valdes[1]

National Optical Astronomy Observatories
Science Data Management

Draft: March 26, 2010
V1.0: June 29, 2010

[1]NOAO Data Products Program, P.O. Box 26732, Tucson, AZ 85732

Table of Contents

Abstract

This document describes a powerful way to use *NOAO High Performance Pipeline System (NHPPS) pipeline applications (PA)* as *grid applications (GA)*. For the ODI Pipeline Project this consists of wrapping complete NHPPS pipelines as OGCE Grid Services using a single wrapper command supporting standardized input and output parameters. This requires dynamically running the NHPPS execution framework internally to the wrapper command so that it becomes a totally self-contained executable. The advantages of this architecture are identified.

Keywords: NHPPS, OGCE, pipelines

Purpose of this Document

The purpose of this document is to foster discussion about how to transform NOAO/NHPPS work into an ODI/OGCE workflow. This document advocates the architectural vision of the NOAO ODI pipeline development team to the ODI Pipeline/Archive Project.

1 Introduction

This document describes a powerful way to use *NOAO High Performance Pipeline System (NHPPS) pipeline applications (PA)* as *grid applications (GA)*. An NHPPS PA is a hierarchy of pipelines which perform a complex workflow using coarse-grained processes and data parallelism. A *pipeline* in this context is a smaller workflow, or conceptually a service, consisting of a number of *stages* which are executed in a possibly parallel order on a single *node*. A pipeline service operates on a particular type of dataset and there can be multiple instances, each operating on independent datasets, running on the same node or on a distributed set of nodes. Typically these instances are running in parallel. The stages in a pipeline are lower level components, called a *modules* which are typically scripts that perform a step of an algorithm and whose input/output context varies depending on how they interact with other stages. Modules are typically in a scripting language (e.g. python or IRAF) that provide control flow and access to compiled executable elements, called *tasks*.

The NHPPS execution framework is defined primarily by *Node Manager (NM)* and *Pipeline Manager (PM)* processes. The framework orchestrates the work flow based on events rather than as a pre-defined *directed acyclic graph (DAG)*. The events and how they trigger steps in the workflow are described by an NHPPS *pipeline description language (PDL)*.

In the context in which the NHPPS execution framework is currently used there is one NM for a node which controls a PM for each pipeline which may run on the node. The PM is the one responsible for scheduling all the stages of a pipeline for as many datasets as desired. These processes communicate with sibling process on other nodes which allows for pipelines to *call* other pipelines as services on the same node or other nodes for different datasets. In this configuration the PA forms a network of pipelines across different nodes with no head node and the workflow implicitly defined by *events* and a *pipeline selection* function that provides a list of available pipelines across the network to be called. The proviso in this architecture is the NM and PM processes are first started across a known set of nodes which can communicate with each other.

In a GA, the architecture is typically a, potentially, very large pool of nodes which execute pieces of a workflow independently and as leaf nodes returning results to a central orchestration machine. The orchestration is defined by a DAG description in some language. In particular, we wish to target an OGCE workflow application using the workflow constructs of that system. The grid nodes do not need to communicate and pre-existing daemon servers are not used.

The question addressed in this document is how can an NHPPS PA be turned into a typical GA. One answer might be to take the atomic tasks or stages in the PA out of the control of the NHPPS framework as described by the PDL and orchestrate them as a GA described by a DAG. However, the approach described here is to include the NHPPS execution framework in GA components, called *grid services*, so that it is the pipelines that are wrapped and executed on grid nodes. In brief, mapping each pipeline in an NHPPS PA to a single grid service command thereby maintaining the same overall workflow design. This has a number of advantages described below but two of which are 1) making use of the best features of both frameworks and 2) making the wrapping simpler by not having to translate the PDL into a complex DAG.

Before talking about the advantages we make this approach clearer by describing how this

wrapping is done.

2 Wrapping an NHPPS Pipeline as a Grid Service

The key transformation needed is to change the NHPPS framework from *node-centric*, where long-lived and intercommunicating NMs and PMs process datasets as they are available, to *dataset-centric*, where each dataset spawns its own isolated stand-alone NM and PM(s) to process it and quit when the work is completed.

An important point is that there can be more than one pipeline (i.e. PM) within an NHPPS wrapper such that a "higher level" pipeline makes use of one or more subpipelines in a possibly parallel fashion. The only difference from the non-GA configuration is that all subpipelines must execute only the same machine. This allows decomposition of a dataset into parallel subpipelines which can efficiently utilize multiple-cores even though individual stage modules can not. Note that this is only done when desired, the GA DAG can still be used when the dataset pieces are large and benefit from use of a larger pool of grid nodes. It is up to the PA/GA designer to define the workflow appropriately.

Most of the flexibility to transform the NHPPS framework in this way was already built-in to the system. The primary tweak needed was to allow multiple NMs to coexist on the same node as the same user (the NHPPS framework was already independent by user). Each independent NM then manages its own set of PM. The way this tweak is accomplished is to use different NM ports within the wrapper script.

