Integrating Policy with Scientific Workflow Management for Data-Intensive Applications

Ann L. Chervenak, David E. Smith, Weiwei Chen, Ewa Deelman
Information Sciences Institute
University of Southern California
Marina del Rey, CA, USA
{annc, smithd, wchen, deelman}@isi.edu

***Abstract*—As scientific applications generate and consume data at ever-increasing rates, scientific workflow systems that manage the growing complexity of analyses and data movement will increase in importance. The goal of our work is to improve the overall performance of scientific workflows by using policy to improve data staging into and out of computational resources. We developed a Policy Service that gives advice to the workflow system about how to stage data, including advice on the order of data transfers and on transfer parameters. The Policy Service gives this advice based on its knowledge of ongoing transfers, recent transfer performance, and the current allocation of resources for data staging. The paper describes the architecture of the Policy Service and its integration with the Pegasus Workflow Management System. It employs a range of policies for data staging, and presents performance results for one policy that does a greedy allocation of data transfer streams between source and destination sites. The results show performance improvements for a data-intensive workflow: the Montage astronomy workflow augmented to perform additional large data staging operations.**

***Index Terms*—data placement, scientific workflow, policy service, greedy allocation policy.**

I. Introduction

Scientific applications in a number of domains are generating and consuming data at increasing rates. As a result, the analyses needed to process these data are becoming ever more complex, and they are often built out of individual application components. These components often must execute in a specific order, based on data or control dependencies among the components. The trend of ever increasing data sizes and of growing analysis complexity is expected to continue and to pose ever greater scalability challenges on the application and on resource management systems [1].

Scientific workflows are a way of managing these complex scientific analyses [2, 3]. Workflows allow scientists to declaratively describe their application's components and the application inputs and outputs. Often, the data in the workflows are represented in files. Due to the large data sizes and the complexity of the computations, these workflows need to be executed on remote, potentially distributed resources such as campus clusters, grids, and clouds. The data sets may be distributed over the wide area network as well. Thus, workflow management systems need to be able to orchestrate the execution of the computations on the available resources, and they also need to manage the staging of data in and out of computational resources. Since storage, especially at computational sites, is finite, the workflow management system also needs to remove data that are no longer needed for upcoming computations.

The goal of our work is to improve the overall performance of scientific workflows by using policy to improve the performance of data staging into and out of computational resources. Our approach is to off-load data staging policy decisions from the workflow management system to a Policy Service, which is aware of ongoing transfers, recent data transfer performance, and the current allocation of storage and network resources for data staging. The policy engine then balances the data movement within a workflow and across multiple concurrently executing workflows.

The contributions of this paper are:

- An architecture and implementation of a general policy service that can be tailored to specific purposes, such as workflow management in this paper or to adaptive data transfer [4].
- The integration of the policy service with a workflow system: the Pegasus Workflow Management System [5].
- A set of data staging policies that can be applied within the service and the implementation of two of them in policy rules.
- Initial performance results for one data staging policy based on greedy allocation of streams to data transfers between source and destination sites.

Our evaluation shows that by managing stream allocations, we can significantly improve the performance of an astronomy workflow (Montage) that is augmented to perform additional large data staging operations, a scenario that is reflective of emerging *big data* applications [6-8].

II. Policy Engine

Fig. 1 shows the architecture of our Policy Service, which provides advice to Pegasus about data staging and cleanup operations performed as part of the workflow execution while taking into account the entire data staging environment. The

Policy Service removes duplicate staging and cleanup requests, allows multiple workflows to share staged files safely, defines the default number of parallel streams to use for each transfer, and enforces a maximum number of parallel streams to be allocated between a source and destination host.

A. Architecture Overview

As shown in Fig. 1 our Policy Service architecture consists of the following components. An *Apache Tomcat Container* manages the runtime environment of the Policy Service components. A *RESTful Web Interface* allows access to the policy service over the web using XML or JSON data structures. A *Policy Controller* manages communication between the web interface and the policy engine. The *Policy Service* or policy engine manages policy sessions that contain *Policy Rules* and persistent *Policy Memory*. Policy rules specify the required behavior for data staging and cleanup operations in the system; these rules are separated from application logic so that they can be customized to fit the requirements of the deployed environment. Finally, *Policy Memory* retains the current knowledge stored in the policy engine; its state persists for the length of transfer and cleanup requests.

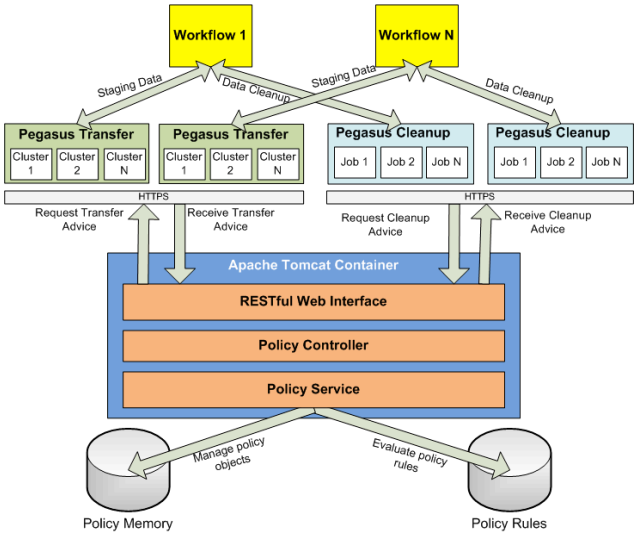

Fig. 1. Overview of Policy Service Architecture.

B. Integration of Policy Engine with Pegasus Workflow Management System

The Policy Engine applies specified policy rules to data transfer operations of a Pegasus workflow. Next, we describe how our Policy Engine is integrated with Pegasus for data staging and cleanup operations.

