Scheduling Data-Intensive Workflows onto Storage-Constrained Distributed Resources

Arun Ramakrishnan[1], Gurmeet Singh[2], Henan Zhao[3], Ewa Deelman[2], Rizos Sakellariou[3], Karan Vahi[2]
Kent Blackburn[4], David Meyers[4,5], and Michael Samidi[4]

[1] University of Southern California, Los Angeles, USA
arunrama@usc.edu
[2] USC Information Sciences Institute, 4676 Admiralty Way, Marina Del Rey, CA 90292, USA.
{gurmeet,deelman,vahi}@isi.edu
[3] School of Computer Science, University of Manchester, Manchester M13 9PL, UK.
{hzhao,rizos}@cs.man.ac.uk
[4] LIGO Laboratory, California Institute of Technology, MS 18-34, Pasadena, CA 91125, USA
{kent,dmeyers,msamidi}@ligo.caltech.edu
[5] Northrop Grumman Information Technology, 320 North Halstead Suite 170, Pasadena, CA 91107, USA

Abstract

In this paper we examine the issue of optimizing disk usage and of scheduling large-scale scientific workflows onto distributed resources where the workflows are data-intensive, requiring large amounts of data storage, and where the resources have limited storage resources. Our approach is two-fold: we minimize the amount of space a workflow requires during execution by removing data files at runtime when they are no longer required and we schedule the workflows in a way that assures that the amount of data required and generated by the workflow fits onto the individual resources. For a workflow used by gravitational-wave physicists, we were able to improve the amount of storage required by the workflow by up to 57%. We also designed an algorithm that can not only find feasible solutions for workflow task assignment to resources in disk-space constrained environments, but can also improve the overall workflow performance.

1. Introduction

Today, scientific analyses are frequently composed of several application components, each often designed and tuned by a different researcher. Recently, scientific workflows [1, 2] have emerged as a means of combining individual application components into large-scale analysis by defining the interactions between the components and the data that they rely on. Scientific workflows provide a systematic way to capture scientific methodology by supplying a detailed trace (provenance) of how the results were obtained. Additionally, workflows are collaboratively designed, assembled, validated, and analyzed. Workflows can be shared in the same manner that data collections and compute resources are shared today among communities. The scale of the analysis and thus of the workflows often necessitates that substantial computational and data resources be used to generate the required results. CyberInfrastructure projects such as the TeraGrid [3] and the Open Science Grid (OSG) [4] can provide an execution platform for workflows, but they require a significant amount of expertise on the part of the scientist to be able to make efficient use of them.

Pegasus [5, 6], which stands for Planning for Execution in Grids, is a workflow mapping engine developed and used as part of several projects in physics [7], astronomy [8], gravitational-wave science [9, 10], earthquake science [11], neuroscience [12], and others. Pegasus bridges the scientific domain and the execution environment by automatically mapping the high-level workflow descriptions onto distributed resources such as the TeraGrid, the Open Science Grid, and others. Pegasus relies on the Condor DAGMan [13] workflow engine to launch workflow tasks and maintain the dependencies between them. Pegasus enables scientists to construct workflows in abstract terms without worrying about the details of the underlying CyberInfrastructure or the particulars of the low-level specifications required by the underlying middleware (Globus [14] and Condor [15]). Pegasus is used day-to-day to map complex, large-scale scientific workflows with thousands of tasks processing terabytes of data onto the Grid. As part of the mapping, Pegasus automatically manages data generated during workflow execution by staging them out to user-specified locations, by registering them in data catalogs, and by capturing their provenance information.

When workflows are mapped onto distributed resources, issues of performance related to workflow job scheduling and data replica selection are most often the primary drivers in optimizing the mapping. However, in the case of data-intensive workflows it is possible that typical workflow mapping techniques produce workflows that are unable to execute due to the lack of disk space necessary for the successful execution. In this paper we examine two issues related to this problem. The first deals with optimizing the amount of space that a workflow (or a portion of a workflow) requires to run on a given resource and the second explores a scheduling technique that takes into account the space needed by the workflow when deciding where to run the jobs.

The remainder of the paper is organized as follows. The next section provides further motivation for this work by examining a Laser Interferometer Gravitational Wave Observatory (LIGO) [16] application which requires large amounts of space and targets the OSG as its execution environment. This application exhibits behaviours typical in many scientific workflows used today. Section 3 describes an algorithm for reducing the amount of space required by a workflow followed by showing the space usage improvements in the case of a small simulated LIGO application. Sections 4 and 5 describe an algorithm and show the results of scheduling workflows to space-constrained resources. Finally we give an overview of related work and include concluding remarks.