3 Pipelines vs. Module

A question that comes up is why not use modules directly as grid services? This is essentially a definition and implementation question. No matter what level of NHPPS software is used one needs to provide a wrapper to transform the software to match the requirements of the grid service execution framework.

For NOAO developers a module has generally been considered an internal component of a pipeline. As such the environment in which it executes and the input and output data are dependent on what come before and after. Therefore individual wrappers would need to be created for each module. An advantage of wrapping NHPPS pipelines is that a single standard wrapper can be used.

A point to recognize is that pipelines, in the NHPPS sense, can be written with as many or as few modules as desired not counting some standard housekeeping functions that essentially are what provide the standardization of the input and output for a pipeline. So if a single significant science module is desired it could be wrapped as a short NHPPS pipeline and then as a grid service.

The considerations are then how fine the pipeline application workflow should be decomposed (i.e. how many pieces) and whether including the NHPPS execution framework in the grid service is a problem.

4 The Advantages

Below is a partial list of the advantages to making grid applications be complete NHPPS pipelines. However, I would first like to emphasize the main advantages from my perspective.

NHPPS pipelines are functionally designed as user level components. In other words, to do a particular processing operation that an astronomer would easily understand. As part of this the inputs and outputs are simple and standardized. This consists of a list of data to be used with a list of final data products along with various log information as output. Because of this standardization, wrapping of a pipeline for a grid application is very simple and can be done with a single wrapper command.

In contrast, the stages in a pipeline are sometimes just for bookkeeping, setup, or perform only part of a algorithm. Different stages have different functions and, hence, input and output. One could write larger stages (with loss of parallelization flexibility) but ultimately this would result in producing a single module that is functionally the same thing as the pipeline.

- The stage modules are generally too fine-grained for generic and efficient reuse while the pipelines are good candidates.
- Understanding and documenting the workflow logic at the pipeline level is easier and more appropriate for most people. A recently added capability to embedded well-formatted documentation in the PDL encourages good documentation and makes it easy to maintain and extract.
- The number of things to wrap in a complete workflow application is on the order of 20 pipelines rather than of order 200 stages. It should be obvious that decomposing down to the level of tasks (such as each IRAF command) is out of the question.
- The ability to parallelize within the GA service component, by processing more than one dataset at a time in an NHPPS wrapper service, will improve utilization of multiple cores.
- Handling input and output in OGCE at the pipeline level is more appropriate and standardized while the internals of communications between stages is varied and more complex.
- Much of the control flow that would be harder in a GA DAG, i.e. alternate paths through the workflow depending on the data, are taken care of by the NHPPS event flow and PDL.
- The NHPPS developers can more easily create workflows for both an NHPPS dedicated cluster and an OGCE grid application.

5 The Disadvantages

The main aspect of using NHPPS pipelines as grid services is that the NHPPS execution framework is included in the service. One might worry that this is inefficient. The NHPPS framework is fairly

light-weight so what is left is a trade-off between the overhead of the framework and the work done within the service. An advantage noted above is that an NHPPS pipeline can contribute to better efficiency by doing more work and making coarse-grained use of multiple cores with multiple asyncronous modules and independent datasets.

So the case where there would be a disadvantage is when a service does very little work. Many modules in NHPPS pipelines are in this category which is why it would not be a good idea to use them as separate grid services. There are a few modules that do significant work but then the framework is of lesser consequence and it would become a choice between the customized work of wrapping the module directly to eliminate the small overhead of the NHPPS framework and using the standardized pipeline wrapper for a pipeline of essentially one module.

The other potential concern is that one has to consider the additional failure modes that might occur from the framework. The framework is extremely reliable at this point but a pipeline has the potential failure of stopping after a module failure that would need to be caught.

6 Conclusion

The overhead of including the NHPPS execution framework as part of a grid service is less significant than its ability to efficiently utilize the resources of a grid node and to standardize the wrapper. It also has the advantages of reducing the number of grid services to a number that is suitable for potential reuse and understood by users as opposed to modules which often are more fine-grained because they form part of an algorithm.

Concerns about the loss of flexibility to extract a particular functionality for use in other workflows is ultimately part of the design of the pipeline services rather than blindly attempting to provide every module in a pipeline application as a grid service. If a particular science module is large enough and significant enough to be a separate grid service it can be individually wrapped either outside of an NHPPS pipeline or as an NHPPS pipeline with just a few modules.

7 Status

All the tweaks needed to run stand-alone and independent NHPPS pipelines on nodes with nothing but access to the code stack and standard Linux have been completed and demonstrated. A single OGCE-aware command (one that understands how to use OGCE input and output parameters) has been written to wrap any NHPPS pipeline. The command looks like:

nhppsapp config_file pipeline_name other_pipelines input_list

and the standard outputs, in OGCE format, include the output data list, process logs, metadata files, etc.