1) Data Staging Requests

During its planning phase, Pegasus adds to the workflow data staging tasks that move input data sets to resources where compute jobs will execute, that transfer intermediate results from one compute resource to another, and that transfer results to permanent storage.

Pegasus stages all required data input files for each compute job before releasing the job for execution. If Pegasus is configured to use task clustering (combining multiple workflow tasks into a single executable job) [9], then the data staging operations are grouped together, with the number of groups of transfers set by the clustering factor; clustering groups multiple transfers together for greater transfer efficiency. Fig. 2 illustrates data clustering with two data transfer tasks. In the non-clustering case, there is initialization overhead between transfer jobs. Grouping multiple transfers together can improve transfer efficiency by eliminating these overheads. Each cluster transfers or stages its files by initiating data transfers serially using the *Pegasus Transfer Tool* (PTT), which initiates data transfers using GridFTP, HTTP, or other transfer protocols.

We integrated the Policy Service with the Pegasus Workflow Management system via the Pegasus Transfer Tool. We modified the PTT so that when it receives a list of transfers to perform, it first sends this list of transfer requests to the Policy Engine, which applies policies to the pending transfers. Based on these policies, the Policy Service can assign priorities to the transfers, group the transfers based on specified criteria, and offer advice on appropriate transfer parameters.

The PTT sends transfer requests to the Policy Controller via HTTP using its RESTful Web Interface. The Policy Controller delegates this list of transfers to the Policy Service. The Policy Service then adds the list of transfers to the Policy Memory and evaluates the state of the policy session against the set of predefined Policy Rules. After these rules have been enforced, the Policy Service constructs a modified list of transfers and sends the list back to the Pegasus Transfer Tool through the Policy Controller and RESTful Web Interface. The Policy Service assigns each transfer a unique ID so that the transfers can be monitored and modified.

Modifications to the transfer list by the Policy Service may include removing duplicate transfers, grouping transfers for greater efficiency, and modifying transfer parameters. Based on the state that it maintains about pending and completed transfers, the Policy Service identifies duplicate transfer requests that are in progress or have already been successfully staged by the current workflow or by another workflow. The Policy Service removes those duplicate transfers from the list that is returned to the PTT for transfer execution.

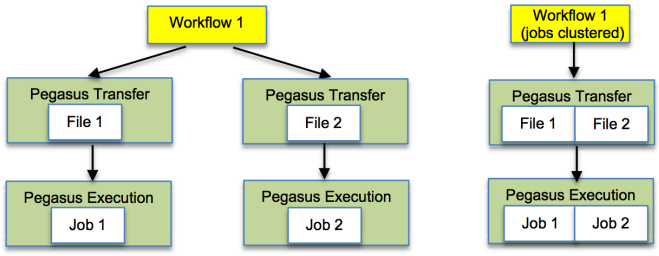

Fig. 2. Illustration of clustering with 2 data jobs and 2 transfers performed at a time.

The Policy Service can also group transfers together based on specified policies. For example, we implemented policy rules that sort the list of transfers by source and destination URLs. The Policy Service then assigns a common Group ID to transfers with the same source and destination URLs before returning the list of transfers to the PTT, so that the transfer client can perform these operations in a single session to achieve greater transfer efficiency.

The Policy Service also gives advice on transfer parameters, such as the desired number of parallel streams for each transfer, to manage and optimize the throughput of concurrent staging operations between source and destination hosts. We use this capability in the greedy and balanced stream allocation policies described in Section III.

Once it receives the modified list of transfers from the Policy Engine, the Pegasus Transfer Tool processes all the transfers in each group sequentially, using the sorted order and transfer parameters specified by the Policy Engine.

After the Pegasus Transfer Tool completes the specified data transfers, it sends a list of completed transfers to the Policy Service. The detailed state about successfully completed transfers is removed from the Policy Memory; however, the Policy Service maintains information about the location of staged files so that it can prevent subsequent staging operations from restaging the same files.

2) Cleanup Requests

Workflows also include cleanup jobs that delete files (such as intermediate results) that are no longer needed for workflow execution. Each Pegasus cleanup process submits the list of files that it is assigned to remove to the Policy Server via HTTP using its RESTful Web Interface. As before, the Policy Controller delegates this list of files to the Policy Service, which adds the list of files to be deleted to the Policy Memory. The Policy Service then evaluates the state of the policy session against the set of predefined Policy Rules. The Policy Service assigns each cleanup operation a unique ID so that operations can be monitored and modified. If there is a duplicate cleanup request and the cleanup operation is in progress or completed, the Policy Service removes the current operation from the cleanup list. If the Policy Service receives a cleanup request for a file that is in use by other workflows, then it removes the cleanup operation from the list. The Policy Service returns the modified list of cleanup operations to the Pegasus cleanup process, which performs the cleanup operations.

Once these file deletions are complete, the cleanup process sends the Policy Service the list of successful cleanups operations via HTTP using its RESTful Interface. The Policy Controller delegates the list of completed cleanups to the Policy Service, which removes state about the completed cleanups from Policy Memory.

