Optimizing Workflow Data Footprint

Gurmeet Singh[1], Karan Vahi[1], Arun Ramakrishnan[1], Gaurang Mehta[1], Ewa Deelman[1], Henan Zhao[2], Rizos Sakellariou[2], Kent Blackburn[3], Duncan Brown[3,4], Stephen Fairhurst[3,5], David Meyers[3,6], G. Bruce Berriman[7], John Good[7], Daniel S. Katz[8]

[1] USC Information Sciences Institute, 4676 Admiralty Way, Marina Del Rey, CA 90292, USA.
`{gurmeet, vahi, arun, gmehta, deelman}@isi.edu`
[2] School of Computer Science, University of Manchester, Manchester M13 9PL, UK.
`{hzhao,rizos}@cs.man.ac.uk`
[3] LIGO Laboratory, California Institute of Technology, MS 18-34, Pasadena, CA 91125, USA
`{kent, dbrown, fairhurst_s, dmeyers}@ligo.caltech.edu`
[4] Theoretical Astrophysics, California Institute of Technology, MS 130-33, Pasadena, CA 91125
[5] Physics Department, University of Wisconsin--Milwaukee, Milwaukee, WI 53202
[6] Northrop Grumman Information Technology, 320 North Halstead Suite 170, Pasadena, CA 91107, USA
[7] Infrared Processing and Analysis Center, California Institute of Technology, CA 91125
`{gbb,jcg@ipac.caltech.edu}`
[8] Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803
`{dsk@cct.lsu.edu}`

Abstract

In this paper we examine the issue of optimizing disk usage and scheduling large-scale scientific workflows onto distributed resources where the workflows are data-intensive, requiring large amounts of data storage, and 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 needed and we demonstrate that workflows may have to be restructured to reduce the overall data footprint of the workflow. We show the results of our data management and workflow restructuring solutions using a Laser Interferometer Gravitational-Wave Observatory (LIGO) application and an astronomy application, Montage, running on a large-scale production grid—the Open Science Grid. We show that although reducing the data footprint of Montage by 48% can be achieved with dynamic data cleanup techniques, LIGO Scientific Collaboration workflows require additional restructuring to achieve a 56% reduction in data space usage. We also examine the cost of the workflow restructuring in terms of the application's runtime.

1. Introduction

Today, scientific analyses are frequently composed of several application components, each often designed and tuned by a different researcher. Scientific workflows [6] 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 [2] 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 [20, 22, 23] which stands for Planning for Execution in Grids, is a workflow mapping engine that is used day-to-day to map complex, large-scale scientific workflows with thousands of tasks processing terabytes of data onto the Grid. Some examples include applications in physics [19], astronomy [13, 14], gravitational-wave science [18], as well as earthquake science [21, 30], neuroscience [28], and others [32]. 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 [1] 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 [25] or Condor [29]).

As a 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 registries, 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 the issue of minimizing the amount of storage space that a workflow requires for execution, also called the *workflow data footprint*. In some cases, we also demonstrate that workflow restructuring is needed to obtain a reduction in the workflow data footprint.

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) [10] Scientific Collaboration (LSC) application, which requires large amounts of storage space and targets the OSG as its execution environment and an astronomy application called Montage [15]. These applications exhibit behaviors found in many scientific workflows used today. Section 3 describes two data cleanup algorithms for reducing the amount of space required by a workflow by dynamically removing files when they are no longer required while the workflow is executing. Section 4 shows the simulation and experimental results of applying the cleanup algorithm to the LSC and Montage application and discusses the reasons for limited improvement in the case of the LSC workflow. In

Section 5 we examine workflow restructuring as a means of reducing the data footprint of the LSC workflow and present results for the restructured workflow. Section 6 follows up with a discussion on the issues of restructuring of the LIGO workflow from the science perspective. Finally, we give an overview of related work and include concluding remarks.

2. Motivation

Many applications today are structured as workflows. Some examples of such applications that are being routinely used in large-scale collaborations are the LSC's binary inspiral search [17] and the Montage application. Here we describe them both and focus on their computational requirements.

2.1. Laser-Interferometer Gravitational-Wave Observatory

LIGO 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 phenomenon which is expected to be a 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 [8, 9]. Gravitational waves interact extremely weakly with matter, and the measurable effects produced in terrestrial instruments by their passage will be miniscule. In order to increase the probability of detection, a large amount of data needs to be acquired and analyzed which contains the strain signal that measures the passage of gravitational waves. LSC 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 increase 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. A scientifically meaningful run of the binary inspiral analysis requires a minimum of 221 GBytes of gravitational-wave data and approximately 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 a High Performance Computing or High Throughput Computing resource, with, on average, 258 GB of shared scratch disk space. The shared scratch disk space is used by approximately 20 VOs within the OSG. The LIGO VO can not reserve space on these shared resources. Thus reducing the disk space requirements of the LSC application minimizes the risk of failure due to lack of storage space.

