Asynchronous Data Stage-in to Improve the Performance of Scientific Workflows

Erin Griffiths, Samuel Hopkins, Ann Chervenak, Ewa Deelman
University of Southern California Information Sciences Institute
Marina Del Rey, CA USA
{ering,shopkins,annc,deelman}@isi.edu

***Abstract*—The stage-in of input data to a scientific workflow runs on a remote cluster accounts for a significant amount of the workflow's runtime. In this paper we present an implementation of a Data Placement Service, integrated with the Pegasus Workflow Management System, where Pegasus sends stage-in requests to the DPS for transfer. The goal of this implementation is to improve workflow runtime by staging in data asynchronously.**

I. Introduction

Many areas of science have complex analysis or simulations that have benefitted from workflow technologies. These science applications may contain hundreds of thousands of interdependent tasks that require a large amount of data in order to begin computations. The management of this data plays a significant role in the running of a workflow. Previous studies have shown that improving the management of data can significantly improve the efficiency of a workflow [1]. Techniques like data prestaging, where the input data is transferred before the workflow even begins execution, and accessing data via symbolic links that is on a storage system available to a computation cluster have been shown to improve workflow runtime.

One study in particular looked at the stage-out and cleanup of data produced by an executed workflow. By assigning the tasks of stage-out and cleanup to an asynchronous Data Placement Service Amer, *et al.* [2] showed that workflow runtime can be improved significantly since the management system is no longer responsible for these tasks.

This paper examines the effect of assigning stage-in jobs to an asynchronous Data Placement Service (DPS) and if this will improve the runtime of a workflow. We integrated a DPS with the Pegasus Workflow Management System [3], where Pegasus will rely on the DPS to perform stage-in data transfers. We explored the benefits of this change using the astronomy application Montage [4].

This paper begins with an overview of the Pegasus Workflow Management System and its default behavior handling the stage-in of data. We then describe the different modifications made to Pegasus in order to use a DPS to perform stage-in, while still keeping the data dependencies need to run each job of the workflow. Afterwards, we describe the steps preformed by the DPS to stage-in the data. Next, in section III, we describe the workflow used to examine the effect of this change in the workflow's runtime. In section IV we describe the performance evaluation, what variables were measured, and what results we found. We conclude with future work in this area.

II. Pegasus and the Data Placement Service

Pegasus is a workflow management system designed to map and execute complex workflows onto distributed systems. It allows users to specify an abstract of the workflow they wish to run without worrying about the exact resources it will be run on, or if the runtime of their workflow will be optimal.

Pegasus is run on *submit host*, where the workflow application and other configuration details are specified. First the workflow is planned using Pegasus, then Pegasus submits the executable workflow it just planned to a remote resource. This remote resource, the *execution site,* is where the actual computations are preformed.

When a workflow is submitted to an execution site each job must wait in two successive queues before being executed. The first is the local queue on the submit host. Pegasus only releases a job into this queue if its dependant jobs have completed successfully. If job two depends on the output from job one, job two will only be released into the local queue once job one has competed successfully on the execution site. Once a job has passed through the local queue, it must then also pass through the remote queue before being executed.

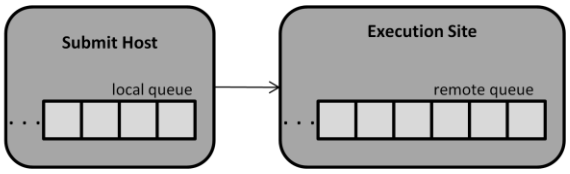

Figure 1: Workflow Submission and Execution

Modifications to Pegasus:

Pegasus' default behavior for stage-in jobs is to call a data transfer service, like GridFTP, internally and block until all stage-in transfers are complete. This blocking preserves the data dependencies in the workflow so that each subsequent job has the data it needs available. Once the Pegasus stage-in is complete, depended compute jobs are released into the local queue.

Our modified Pegasus does not call a file transfer service. Instead, a request is formatted and sent to the DPS asking for it to transfer the file. This request is formatted as a list of source/destination physical filename pairs. From the view of Pegasus this request is non-blocking as the DPS releases Pegasus before beginning to transfer files and the transfer time does not impede the progression of the workflow.