III. Policy Overview

Our current research focuses on three policy alternatives for staging and cleanup operations for workflows. Two (greedy and balanced) are focused on resource allocation, to manage the number of parallel streams between source and destination hosts. The third policy assigns priorities to data staging jobs based on the structure of the workflow. Although we explore a number of different algorithms, the experiments in this paper focus on the greedy algorithm described next.

a) Greedy Allocation Algorithm

The Greedy Allocation Algorithm enforces policies related to controlling the number of streams between a source and destination pair, with the goal of increasing the efficiency of data transfers. A site or Virtual Organization administrator provides as input to the Policy Service a threshold number of streams that should be allowed between a source and destination pair. When a new request comes in for a transfer between that source and destination, the policy engine checks its state regarding how many streams have already been allocated for ongoing transfers. If there are sufficient streams remaining below this threshold to satisfy the new request, then the requested number of streams is allocated to that transfer. However, if a new transfer will exceed the threshold set by the site or VO administrator, then the new transfer is allocated fewer streams (only one stream, if the threshold has been reached). The goal of the greedy algorithm is to allocate all transfers their requested number of streams up to the threshold value; additional transfers are allowed to proceed with a smaller number of streams to avoid starvation. As transfers complete and free up streams, those streams are allocated to new transfers (but currently not to ongoing transfers).

b) Balanced Allocation Algorithm

The Balanced Allocation Algorithm uses information about the Pegasus clustering factor to allocate streams between a source and destination host. The clustering factor is specified during workflow planning and indicates the maximum number of clustered jobs at the same horizontal level in the workflow. The goal of this policy is to ensure that each cluster receives a similar amount of bandwidth between a source and destination pair. Pegasus uses a clustering factor to group together data staging operations; for each cluster, at most one data transfer operation will be performed at a time, so the clustering factor is equivalent to the number of transfer operations that will be performed concurrently. Our Balanced Allocation policy attempts to balance the available bandwidth among the specified number of clusters. The cluster factor for the workflow is provided as an input to the Policy Service. When the Policy Service receives a transfer request from a particular cluster, it allocates the requested number of parallel streams until the cluster threshold is exceeded. Transfers initiated by a cluster that arrive after the cluster threshold is met are allocated a single stream. Thus, if one cluster's transfer requests arrive later than another cluster's, the later requests will not be starved, because available resources have been reserved for each cluster in a balanced allocation.

c) Structure-Based Job Priorities

Our workflows are structured as directed acyclic graphs. Thus, when staging data to the computations, it may be advantageous to consider the relationships among the workflow nodes when making data staging decisions. For example, it is more important to stage data to a root job before staging data to

other jobs that depend on that root job. Based on this, we can augment the workflow with priorities that indicate the relative importance of staging data to each workflow component.

We can assign priorities to the workflow components based on various graph traversal algorithms: breadth-first search, depth-first search, and two graph node analysis algorithms called *direct-dependent-based* and *dependent-based*. The breadth- and depth-first search algorithms are standard graph traversal algorithms; they assign priorities based on the order of graph traversal, with higher priorities assigned to the nodes traversed earlier. The direct-dependent-based algorithm assigns the priorities based on the number of children a node has (the node's fan-out), so the node with the largest number of children has the highest priority. The dependent-based algorithm assigns the highest priority to the node with the most total descendants (not just direct children).

The Policy service can then use the priorities to determine the order of the transfers to be performed as well as the number of streams to allocate for particular data transfers.

IV. Policy Rules

The Policy Service is implemented using the Drools open source policy engine [10]. Here, we describe the core policy rules that we developed in the current phase of the project.

A. Policy rules that apply to all transfers

When the Policy Engine receives a list of transfers from the Pegasus Transfer Tool for evaluation, it applies the policies shown in Table I to each transfer.

TABLE I. Policies that apply to all transfers

Policies Enforced for All Transfers
Insert new transfers into policy memory
Remove duplicate transfers from the transfer list
Remove transfers from the transfer list that are already in progress
Create a resource for a new transfer to track the resulting staged file
Associate a transfer with a resource to track the number of workflows using the staged file
Generate a unique group ID for a source and destination host pair
Assign the group ID to a transfer based on its source and destination host pair
Detach a transfer from the resource when it requests to cleanup the resource's staged file
Remove cleanups from the cleanup list that specify resources that have other transfers using the staged files
Insert new cleanups into policy memory for resources that no longer have transfers using their staged files
Assign a default level of parallel streams to a transfer
Remove a transfer that has completed
Remove a transfer that has failed
Ensure each transfer has at least one parallel stream assigned
Sort the list of transfers by the source and destination URLs

B. Greedy Allocation Rules

The following describes the policy rules that allow a limited number of parallel streams to be assigned between a source and destination host pair using a greedy allocation algorithm, where transfers are allocated their requested number of parallel streams until the threshold is exceeded. Transfers that are initiated after this threshold is reached are allocated a single stream.

TABLE II. Greedy Algorithm Policies

Policy Rules for Greedy Algorithm
Retrieve the parallel streams threshold defined between a source and destination host
Enforce the maximum number of parallel streams on a transfer
If the number of requested streams would exceed the maximum streams threshold, then allocate only the number of streams that does not exceed the threshold
If the threshold has been reached or exceeded, allocate one stream for the new transfer
Record the number of parallel streams used by a transfer against the defined threshold

C. Balanced Allocation Algorithm Rules

The following describes the policy rules that allow a limited number of parallel streams to be assigned between a source and destination host pair using equal allocation among all clusters that run in parallel, where each cluster is allowed an equal share of the total streams between the hosts. Transfers on the cluster are allocated their requested number of parallel streams until the cluster threshold is exceeded. Transfer requests that arrive later from other clusters are therefore not starved because available resources have already been reserved for use by each cluster.

TABLE III. Balanced Allocation Algorithm Policies

Policy Rules for Balanced Algorithm
Retrieve the parallel streams threshold defined for a single cluster between a source and destination host
Retrieve the number of clusters used in the system
Enforce the max number of parallel streams on a transfer that violates the number of available streams below the threshold on its cluster
Record the number of parallel streams used by a transfer against the defined cluster threshold

The implementation of rules related to the structure-based job priorities is left for future work.

V. The effect of policies on workflow performance

Next, we present our evaluation of the greedy allocation policy on the performance of the Montage astronomy workflow [11, 12]. Montage is an astronomy application that is used to construct large image mosaics of the sky, and it is widely used as a workflow benchmark. The input files are images re-

projected onto a sphere, and overlap is calculated for each input image. The application re-projects input images to the correct orientation while keeping background emission level constant in all images. The images are then added by rectifying them into a common flux scale and background level. Finally, the re-projected images are co-added into a final mosaic.