2.2. Montage

Montage [3, 14] is an application that constructs custom science-grade astronomical image mosaics on demand based on several existing images. The inputs to the workflow include the input images in standard FITS format (a file format used throughout the astronomy community) taken from image archives such as 2MASS [5] and a "template header file" that specifies the mosaic to be constructed. The input images are first reprojected to the coordinate space of the output mosaic, the reprojected images are then background rectified and finally coadded to create the final output mosaic. Figure 1 shows the structure of a small Montage workflow with tasks represented by the vertices and edges representing the data dependencies between the tasks. In order to facilitate subsequent discussion, we assign a level to each task in the workflow. The tasks with no parents are defined to be at level one and the level of any other task in the workflow is the maximum level of any of its parents plus one. The levels of the tasks in the workflow in Figure 1 are shown by the numbers in the vertices representing these tasks. The figure only shows the graph of the resource-independent abstract workflow. The executable workflow will additionally contain data transfer, registration nodes, and optionally data cleanup nodes.

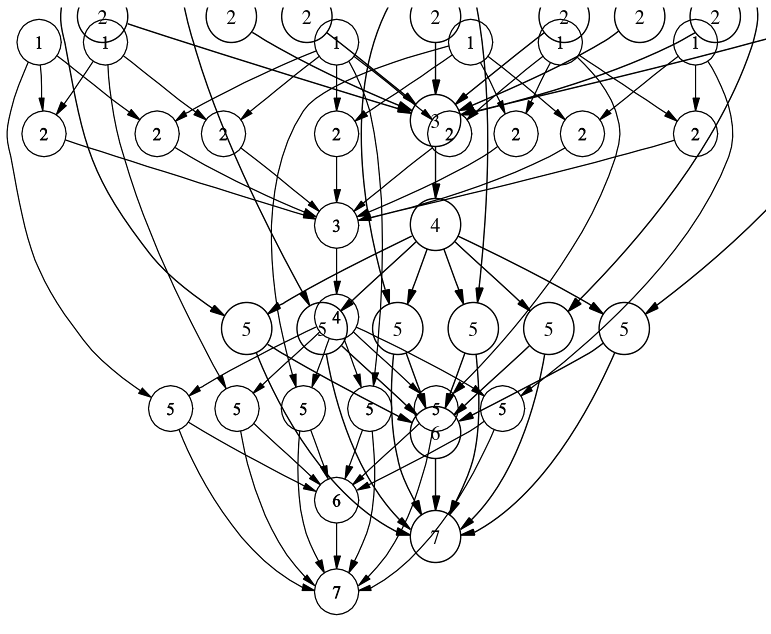


Figure 1: A small Montage workflow.

Montage workflows typically contain a large number of tasks that process large amounts of data. Considering a workflow for creating a representative 2 degree square mosaic of 2MASS images centered around the celestial object M17, Table 1 gives a description of the number of tasks at each level of the workflow and the average amount of input data required and output data produced by each task at the level. All the tasks at a particular level are invocations of the same program, such as *mProject* for level one. The actual workflow has 4 more levels in addition to the 7 shown in Table 1 and these remaining levels consists of tasks for creating a visualizable image of the mosaic.

Table 1: Characteristics of the Tasks/Transformations in the 2 Degree Square Montage Workflow.

Level	Transformation Name	Description	No. of jobs at the level	Approximate Size of data input/output for each job (MB)
1	mProject	Reprojects a single image to the image parameters and footprints defined in a header file.	152	4.2 / 8.1
2	mDiffFit	Finds the difference between two images and fits a plane to that difference image	410	16.3 / 0.6
3	mConcatFit	Does a simple concatenation of the plane fit parameters from multiple mDiffFit jobs into a single file	1	0.12 / 0.08
4	mBgModel	Models the sky background using the plane fit parameters from mDiffFit and computes planar corrections for the input images that will rectify the background across the entire mosaic	1	0.1 / 0.007
5	mBackground	Applies the planar correction to a single image	410	8.1 / 8.1
6	mImgtbl	Extracts the FITS header geometry information from a set of files and stores it in an image metadata table	4	407.6 / 0.01
7	mAdd	Co-adds a set of reprojected images to produce a mosaic as specified in a template header file	4	407.6 / 272.4

As the sizes of the mosaics increase, the data processed and generated in the workflow increase. In the near future Montage will be provided as a service to the astronomy community. It is expected that there will be many simultaneous mosaics being generated based on the user requests, and thus it will become important to minimize the data footprint of each workflow.