2. Motivation

LIGO [16] is a network of gravitational-wave detectors, one located in Livingston, LA and two co-located in Hanford, WA. The observatories' mission is to detect and measure gravitational waves predicted by general relativity—Einstein's theory of gravity—in which gravity is described as due to the curvature of the fabric of time and space. One well-studied source of gravitational waves is the inspiral and coalescence of a pair of dense, massive astrophysical objects such as neutron stars and black holes. Such binary inspiral signals are among the most promising sources for LIGO [17, 18]. Gravitational waves interact extremely weakly with matter, and the measurable effects produced in terrestrial instruments by their passage will be miniscule. In order to establish a confident detection or measurement, a large amount of data needs to be acquired and analyzed which contains the strain signal that measures the passage of gravitational waves. LIGO applications often require on the order of a terabyte of data to produce meaningful results.

Data from the LIGO detectors is analyzed by the LIGO Scientific Collaboration (LSC) which possesses many project-wide computational resources. Additional resources would allow the LSC to extend its science goals. Thus, the LSC has been reaching out toward Grid deployments such as the OSG to extend their own capabilities. OSG supports the computations of a variety of scientific projects ranging from high-energy physics, biology, material science, and many others.

The shared nature of OSG resources imposes limits on the amount of computational power and data storage available to any particular application. As mentioned before, a scientifically meaningful run of the binary inspiral analysis requires a minimum of 221 GBytes of gravitational-wave data and approximate 70,000 computational workflow tasks.

The LIGO Virtual Organization (VO) is supported on nine distinct Compute Elements managed by other institutions supporting the OSG. Each Compute Element is an HPC or High Throughput Computer (HTC) resource, with, on average, 258 GB of shared scratch disk space. The shared scratch disk space is used by approximately 20 VOs with the OSG. The LIGO VO can not reserve space on these shared resources.

Currently Pegasus automatically generates a "cleanup workflow" that is run after a workflow has finished and the analysis results have been staged out to a user-specified location. The cleanup workflow deletes all data staged-in, and data products generated on the Compute Element. Statically cleaning up files after all the data processing occurs, entails significant overhead as the data processing for a single run may require a week of wall time. Opportunities exist to dynamically delete the input and intermediate data immediately after these data have been consumed by the jobs in the workfow. This can substantially reduce the storage requirements on the Compute Element during the data analysis.

Next we describe the algorithm that determines when a given data file is no longer needed and we use this algorithm to add dynamic cleanup jobs to the executable workflow.

3. Improving Workflow Data Storage Use

The algorithm described in this section adds a cleanup job for a data file when that file is no longer required by other tasks in the workflow or when it has already been transferred to permanent storage. The purpose of the cleanup job is to delete the data file from a specified resource. Since a data file can be potentially replicated on multiple resources (in case the compute tasks are mapped

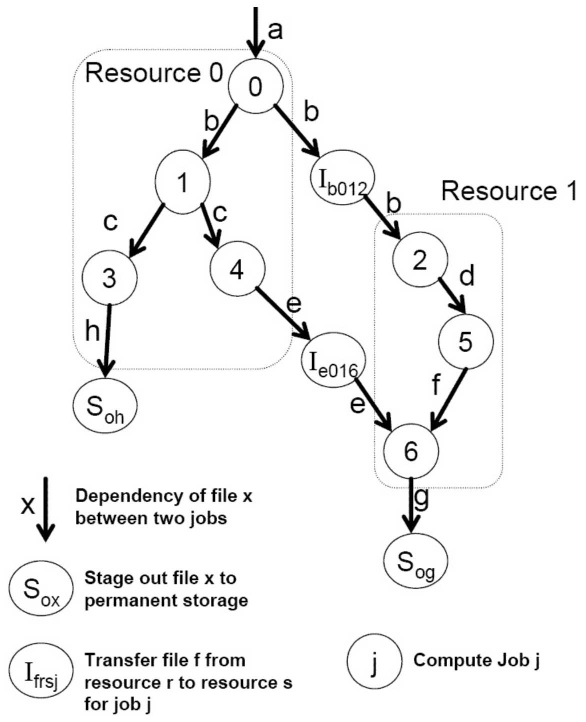
a
Resource 0
0
b
b
1
I_{b012}
Resource 1
c
c
b
3
4
2
d
e
h
5
I_{e016}
f
e
S_{oh}
6
g
S_{og}
x
Dependency of file x between two jobs
S_{ox}
Stage out file x to permanent storage
I_{frsj}
Transfer file f from resource r to resource s for job j
j
Compute Job j

Figure 1. Executable workflow with 7 compute jobs mapped to two resources.