Because we are examining stage-in, subsequent jobs in the workflow are dependent on the success of the transfers preformed by the DPS. The workflow cannot proceed without these requested files, meaning some communication is required between the DPS and Pegasus. This communication was achieved by implementing a replica location service (RLS) that holds information on the transferred files [5]. After sending the request, the modified Pegasus stage-in job then polls this RLS for the DPS status of each requested transfer. If any of the transfers are marked as failed, then that stage-in job fails. If all requested transfers are marked as successful, Pegasus can continue the execution of the workflow by releasing the subsequent jobs into the local queue.

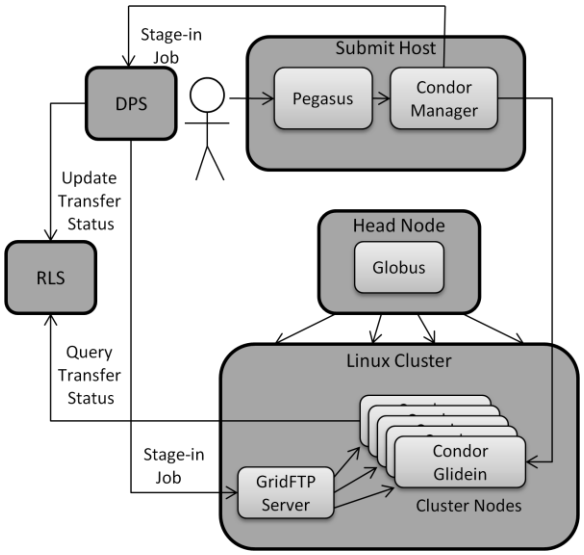

Figure 1: Overall Control Flow

We implemented this RLS polling in two ways. The first was to have it simply as part of the stage-in job, where directly after the request was sent, polling began to see if the transfers completed successfully. We did not expect this implementation to have a beneficial effect on the runtime of the workflow. In comparison to Pegasus' default behavior this implementation added the extra overhead of communicating with both the RLS and DPS to the transfers that Pegasus had already been making in its default behavior.

The second implementation moves this polling into the beginning of the workflow's compute jobs. Before each compute job Pegasus polls for the files specific to that compute job. With this implementation, in order for intermediate data, which is not sent to the DPS for transfer, to not cause the job to

fail the polling of the RLS must be modified. The entries in the RLS that do not have a DPS status attribute must be assumed to have been transferred by another service. Therefore, only the files that have a DPS status in the RLS are polled for success or failure.

We expect this implementation to improve the runtime of the workflow because transfers are now made asynchronously to the executing workflow. One the stage-in job reaches the execution site it sends the DPS request and finished successfully. Then the subsequent compute jobs are released into the local queue. While these compute jobs travel though the local and remote queue, the DPS is transferring the files they will require. This means the transfer and queue time can run in parallel for a time, rather than one after the other.

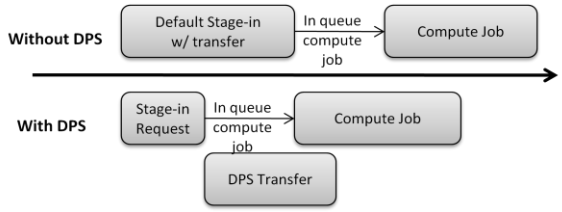

Figure 2: Time gain using polling within the compute job.

The drawback to this second implementation is that if compute job reaches the execution site long before the file is finished transferring, the requisite polling will waste CPU time.

Data Placement Service:

The DPS that we modified was modeled from a DPS previously designed to handle stage-out and clean up jobs for the Pegasus Workflow Management System. Like that model, our DPS manages a central thread pool for stage-in requests and uses GridFTP to perform the transfers. The requests are received over a java RMI. GSI security is not currently used in our design.

When the DPS receives a stage-in request it first updates the RLS so that each requested transfer has a DPS Status of "in progress". Then the DPS releases the request and checks to see if the destination directories exist. If not, the DPS recursively creates this directory. If the source file cannot be found, the DPS makes five attempts at the transfer before giving up. Each file is sent using GridFTP. If the transfer completes successfully, then the DPS status in the RLS is updated to success. If any error occurs the entry is updated to error.