3. Approach

The first algorithm (Figure 3) 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 computational resource to make room for the subsequent computations. Since a data file can be potentially replicated on multiple resources (in case the compute tasks are mapped to multiple resources) the decisions to add cleanup jobs are made on a per resource basis. The algorithm is applied after the executable workflow has been created but before the workflow is executed.

In order to illustrate the working of the algorithm, Figure 2(a) 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}. The cleanup nodes added to this workflow using the algorithm are shown in Figure 2(b). The cleanup job for removing file f on resource r is denoted as C_{fr}.

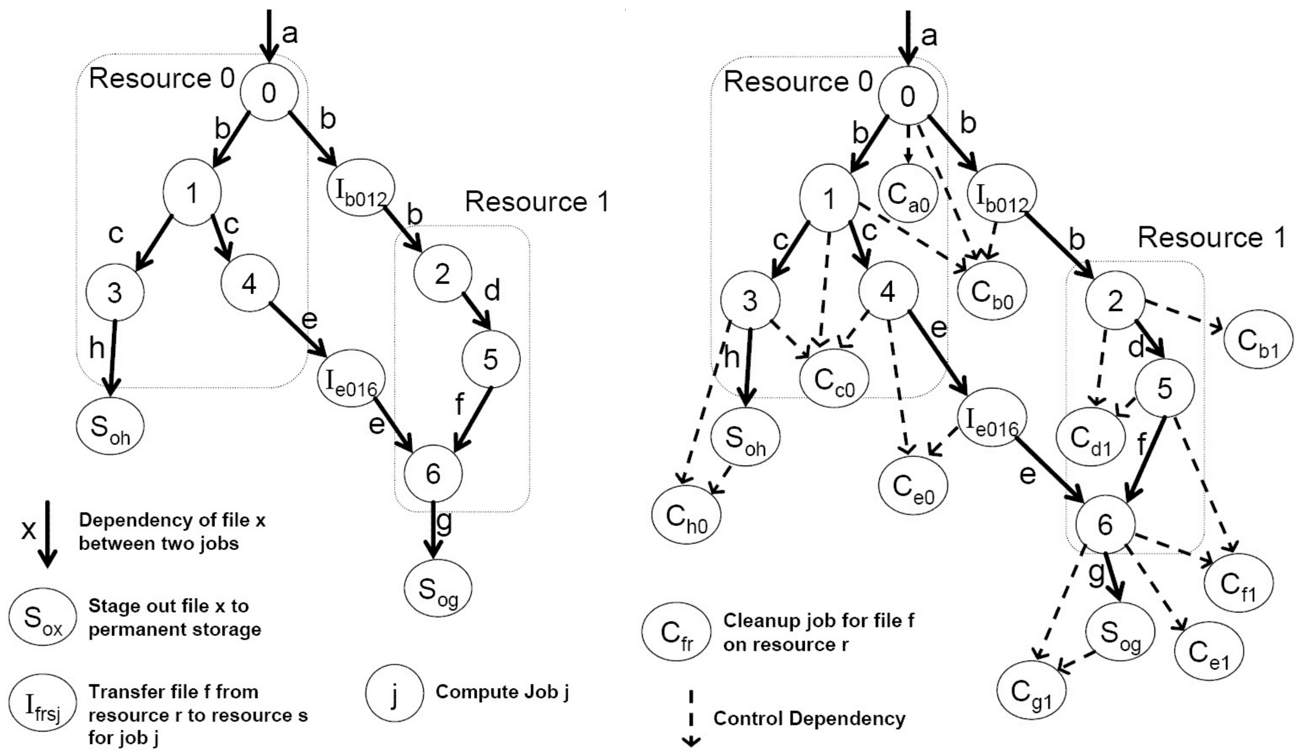

Resource 0
Resource 1
Dependency of file x between two jobs
Stage out file x to permanent storage
Transfer file f from resource r to resource s for job j
Compute Job j
Cleanup job for file f on resource r
Control Dependency

Figure 2: (a)The Original Executable Workflow Mapped by Pegasus to Two resources. (b) The Workflow with Dynamic Cleanup Nodes Added.

For each task in the subgraph, a list of files either required or produced by the task is constructed. For example, the 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 the parent of the cleanup job. Thus, a cleanup job, C_{c0}, which will remove file c on resource 0 is created and task 1 is made the 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 a 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, 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 running time of the algorithm 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 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.