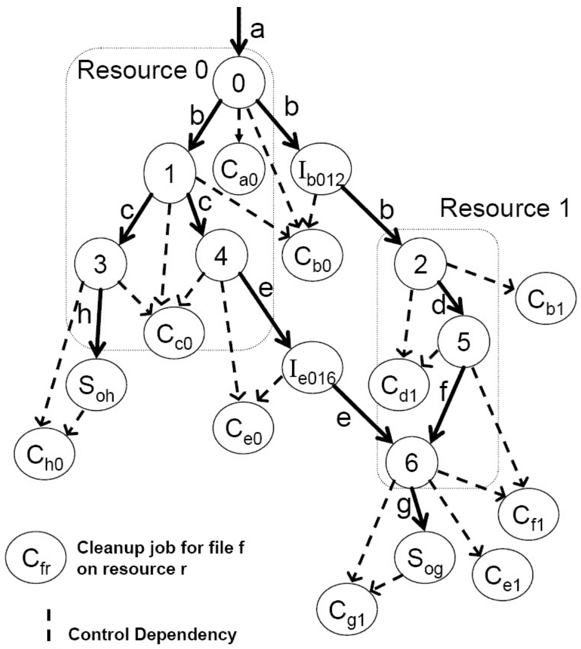

Figure 2. Cleanup nodes added to the executable workflow.

to multiple resources) the decision to add cleanup jobs are made on a per resource basis.

In order to illustrate the working of the algorithm, Figure 1 shows an executable workflow containing 7 compute jobs {0,1,..,6} mapped to 2 resources {0,1}. The algorithm first creates a subgraph of the executable workflow for each execution resource used in the workflow. The subgraph of the workflow on resource 0 contains jobs {0,1,3,4} and the subgraph on resource 1 contains jobs {2,5,6} (shown in figure 1). The cleanup nodes added to this workflow using the algorithm are shown in Figure 2. The cleanup job for removing file f on resource r is denoted as C_{fr}.

For each task in the subgraph, a list of files either required or produced by the task is constructed. For example list of files for task 1 mapped to resource 0 contains files b and c. For each file in the list, a cleanup job for that file on that resource is created (if it does not already exist) and the task is made parent of the cleanup job. Thus a cleanup job, C_{c0}, for removing file c on resource 0 is created and task 1 is made parent of this cleanup job. The cleanup jobs for some files might already have been created as a result of parsing previous tasks. For example, the cleanup job C_{b0} for removing file b on resource 0 already exists (as a result of parsing task 0). In this case the task being parsed is added as an parent of the cleanup job. Thus task 1 is added as a parent of cleanup job C_{b0}. When the entire subgraph has been traversed, there exists one cleanup job for every file required or produced by tasks mapped to the resource.

If a file required by a task is being staged-in from another resource, then the algorithm makes the cleanup job for the file on the source resource a child of the stage-in job, thus ensuring that the file is not cleaned up on the source resource before it is transferred to the target resource. For example, file b required by task 2 mapped to resource 1 is being staged-in from resource 0 using stage-in job I_{b012}, and so the cleanup job for file b on resource 0 (C_{b0}) is made a child of I_{b012}. Finally, if a file produced by a task is being staged-out to a storage location, the cleanup job is made a child of the stage-out job. For instance, the cleanup job C_{h0} for removing file h on resource 0 is made a child of the stage-out job S_{oh} that stages out file h to permanent storage. By adding the appropriate dependencies, the algorithm makes sure that the file is cleaned up only when it is no longer required by any task in the workflow.

The pseudocode for the algorithm is shown in Figure 3. Its running time is $O(e+n)$, where e is the number of edges and n is the number of tasks in the executable workflow assuming that each edge represents the dependency of a particular file between two tasks. Multiple file dependencies between two tasks are represented by multiple edges. The

```
Input: Executable Workflow, r = 1..R (list of resources)
Output: Executable Workflow including cleanup jobs

For every resource r = 1..R
  Let Gr=(Vr,Er) be the subgraph induced by the tasks mapped to resource r
  For every job j in Vr
    For every file f required by j
      create cleanUpJob $C_{fr}$ for file f for resource r if it does not already exist
      add job j as parent of the cleanUpJob $C_{fr}$
      if file f is produced at another resource s
        Let $I_{frsj}$ = stage-in job for transferring file f from resource r to resource s for job j
        create cleanUpJob $C_f s$ for file f at resource s if it does not exist and make $I_{frsj}$ parent of $C_{fs}$
      End if
    End For
    For every file f produced by j
      create cleanUpJob Cfr for file f for resource r if it does not already exist
      add job j as parent of the cleanUpJob $C_{fr}$
      If f is being staged out to final storage, add $C_{fr}$ as child of the stage-out job $S_{of}$.
    End For
  End For
End For
```

Figure 3. Algorithm for adding cleanup jobs to an executable workflow.

algorithm makes sure that the workflow cleans up the unnecessary data files as it executes (by adding cleanup nodes to the executable workflow) and at the end there are no files remaining on the execution resources.