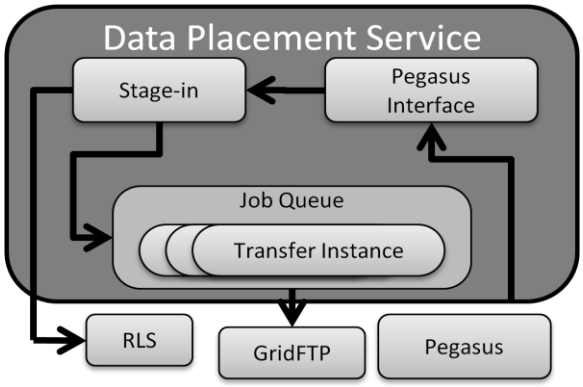

Figure 3: Data Placement Service

III. Evaluated Workflows

Montage:

Montage [4] is an astronomy application used to create science grade images of the night sky. It is an I/O intensive application that takes many smaller input images, re-projects, overlaps, and corrects them though a series of steps to make a comprehensive final image. We used Montage as our test application because it is both widely used in the scientific community and requires a large number of input files to be staged in. Workflows of higher degrees require more input data and compute jobs, and produce a larger image of the night sky.

IV. Performance Evaluation

Experimental Setup:

The DPS ran on a 2 core, 2.66 GHz, Linux machine with 2 GB of RAM, Pegasus also ran on a 2 core, 2.66 GHz, Linux machine with 2 GB of RAM (the submit host), and the workflow

was executed on a twenty node Linux based cluster, with each node running Linux 2.6.23 kernel on a 4 core Intel Xeon 2.33 GHz CPU and 8 GB RAM (the execution site).

In our experiments we used a Corral Server to provision resources on the execution site before running the workflows in order to provide a stable testing environment. All experiments were run using this configuration.

Measurements Taken:

To examine the performance change of the workflow we looked first at the overall runtime. We measured the runtime of the workflow using the default stage-in preformed by Pegasus, as well as both implementations of the DPS stage-in with polling in Pegasus. For all three configurations we planned to run the workflow five times and computed the average runtime and standard deviation.

Results:

The experiments were run on Montage workflows of degree 0.5 and 8. When running the 0.5 degree no other runtime improvement techniques, such as clustering, were used. The 8 degree Montage was performed with clustering. This allowed multiple compute jobs to be clustered into one job. Here polling acts upon this clustered job as if it was only one job. The job polls for each file needed by every compute job in its cluster before running any of the contained jobs.

Preliminary results from the 0.5 degree Montage showed no significant change in the runtime for either of our implementations.

Preliminary results from the 8 degree Montage showed a slight, but noticeable decrease in the runtime for workflows when using implementation two, polling before the compute job.

V. Conclusions and Future Work

Unfortunately, due to time constraints we were unable to run a full battery of test. Preliminary testing was all we were able to achieve. We documented our changes thoroughly so that the work can be picked up at a later date and continued. This work includes fully testing our implementations and looking at other implementations for polling in Pegasus. Further runtime improvement might be achieved if the location of the polling and the amount of clustering can be tuned and optimized for a workflow.

Future work in this area includes looking further at ways to implement asynchronous stage-in communication between a DPS and a workflow management system. Further testing on alternate workflows, multiple workflows, and grids that are in higher demand (like the Open Science Grid) would also help improve our understanding of asynchronous data stage-in. Future work for a DPS also includes adding VO policy considerations to its data management, including data storage availability and policies.

VI. References

[1] A. L. Chervenak, *et al.,* "Data placement for scientific applications in distributive environments," presented at the Grid Computing, 2007 8th IEEE/ACM International Conference on, 2007.

[2] M. Amer, *et al.*, "Separating Workflow Management and Data Staging to Improve the Performance of Scientific Workflows," in submission.

[3] E. Deelman, *et al.,* "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program*., vol. 13, pp. 219-237 2005.

[4] B. Berriman, *et al.*, "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand," in *SPIE Conference 5487*: 2004.

[5] A. L. Chervenak, *et al.,* "The Globus Replica Location Service: Design and Experience," *IEEE Transactions on Parallel and Distributed Systems.* Vol. 20 (9) (September 2009), pp. 1260-1272