Our early experiments showed that the data staging time of the Montage workflow was a relatively small proportion of the total workflow execution time compared to computational time. To better explore the impact of policies on a data-intensive application, we augmented the Montage 1 degree square workflow to stage one additional data file for each data staging job, as illustrated in Fig. 3. This scenario is representative of emerging big data applications that will stage increasing amounts of data for analysis. The sizes of these additional data files are 10 MBytes, 100 MBytes, 500 MBytes and 1 GByte in our experiments. There are 89 data staging jobs in this Montage workflow.

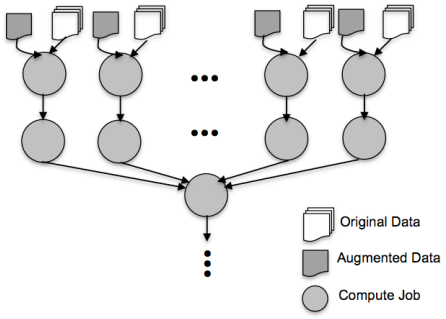

Fig. 3. Illustrates the augmented workflow with an additional file per data staging job.

The experiments were performed using a one-degree square Montage workflow on 9 nodes of the Obelix cluster at the USC Information Sciences Institute. The Obelix cluster nodes are Intel Xeon 6-core 2.67 GHz processors with 40 GBytes of RAM. The cluster uses NFS on its 1 GBit local area network.

Montage input image files were stored on the Obelix cluster and staged in via an Apache web server for processing on cluster compute nodes. Pegasus was configured to use no clustering (one stage-in job per compute job), a local job limit of 20 (so that at most 20 data staging jobs will be released at once), cleanup enabled, and five retries on failure per job.

The Policy Service ran as a RESTful web service on an Apache Tomcat server inside the Information Sciences Institute network. Prior to each test, the policy service was configured to use a specified default number of streams per transfer and a maximum number of allowable streams between two hosts. In the tests where policy was enabled, Pegasus registered its transfer with the policy service and received advice on the number of streams to use in the transfer.

Having Pegasus call out to an external service to make data placement decisions is potentially beneficial to the performance of workflow tasks, but this approach incurs overheads for the service calls, which include the time the policy engine takes to make its decisions.

As described earlier, in addition to the input image files that were used in the Montage computation jobs, our executable workflows were modified to include an additional input file per stage-in job. These large additional files were staged over the wide area network from a FutureGrid site at the Texas Advanced Computing Center to the NFS mounted drive for Obelix. This data staging is illustrated in Fig. 4. A GridFTP version 6.5 server runs on a Virtual Machine in the FutureGrid Alamo cloud at the Texas Advanced Computing Center and stages data to our computations. Bandwidth for large transfers between the FutureGrid VM and ISI was about 28 MBits/sec.

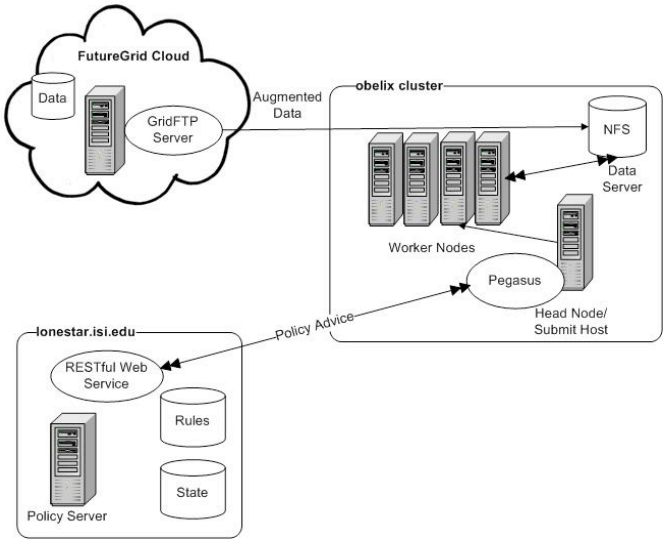

Fig. 4. Illustrates staging of additional data files from FutureGrid cloud.

In our experiments, we plot the workflow execution time versus a varying number of default streams per transfer. In Fig. 5, we fix the threshold at 50 streams for the greedy algorithm and vary the size of the additional files being staged with the Montage workflow. For the other graphs, we fix the size of the files being staged and vary the threshold of streams allocated by the greedy algorithm. We ran each experiment at least 5 times. The graphs show average values with error bars showing standard deviation.

TABLE IV. MAXIMUM STREAMS FOR SIMULTANEOUS TRANSFERS

Greedy streams threshold	**Default streams per transfer**				
	4	**6**	**8**	**10**	**12**
No policy case	80				
50	57	61	63	65	65
100	80	103	107	110	111
200	80	120	160	200	203