We use a simulated LIGO workflow to evaluate the performance of the above algorithm using a modified Grid simulator [19]. We use a workflow (Figure 4) which is a subset of those used for the current LSC binary inspiral analysis [20]. This workflow consists of 166 compute tasks and has the same topology as the inspiral analysis workflow. We replace the inspiral compute nodes with simulated tasks that have the same execution times and data requirements as an inspiral analysis in order to benchmark our algorithm. Our simulated analysis is therefore a good representation of large scale LIGO workflows. In this case the workflow is mapped to 4 homogeneous resources using a random mapping heuristic. In future, we plan to experiment using more advanced mapping strategies. During the simulation, the data stage-in tasks are executed as late as possible and the cleanup jobs are executed as early as possible in order to minimize the storage used.

Figure 5 shows the amount of storage used at the 4 resources as a function of time as the workflow executes both without and with the cleanup jobs. Without the cleanup nodes, the storage being used at the resources is monotonically increasing. However, with the cleanup jobs there is a considerable saving in the amount of storage used during the runtime of the workflow.

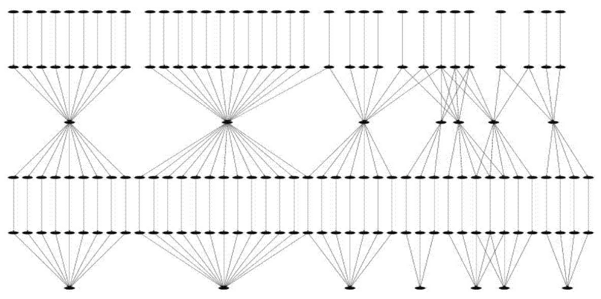

Figure 4. The Structure of the Scaled-Down Version of the Simulated LIGO Workflow. The Workflow Progresses Top to Bottom. Edges represent dependencies and vertices represent tasks.

Initially the storage used by both the approaches is the same. This is because this initial period is mostly used for staging-in the input data files to the resources and the execution of the top-level tasks. When the next level tasks finish execution ,their input files produced by the top-level tasks are no longer required and provide the first cleanup opportunity.

Table 1 shows the maximum amount of storage used at the execution resources both without and with cleanup. On average, the maximum storage used at any resource during

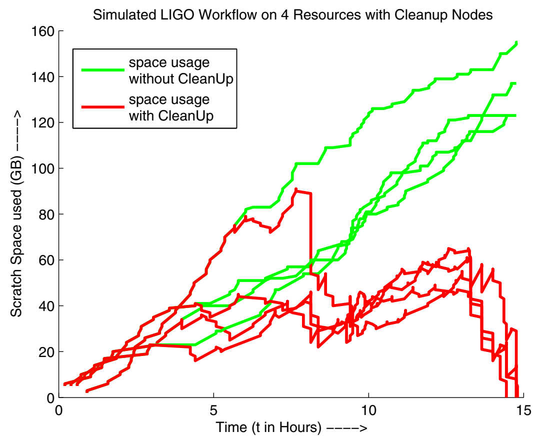

Figure 5. Cleanup Results for the Simulated LIGO Workflow on 4 Resources.

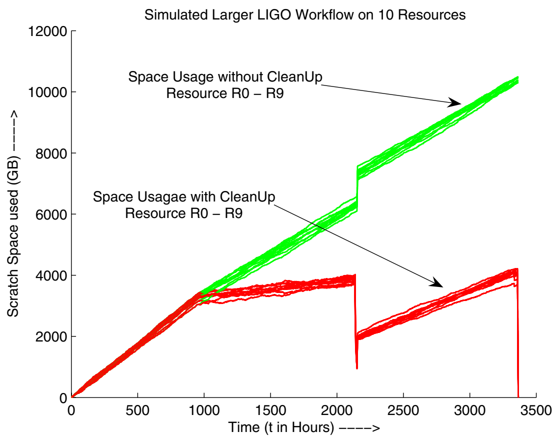

Figure 6. Cleanup Results for the Larger Simulated LIGO workflow.

resource id	no cleanup (GB)	with cleanup (GB)	% improvement
0	137	58	57%
1	123	65	47%
2	155	91	41%
3	123	61	50%

Table 1. Maximum amount of storage used at the resources without and with cleanup

the lifetime of the workflow is about 50 percent less when the cleanup nodes are added to the workflow.

We also simulated the execution of a much larger LIGO workflow containing 38954 tasks. The simulated workflow is similar in structure to the one shown in Figure 4, with the same number of levels but with many more tasks at each level.

The tasks in the workflow were randomly mapped to 10 homogeneous execution resources. Figure 6 shows the result of simulating the execution of the workflow on 10 resources both with and without the addition of cleanup jobs. Due to the large number of tasks in the workflow and the random assignment of tasks to resources, the amount of space used at each resource is approximately the same. Adding the cleanup nodes, the maximum storage used at the resources is approximately 50 percent less than the storage used without the cleanup nodes.

It should be noted here that while the algorithm described in Figure 3 is able to significantly reduce the amount of storage used for the two workflows, the number of cleanup jobs can become greater than the number of tasks in the executable workflow, particularly if the workflow is being executed across multiple resources. For example, our cleanup algorithm generated 544 cleanup tasks for the small workflow with 166 compute tasks. In general, the number of cleanup tasks would be O(the number of files used in workflow times the number of resources the workflow is mapped to). The cleanup tasks are not compute intensive and hence are not likely to affect the runtime of the workflow significantly. However, the sheer number of tasks may cause performance degradation in the workflow execution engine. We have also implemented a heuristic for reducing the number of cleanup tasks. The rationale is to use a single cleanup for removing multiple files instead of using one cleanup job for each file. We were able to reduce the number of cleanup nodes by a factor of 5-6 on synthetic workflows as well as on the simulated LIGO workflows and still obtain the same maximum space usage. In particular, we were able to reduce the number of cleanup jobs for the small workflow to 80 aggregate cleanup jobs from the 544 earlier.

4. Algorithm for Storage-Aware Workflow Scheduling

The removal of data files when they are no longer needed is only one step towards the efficient mapping and execution of workflows since it minimizes their overall storage requirements. However, for efficient execution, one also needs to guarantee the usage of resources with ample disk space for the tasks of the workflow and to consider mapping onto those resources in a way that minimizes the overall execution time of the workflow. For the latter, the possible benefits that might result from replication of data files need

```
Input: Executable Workflow, r=1..R (list of compute resources), information about disk usage
Output: Mapping of Workflow tasks onto resources

While (there are unscheduled tasks) do
  Select the first ready task, i.
  For every resource r=1..R
    Compute expected disk usage of task i on resource r,
    EDU(i) = Input(i) + Output(i).
    Check the maximum disk space of resource r, DS(r), and the current disk space DU(r).
    if ( EDU(i) + DU(r) ) $\geq$ DS(r) )
      resource r must not be considered for the allocation of task i.
    else
      compute earliest finish time of task i on resource r, EFT(i,r).
  End For
  if (no resources available) do
    mark task and repeat the above for the next ready unmarked task.
    if all ready tasks are marked then halt algorithm // failure
  else
    Assign task i to the resource r that minimizes EFT of task i.
    For (each parent task p of task i) do
      Send task p a message to the resource where task p has been allocated, say resource m.
      Request p to transfer all files required by task i to resource s.
      Proceed to cleanup of any unnecessary files required from resource m.
      Update the current disk usage of resource m, DU(m).
    End For
End While
```

Figure 7. The Storage-Aware Workflow Scheduling Algorithm.

to be weighed as well since these benefits will be obtained at the expense of additional disk space. This section describes an algorithm which aims to schedule workflows to storage-constrained resources and at the same time to minimize the overall workflow execution time. The key idea, when allocating tasks, is to consider first disk space availability of resources and then prioritize resources depending on performance (task execution on that resource). The input of the algorithm is a workflow, the execution time estimates for each compute task in the workflow, and the size of input and output files each compute task may require and produce. In addition, there is a set of available (compute) resources, each with its own disk space. The execution time estimates and input and output file sizes can be obtained using historical information from the previous runs of the workflow.

The algorithm consists of three phases: (1) identification of all resources that can accommodate the data files needed for a task; (2) allocation of the task to the resource which can achieve the earliest finish time for the task; and (3) cleanup of any unnecessary data files as indicated by any cleanup jobs inserted using the algorithm in the previous section.

In the first phase, the expected disk usage (EDU) of a task, i, which is ready for execution (ready means that its parents have completed their execution) is calculated. The value of EDU is the sum of the size of the input files of the task, Input(i), and the size of the output files the task may generate, Output(i). If the task is allocated to the same resource as all its parent tasks, then the value of Input(i) is set to zero since the disk space for its input data has already been accounted for under output file sizes of its parent tasks (Output(k), where k is a parent of task i). The algorithm then decides if task i can be allocated to a resource by considering the task's expected disk space usage, the current disk space usage and the total disk space this resource has. If the allocation of task i does not exceed the maximal disk space of a given resource, this resource is considered to be a candidate for the next phase. This process is repeated for each available resource. If no resources at all satisfy the space requirements of any ready task, the algorithm halts and results in a failure for allocation.