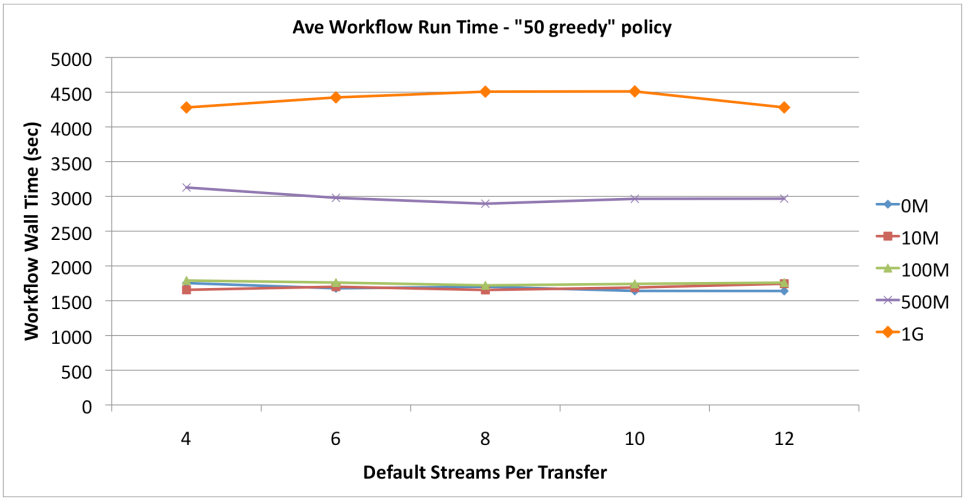

Fig. 5. Shows the effect on workflow execution time of increasing the file size of the additional staged data for 50 stream threshold and increasing number of default streams per transfer.

Table IV shows the maximum streams that will be allocated for specified values for default streams per transfer and threshold for the greedy algorithm. In the *no policy case*, with up to 20 data staging jobs running and each using a default of 4 streams, at most 80 simultaneous streams will be active for data transfers. With a *greedy threshold* of 50 streams and a *default allocation* of 8 streams, the first 6 staging jobs will receive an allocation of 8 streams (for a total of 48 streams); the next job will receive 2 streams (reaching the threshold of 50 streams); and the remaining 13 data staging jobs will receive 1 stream, for a total of 63 allocated streams. We will refer to Table IV to provide insights into the performance results that follow.

Fig. 5 shows the effect of increasing the size of the additional files staged with the Montage workflow with a threshold value of 50 streams for the greedy algorithm. The file sizes range from 0 (no additional data staged) to 1 GByte additional data transferred per data staging job. This is in contrast to the average size of 2 MBytes for stage-in files for the most data-intensive Montage job (*mProjectPP*); the runtime for these jobs is several seconds. The graph shows the workflow execution time as a function of the default streams per transfer. The additional file size has a significant effect on workflow execution time for file sizes over 100 Megabytes. By contrast, increasing the default number of streams per transfer has relatively little impact on performance.

Next, we show the impact of fixing the size of the additional data staging files and increasing the maximum number of streams used in our experiments from 50 to 200. We compare the performance of the greedy algorithm with the default performance for Pegasus without any policy applied.

Fig. 6 shows that when the additional data files staged with the Montage workflow are relatively small (10 Megabytes), there is not much difference in the behavior as the number of maximum streams increases. The use of the policy engine performs slightly better (at most 6%) than default Pegasus with no policy (shown as a single data point with a blue circle at 4 streams per transfer) at lower values for default streams per transfer. We note that the greedy policy with a threshold of 50 maximum streams has the best performance of the three threshold values. According to Table IV, this policy allocates a maximum of 57 to 65 total streams, compared to up to 111 streams for the 100 greedy policy and up to 203 streams allocated for the 200 greedy policy. We conclude that limiting the number of allocated streams can result in better performance between source and destination sites by avoiding overwhelming host resources at the source and destination and the network resources between them.

Similarly, Fig. 7 shows workflow performance when the size of additional files transferred is 100 MBytes per data staging job. This graph shows a more significant difference in performance for the three threshold values for the greedy algorithm. The best performance is achieved with 50 maximum streams, which outperforms default Pegasus with no policy by 6.7% at 8 streams per transfer. The performance with a threshold of 200 is 28.8% worse than for a threshold of 50 with 8 streams per transfer. This suggests that the greedy algorithm can over-allocate the number of streams between the source and destination, resulting in worse performance. A threshold value of 50 works best for this environment and transfer size.

Fig. 8 shows performance when the file size for extra staged data increases to 500 MBytes per data staging job. For the large files in these experiments, a threshold value of 50 streams performs best, with a threshold of 100 streams also providing good performance. Both threshold values perform significantly better than default Pegasus using no policy. For example, the threshold of 50 with 8 streams per transfer performs 14% faster than default Pegasus. A threshold value of 200 performs well for low values for streams per transfer, but performs poorly for larger values. Based on Table IV, we speculate that overall performance of the workflow declines because these relatively long-running, 500 megabyte transfers overwhelm the available resources when the maximum number of allocated steams reaches 160 for 8 streams per transfer and 203 for 12 streams per transfer.

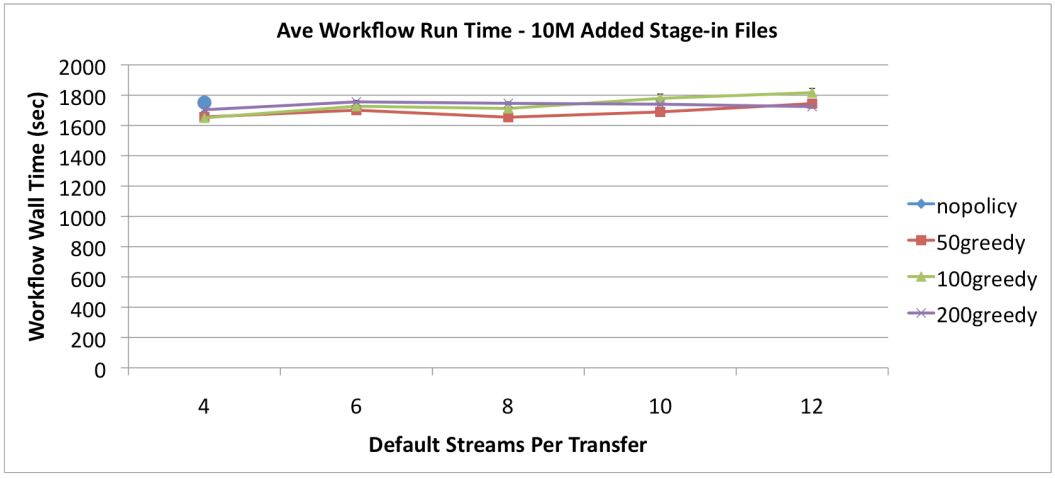

Fig. 6. Workflow Performance with staging of additional 10 MByte files. The blue circle indicates the single point for the nopolicy case, where default Pegasus runs with 4 streams per transfer.