If there are resources which can accommodate the space requirements of the task being considered, the algorithm proceeds to the second phase. In this phase, the expected finish time of the job (corresponding to this task) on each of these resources is considered. The finish time is computed as the sum of the time to transfer any data from parents and the time to execute the job on the resource. The job is then allocated to the resource which results in the smallest finish time. It is noted here that considering the resource that gives the smallest finish time implicitly evaluates the benefits of data replication. This is because the time to transfer any data from the parent resources is also considered when determining the resource that gives the smallest finish time. Finally, once an allocation decision has been made and all the files required by a job have been sent to the resource that executes this job, the files (if they are no longer needed) can be removed from the parent job's resource.

An outline of the algorithm is given in Figure 7. Its complexity is $O(e + (n \times m))$, where e is the number of edges, n is the number of compute tasks and m is the number of resources available for execution. In practice, however, the running time is insignificant, since there are only low-cost operations involved in the algorithm.

5. Evaluation and Discussion

This section evaluates the benefits of the storage-aware workflow scheduling algorithm against two other approaches available for workflow scheduling, which either do not take into account individual resource characteristics or do not perform any cleanup. The aim is to examine the rate of failure and the overall performance of the proposed algorithm with different combinations of network capacities, disk storage, and the number of available sites.

Same as before, we used simulation and the workflow shown in Figure 4, containing 166 compute tasks. The total file size required by the workflow (without cleanup) was approximately 118 GBytes. We assume that the workflow is mapped onto homogeneous resources, which are connected by a network which has the same speed between any two resources. We considered a number of experiments, where we chose different values for the number of compute resources available, the network speed between them, and the disk space available at each of the resources in order to observe the behavior of different scheduling algorithms. Thus, we considered a number of: 3, 6, or 9 resources available for execution, with network speeds between these resources of: 100MB/sec, 10MB/sec or 1MB/sec, and the maximum disk space available in each resource at the start of the execution of the workflow of: 10, 15, 20, 25, or 30 GBytes. The maximum disk space available was the same for each resource in all the runs. Therefore, in total, we considered 45 ($= 3 \times 3 \times 5$) different execution environments.

The results are shown in Table 2. Our proposed storage-aware scheduling algorithm has been implemented and is denoted in the table as 'alg1'. The other two algorithms used in the evaluation are denoted as 'alg2' and 'alg3'. Algorithm 'alg2' considers data cleanup (implementing the algorithm in Section 3 of this paper), but does not take into account the space available at each individual resource when allocating tasks onto resources, nor the execution time on the resource; it simply selects resources randomly to assign tasks. This may lead to the assignment of a task to a resource which does not have enough storage for the files needed by a task. On the other hand, 'alg3' considers resource storage availabilities when allocating jobs and assigns the job to the best machine, but the algorithm does not perform any cleanup (for data files that are no longer needed). All three algorithms require that all the input data files of each task are available on the resource that this task was allocated for the task to start execution.

Network Speed (MB/sec)	Disk (GB per resource)	number of resources	alg1	alg2	alg3
100	15-30	9	1444	1739	1444
100	10	9	1444	1739	Fail
10	15-30	9	2404	4395	2404
10	10	9	2404	4395	Fail
1	15-30	9	12002	30956	12002
1	10	9	12002	30956	Fail
100	20-30	6	2154	2548	2154
100	15	6	2154	2548	Fail
100	10	6	2154	Fail	Fail
10	20-30	6	3584	6308	3584
10	15	6	3584	6308	Fail
10	10	6	3584	Fail	Fail
1	20-30	6	17889	43910	17889
1	15	6	17889	43910	Fail
1	10	6	17889	Fail	Fail
100	25-30	3	4281	9957	Fail
100	20	3	4281	Fail	Fail
100	10-15	3	Fail	Fail	Fail
10	30	3	6850	12569	Fail
10	20-25	3	6850	Fail	Fail
10	10-15	3	Fail	Fail	Fail
1	30	3	32532	87738	Fail
1	20-25	3	32532	Fail	Fail
1	10-15	3	Fail	Fail	Fail

Table 2. Simulated execution time (in sec) for the LIGO workflow in Figure 4, for different environment settings and different scheduling algorithms.

The results in Table 2 show the execution time of the workflow for each different setting and algorithm. The entry 'Fail' means that the corresponding algorithm could not finish the workflow allocation due to space constraints at some stage during the execution. Since disk capacity primarily affects the ability to run the workflow, rather than its overall execution time, the results are grouped when the outcome does not differ. So, for example, the first row of the table indicates that the execution time of each algorithm remains the same for disk capacity per resource of 15-30 GBytes.