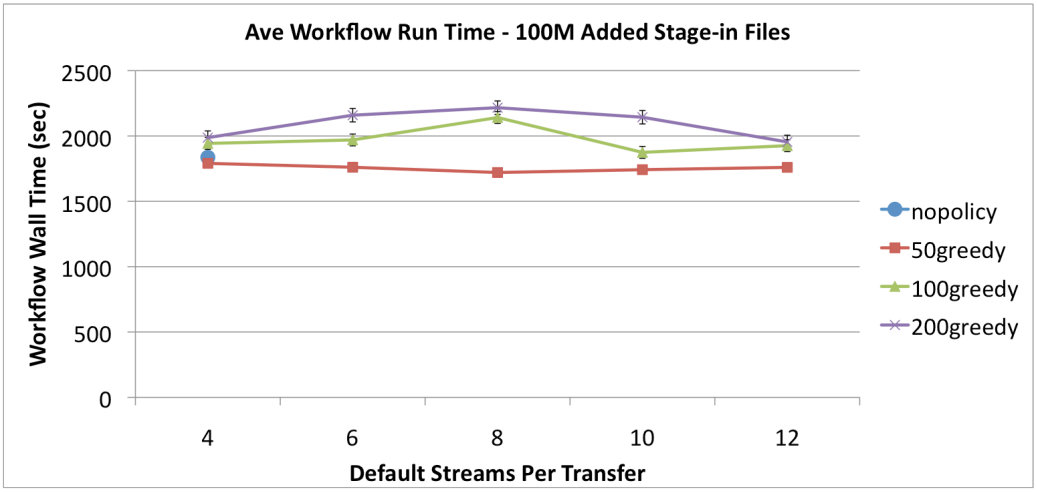

Fig. 7. Workflow Performance with staging of additional 100 MByte files.

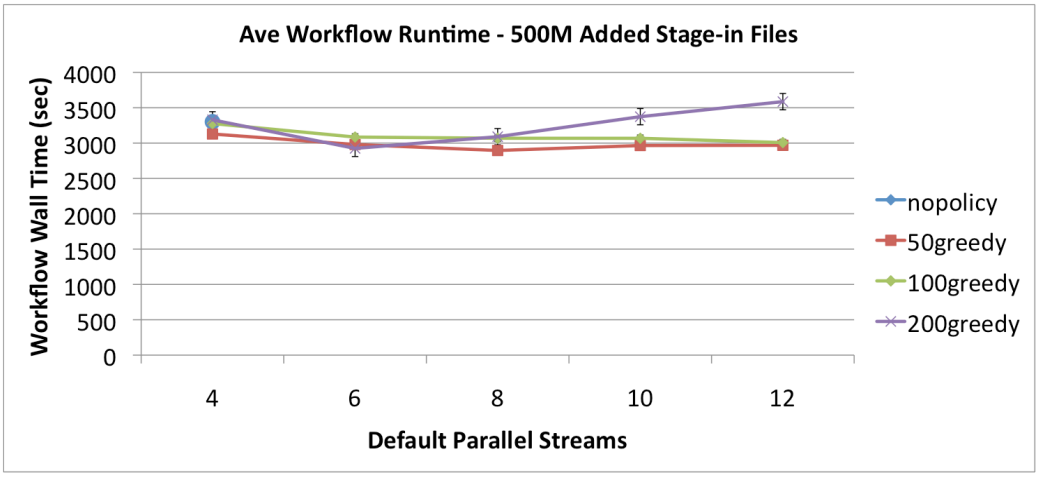

Fig. 8. Workflow performance of staging additional 500 MByte files.

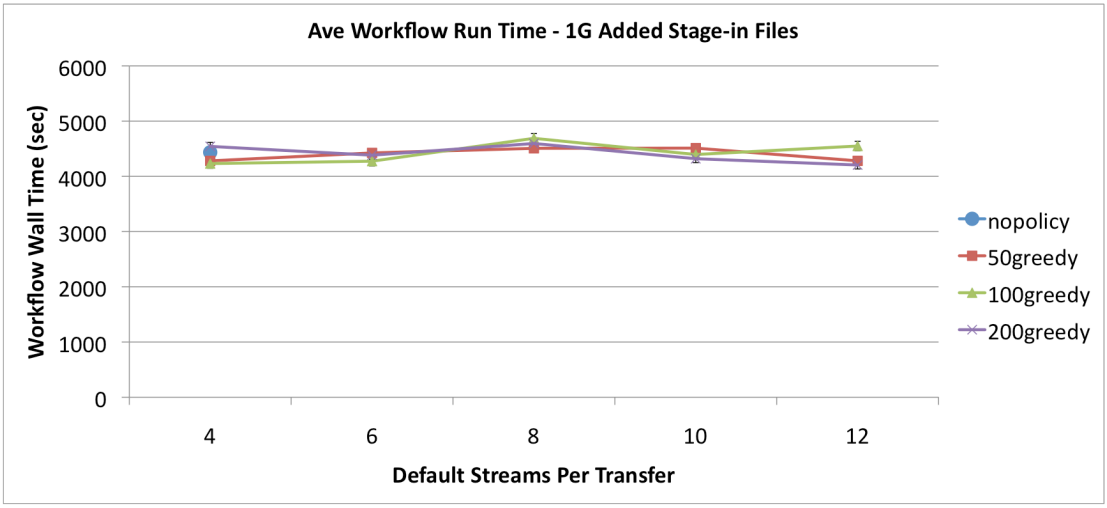

Fig. 9. Workflow performance with additional data staging with 1 GByte files.

Finally, Fig. 9 shows the performance when the file size for additional staging is 1 GByte per data staging job. For these experiments, there is no clear advantage to using any of the greedy threshold values over the default Pegasus performance. Based on our earlier experiments, we speculate that these large, long-running data transfers make use of all available resources between the source and destination regardless of the policy used.

We plan to run further experiments in higher-performance environments to study the effect of policies on resource allocation. We will also explore machine learning algorithms to identify the data transfer settings (such as the threshold number of streams) that are the most beneficial for applications; we speculate that these will depend on available host resources and on network performance between computing and data storage sites.

VI. Related Work

A. Data Management and Placement for Scientific Applications

In earlier work, we measured the impact of data pre-staging on Pegasus [13] workflows. We prestaged input data sets near expected computation sites for the Montage workflow and measured the performance improvement when the workflow accessed prestaged data. We categorized placement operations and explored the ability of workflow systems to provide hints regarding where to place data.

We also examined data management for workflows executing in storage constrained environments [14, 15]. We showed that the overall data footprint for a workflow can be reduced by removing files that are no longer required by the workflow during the remaining execution (i.e., clean-up operations) [14]. We also presented an algorithm that finds feasible solutions for task assignment in storage constrained environments [15].

Ranganathan et. al [16] design and provide simulation results for a scheduling framework for grid environments that supports combinations of job scheduling and data replication algorithms. They achieve the best performance by scheduling jobs where data are located and replicating popular data sets at each site, a scheme that decouples data and computational scheduling.

Stork, developed by Kosar et al. [17, 18], is a data placement scheduler that schedules, monitors and manages data placement jobs with fault tolerance. This group also proposed a data-intensive scheduling paradigm [19] that schedules data placement jobs independently of compute jobs in scientific workflows; these different job types are managed by Stork and the DAGMan directed acyclic graph manager [20], respectively. After DAGMan receives an executable workflow from Pegasus, it divides the workflow execution between Condor and Stork, with Stork handling all data placement jobs. In our approach, the Pegasus Transfer Tool still performance data placement and staging tasks, but it first consults the Policy Service to get advice on the ordering of requests and desirable transfer parameters for requests.

Yuan et al. [21] propose a matrix based k-means clustering strategy for data placement in scientific workflows. They assume that datasets are located in different data centers, while we assume a single data storage resource. They use two strategies that group the existing datasets into k data centers and dynamically cluster newly generated datasets to their optimal destinations based on dependencies during runtime. In our approach, the Policy Service groups together requests with the same source and destination hosts and orders requests based on algorithms for resource allocation or task priorities.

B. Policies for Scientific Data Management

In earlier work, we developed initial prototype policy services and evaluated the impact of simple policies on workflow performance [22-25]. Our current policy architecture and implementation are scalable and robust and

support a broader range of policies, including those evaluated here that control resource allocation for data transfers.

Other large scientific collaborations have developed complex systems for management of data distribution and replication, including the Physics Experiment Data Export (PheDEx) [26, 27] system for high-energy physics and the Lightweight Data Replicator (LDR) [28] for gravitational-wave physics. In each collaboration, the Virtual Organization has defined policies for distribution and replication of data sets to provide high availability and make datasets available to scientists at their institutions or near where computations are likely to run. Wasson et al. [29] examine policies for sharing resources among the resources and users of a VO, either by giving all participants an equal share or based on the contribution of resources by each participant. MyPolyMan by Feng et al. [30] makes permit/denial decisions regarding data movement operations.

The integrated Rule-Oriented Data System (iRODS) [31] uses a rule based system to implement data management constraints. The iRODS system extends the Storage Resource Broker (SRB) [32]. iRODS uses a modular architecture to apply and monitor policies for digital curation. The rule engine evaluates which rule sets are to be enforced and activates the corresponding micro services. iRODS enforces a wide range of policies related to replication, data consistency, provenance and metadata management. By contrast, our Policy Service limits itself to giving runtime advice on transfer order and transfer parameters.

VII. Summary

As scientific data continue to be generated and consumed at ever-increasing rates, the performance of data staging will have an increasing impact on the overall performance of scientific workflows. The goal of our work is to improve the overall performance of scientific workflows by using policy to improve the performance of data staging into and out of computational resources. In this paper, we describe the integration of a policy service with the Pegasus Workflow Management System and describe a range of policies that we are exploring to improve workflow performance. We implement and evaluate a greedy allocation algorithm that enforces controls the number of streams allocated between a source and destination pair, with the goal of increasing the efficiency of data transfers. Our evaluation increases the amount of data staging for the Montage astronomy workflow by staging an additional data file for each data staging job in the workflow.

Our experiments show that there are significant performance benefits to controlling the allocation of streams between source and destination hosts as the amount of data transfer increases, up to the point where the available resources are fully utilized.

In our future work, we plan to do much more extensive performance evaluation of the greedy allocation policy as well as the balanced allocation and the workflow structure-based approaches that we have described. Our goal is to identify a set of policies that will be advantageous to big data scientific workflows. We also plan to explore machine learning algorithms to help us learn what data transfer settings (such as the threshold number of streams) are the most beneficial for the applications. Based on our current results, we assume that these will depend on available host resources and on the network performance between computing and data storage sites. Finally, we will study the scalability of the centralized policy service when planning multiple complex workflows and explore strategies for distribution and replication of policy logic to improve reliability.

Acknowledgment

This work was supported by NFS under grant number IIS-0905032 and used the FutureGrid environment, which was supported by NSF grant number 0910812. Craig Ward contributed to the implementation of the Policy Service.