It can be seen clearly that our proposed algorithm, ‘alg1’, can give solutions in many cases that the other two algorithms fail. The makespan of these solutions is always better than the makespan of ‘alg2’. The difference is more profound with slower network speeds or a smaller number of resources. For example, with 1MB/sec network speed, the makespan of ‘alg1’ can be three times faster than the makespan using ‘alg2’.

The ‘alg3’ algorithm failed to provide solutions in many cases in the experiments. Especially for small number of resources and small disk space, ‘alg3’ was unable to finish the allocation regardless of the network speed. In the case of 6 resources, it can be seen that ‘alg1’ can run in settings with half the available disk space that ‘alg3’ needs (i.e., 10 GB/resource as opposed to 20 GB/resource), a result which is in line with our findings (see Table 1) that the cleanup process reduces the disk requirements of the simulated LIGO workflow by about half.

The results clearly demonstrate that it is not sufficient to consider only data relocation or data locality when running data-intensive workflows in space-constrained environments.

6. Related Work

With Directed Acyclic Graphs (DAGs) being a convenient model to represent workflows, the vast amount of literature on DAG scheduling is of relevance to the problem of workflow scheduling [21]. In recent years, there has been a revival of interest in the context of problems especially motivated by scientific workflow execution and heterogeneous environments [22, 23, 24, 25, 26, 27, 28, 29]. In the majority of these works the aim is to minimize the workflow execution time. No work has taken into account the available data storage when selecting resources, which has proved to be a critical factor when executing data-intensive workflows.

The most interesting work in the context of this paper, which considers data placement, has been presented in [30, 31]. Their proposed scheduling and replication algorithm keeps track of the popularity of datasets and replicates those datasets to different sites. However, the data replication approach does not work well in a storage-constrained environment as it may increase the demand of data storage and may lead to heavy storage requirements for individual resources. To draw an analogy, ‘alg3’ in Section 5 is a simple version of a data replication approach; however, it did not complete the execution in many cases because there was not enough space for data storage.

7. Conclusions

We examined the problem of mapping scientific workflows onto distributed resources where the amount of disk space at the resources is limited. We presented a two-prong approach where we minimized the disk space footprint of the workflow by removing data as soon as it is no longer needed and where we scheduled the workflow tasks by first taking into account the data requirements of the workflow and the data space availability at the resources. Using our approach we were able to decrease the space needed by a workflow used by gravitational-wave physicists by as much as 57% as compared to the un-optimized version of the workflow. Additionally, we presented an algorithm for scheduling the workflow that demonstrated that taking into account space constraints when scheduling workflow tasks onto resources with limited disk space yields not only feasible solutions, where other algorithms may fail, but also does not compromise the overall workflow performance.

In the future, we plan to study disk space-aware algorithms further, in particular examining the tradeoffs between space and time optimizations. We also intend to consider optimizations for scheduling the workflow onto resources that can evaluate the properties of the workflow as a whole in order to make more informed decisions about task allocation. While the results presented in this paper were obtained using simulations, we also plan to do experiments on real operational Grid infrastructure such as TeraGrid [3] in order to demonstrate the efficacy of the presented algorithms.

Acknowledgement: This work was supported by the National Science Foundation under the grant CNS 0615412. R. Sakellariou and H. Zhao would like to acknowledge partial support from the EU-funded CoreGrid Network of Excellence (grant FP6-004265) and the UK EPSRC grant GR/S67654/01. The authors thank Duncan Brown for his contributions to the LIGO workflow used to model simulated workflows. The authors also thank the Open Science Grid for resources used to motivate the work presented. The work of K. Blackburn, D. Meyers. and M. Samidi was supported by the National Science Foundation under awards PHY-0107417 and PHY-0326281. The LIGO Observatories were constructed by the California Institute of Technology and Massachusetts Institute of Technology with funding from the National Science Foundation under cooperative agreement PHY-9210038. The LIGO Laboratory operates under cooperative agreement PHY-0107417. This paper has been assigned LIGO Document Number LIGO-P070003-00-Z. This research was done using resources provided by the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy’s Office of Science.

References

[1] E. Deelman and Y. Gil. Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges. In *Proceedings of Workflows in e-Science*, Amsterdam, 2006.

[2] I. Taylor, E. Deelman, D. Gannon, and M. Shields (Eds). *Workflows in e-Science*, Springer-Verlag, 2006.

[3] TeraGrid. http://www.teragrid.org/

[4] Open Science Grid. http://www.opensciencegrid.org/

[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] Pegasus. http://pegasus.isi.edu/

[7] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow Management in GriPhyN. In *Grid Resource Management*, J. Nabrzyski, J. Schopf, and J. Weglarz, Eds.: Kluwer, 2003.

[8] D. S. Katz, N. Anagnostou, G. B. Berriman, E. Deelman, J. Good, J. C. Jacob, C. Kesselman, A. Laity, T. A. Prince, G. Singh, M.-H. Su, and R. Williams. Astronomical Image Mosaicking on a Grid: Initial Experiences. In *Engineering the Grid - Status and Perspective*, B. D. Martino, J. Dongarra, A. Hoisie, L. Yang, and H. Zima, Eds.: ASP, 2005.

[9] E. Deelman, K. Blackburn, P. Ehrens, C. Kesselman, S. Koranda, A. Lazzarini, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, and R. Williams. GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists. *Proceedings of 11th Intl Symposium on High Performance Distributed Computing*, 2002.

[10] Duncan A. Brown et al. A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis. In *Workflows for e-Science: Scientific Workflows for Grids* Springer-Verlag, 2006.

[11] P. Maechling, H. Chalupsky, M. Dougherty, E. Deelman, Y. Gil, S. Gullapalli, V. Gupta, C. Kesselman, J. Kim, G. Mehta, B. Mendenhall, T. A. Russ, G. Singh, M. Spraragen, G. Staples, and K. Vahi. Simplifying construction of complex workflows for non-expert users of the Southern California Earthquake Center Community Modeling Environment. *SIGMOD Record*, vol. 34, pp. 24-30, 2005.

[12] A. Lathers, M.-H. Su, A. Kulungowski, A. W. Lin, G. Mehta, S. T. Peltier, Ewa Deelman, and M. H. Ellisman. Enabling Parallel Scientific Applications with Workflow Tools. *Proceedings of Challenges of Large Applications in Distributed Environments (CLADE)*, Paris, 2006.

[13] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, vol. 5, pp. 237-246, 2002.

[14] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, vol. 11, pp. 115-128, 1997.

[15] M. Litzkow, M. Livny, and M. Mukta Condor - A Hunter of Idle Workstations In *Proceedings of the 8th International Conference on Distributed Computing Systems*, 1988, 104-111.

[16] B. C. Barish and R. Weiss. LIGO and the Detection of Gravitational Waves. *Physics Today*, vol. 52, pp. 44, 1999.

[17] B. Abbott et al. Search for gravitational waves from primordial black hole binary coalescences in the galactic halo. In *Physical Review D*, Vol 72(08), 2005.

[18] B. Abbott et al. Search for gravitational waves from binary black hole inspirals in LIGO data. In *Physical Review D*, Vol 73(06), 2006.

[19] R. Buyya and M. Murshed. Gridsim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1175-1220, 2002.

[20] B. Abbott et al. Search for gravitational waves from binary inspirals in S3 and S4 LIGO data. *In preparation*, 2007.

[21] Y.-K. Kwok and I. Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs. *ACM Computing Surveys*, 31(4):406-471, 1999.

[22] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Resource Allocation Strategies for Workflows in Grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2005.

[23] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu and L. Johnsson. Scheduling Strategies for Mapping Application Workflows onto the Grid. In *IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*, 2005.

[24] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos. Scheduling Workflows with Budget Constraints. In S. Gorlatch, M. Danelutto (Eds.), *Integrated Research in Grid Computing*, CoreGrid series, Springer-Verlag, to appear.

[25] R. Sakellariou and H. Zhao. A Low-Cost Rescheduling Policy for efficient mapping of Workflows on Grid Systems. *Scientific Programming*, vol. 12, no. 4, pp. 253-262, December 2004.

[26] M. Wieczorek, R. Prodan and T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. *SIGMOD Record*, volume 34(3), September 2005.

[27] J. Yu and R. Buyya. A Budget Constraint Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms. *Proceedings of the Workshop on Workflows in Support of Large-Scale Science (WORKS06)*, Paris, France, 2006.

[28] H. Zhao and R. Sakellariou. Advance Reservation Policies for Workflows. *12th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Saint-Malo, France, June 2006.

[29] H. Zhao and R. Sakellariou. Scheduling Multiple DAGs onto Heterogeneous Systems. *Proceedings of the 15th Heterogeneous Computing Workshop (HCW)*, Rhodes, Greece, 2006.

[30] K. Ranganathan and I. Foster. Design and Evaluation of Dynamic Replication Strategies for a High-Performance Data Grid. *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics*, Beijing, September 2001.

[31] K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. *International Symposium for High Performance Distributed Computing (HPDC)*, Edinburgh, July 2002.