References

[1] A. J. G. Hey, S. Tansley, and K. M. Tolle, *The fourth paradigm: data-intensive scientific discovery*: Microsoft Research Redmond, WA, 2009.

[2] I. Taylor, E. Deelman, D. Gannon, and M. Shields, "Workflows in e-Science," Springer, 2006.

[3] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems,* vol. 25, pp. 528-540, 2009.

[4] J. Gu, D. Smith, A. L. Chervenak, and A. Sim, "Adaptive Data Transfers that Utilize Policies for Resource Sharing," in *2nd Int'l Workshop on Network Aware Data Management, in conjunction with SC12 Conference* Salt Lake City, UT, 2012.

[5] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming Journal,* vol. 13, pp. 219-237, 2005.

[6] D. Howe, M. Costanzo, P. Fey, T. Gojobori, L. Hannick, W. Hide, D. P. Hill, R. Kania, M. Schaeffer, and S. St Pierre, "Big data: The future of biocuration," *Nature,* vol. 455, pp. 47-50, 2008.

[7] F. Frankel and R. Reid, "Big data: distilling meaning from data," *Nature,* vol. 455, pp. 30-30, 2008.

[8] "NSF Leads Federal Efforts In Big Data," National Science Foundation, http://www.nsf.gov/news/news_summ.jsp?cntn_id=123607.

[9] G. Singh, M. H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, "Workflow task clustering for best effort systems with Pegasus," in *Mardi Gras Conference*, Baton Rouge, LA, 2008, p. 9.

[10] Drools project, "Drools, http://www.jboss.org/drools/."

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

[12] G. B. Berriman, et al. , "Montage: A Grid-Enabled Image Mosaic Service for the NVO," in *Astronomical Data Analysis Software & Systems (ADASS) XIII*, 2003.

[13] A. L. Chervenak, E. Deelman, M. Livny, S. Mei-Hui, R. Schuler, S. Bharathi, G. Mehta, and K. Vahi, "Data placement for

scientific applications in distributed environments," in *Grid Computing, 2007 8th IEEE/ACM International Conference on*, 2007, pp. 267-274.

[14] G. Singh, K. Vahi, A. Ramakrishnan, G. Mehta, E. Deelman, H. Zhao, R. Sakellariou, K. Blackburn, D. Brown, S. Fairhurst, D. Meyers, G. B. Berriman, J. Good, and D. S. Katz, "Optimizing workflow data footprint," *Sci. Program.,* vol. 15, pp. 249-268, 2007.

[15] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling data-intensive workflows onto storage-constrained distributed resources," in *proceedings of the 7th IEEE Symposium on Cluster Computing and The Grid (CCGrid)*, 2007.

[16] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, 2002, pp. 352-358 1082-8907.

[17] T. Kosar and M. Livny, "A framework for reliable and efficient data placement in distributed computing systems," *J. of Parallel and Distributed Computing,* vol. 65, pp. 1146-1157, 2005.

[18] T. Kosar and M. Livny, "Stork: making data placement a first class citizen in the grid," in *24th International Conference on Distributed Computing Systems*, 2004, pp. 342-349.

[19] T. Kosar and M. Balman, "A new paradigm: Data-aware scheduling in grid computing," *Future Generation Computer Systems,* vol. 25, pp. 406-413, April 2009 2009.

[20] J. Frey, "Condor DAGMan: Handling Inter-Job Dependencies," http://www.bo.infn.it/calcolo/condor/dagman/

[21] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems,* vol. 26, pp. 1200-1214, 2010.

[22] M. A. Amer, W. Chen, S. Hopkins, E. Griffiths, A. Chervenak, and E. Deelman, "Separating Workflow Management and Data Staging to Improve the Performance of Scientific Workflows (student poster paper)," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10 Conference)* New Orleans, LA, 2010.

[23] M. A. Amer, A. Chervenak, and S. Alspaugh, "A Policy Based Data Placement Service (student poster paper)," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'09 Conference)*, Portland, OR, 2009.

[24] S. Alspaugh, A. Chervenak, and E. Deelman, "Policy-Driven Data Management for Distributed Scientific Collaborations using a Rule Engine (student poster paper)," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'08 Conference)*, Austin, TX, USA, 2008.

[25] M. Amer, W. Chen, and A. L. Chervenak, "Improving Scientific Workflow Performance using Policy Based Data Placement," in *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2012)*, Chapel Hill, North Carolina USA, 2012.

[26] T. A. Barrass, et al., "Software Agents in Data and Workflow Management," in *Computing in High Energy and Nuclear Physics (CHEP) 2004*, Interlaken, Switzerland, 2004.

[27] J. Rehn, T. Barrass, D. Bonacorsi, J. Hernandez, I. Semeniouk, L. Tuura, and Y. Wu, "PhEDEx high-throughput data transfer management system," in *Computing in High Energy and Nuclear Physics (CHEP) 2006*, Mumbai, India, 2006.

[28] LIGO Project, "Lightweight Data Replicator, http://www.lsc-group.phys.uwm.edu/LDR/," 2004.

[29] G. Wasson and M. Humphrey, "Toward explicit policy management for virtual organizations," in *IEEE Workshop on Policies for Distributed Systems and Networks (POLICY,'03)*, 2003.

[30] J. Feng, L. Cui, G. Wasson, and M. Humphrey, "Policy-Directed Data Movement in Grids," in *Proceedings of 12th International Conference on Parallel and Distributed Systems (ICPADS 2006)*, 2006, pp. 12-15.

[31] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder, "A prototype rule-based distributed data management system," in *HPDC Workshop on Next-Generation Distributed Data Management*, 2006.

[32] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research* Toronto, Ontario, Canada: IBM Press, 1998.