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

Method AddCleanUpJobs
For every resource r = 1..R
   Let $G_r=(V_r,E_r)$ be the subgraph induced by the tasks mapped to resource r
   For every job j in $V_r$
      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 $C_{fs}$ 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 $C_{fr}$ for file f for resource r if it does not already exist
         add job j as parent of the cleanUpJob $C_{fr}$
         If f is being staged out to final storage, add $C_{fr}$ as child of the stage-out job.
      End For
   End For
End For
End Method AddCleanUpJobs
```

Figure 3: Data Cleanup Algorithm, With One Cleanup Job Per Data File. The Running Time of the Algorithm O(m), where m is the no of edges in the abstract workflow.

In our initial work [34], we were able to achieve as much as a 57% data footprint improvement for LSC-like workflows in a simulated environment. Our next step was to evaluate the performance of the algorithm on a real application and a real grid. However, in order to make the algorithm viable, we needed to improve the algorithm in terms of the number of cleanup jobs it added to the workflow and in terms of the number of dependencies it introduced. The issue is that increasing the number of tasks and dependencies in a workflow increases the amount of time the workflow engine spends managing the workflow and introduces additional overheads to the overall workflow execution because of the inherent overheads in scheduling jobs onto distributed systems (job handling at the submission site, network latencies, queue time at the remote resource, etc.). Because of these overheads, we decided to design an improved algorithm that will reduce the number of cleanup tasks at the possible cost of the workflow footprint.

The algorithm in Figure 5 creates at most one cleanup node per computational workflow task. The algorithms works by creating a cleanup task per job in the workflow using the *AddCleanUpJobs* method resulting in the workflow shown in Figure 4 when applied to the workflow in Figure 2(a). The method *ReduceDependencies* is used within *AddCleanUpJobs* to eliminate unnecessary dependencies between the cleanup and other jobs. Also the stage-out and stage-in jobs are considered the same way as compute jobs and are logically assigned to the source resource from where they are transferring data. Thus the stage-in jobs I_{b012}, I_{e016} and stage-out job S_{0h} are treated as any other compute job mapped to resource 0 by the AddCleanUpJobs method.

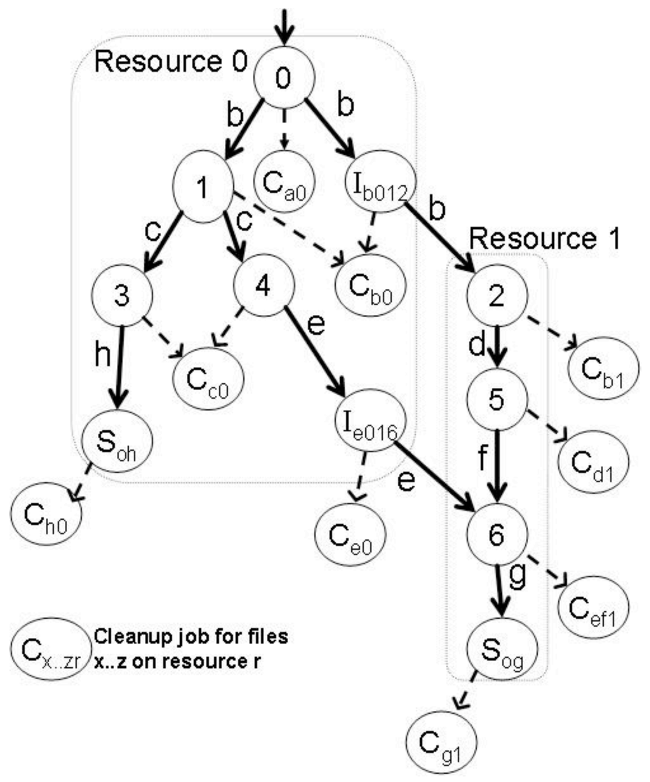


Figure 4: Resulting workflow after applying the improved cleanup algorithm.

The algorithm works as follows. The *AddCleanUpJobs* method first assigns a level to each task in the workflow. The rest of the processing is done on per resource basis. First it creates a priority queue of leaf tasks mapped to that resource with the priorities being their level values. For resource 0, the leaf tasks are tasks S_{0h}, I_{e016} with each having a level value of 4. The task with the highest priority in the queue is considered first. For example, assume that the task S_{0h} is first considered (other equivalent choice is I_{e016}, since both have the same level). Since the only file required by S_{0h} is h, a cleanup job C_{h0} is created to remove h and C_{h0} is made the child of S_{0h}. The parents of S_{0h}, namely 3 is then added to the priority queue. Next I_{e016} is considered and C_{e0} is created and made the child of I_{e016}. Its parent, 4 is also added to the queue. Thereafter either of task 3 or 4 may be considered. Assuming task 3 is considered next, it requires files c and h. However, since the cleanup job C_{h0} already exists to remove file h, task 3 is made the parent of C_{h0} and a new task C_{c0} is created to remove the file c. Task 4 is next considered and it requires files c and e. Since cleanup jobs already exist to remove these files, task 4 is simply made the parent of cleanup jobs C_{c0} and C_{e0} and no new cleanup jobs are created. The rest of the tasks are similarly processed. The use of a priority queue results in a bottom-up traversal of the subgraph mapped to resource 0. Once the subgraph has been traversed, the method *ReduceDependencies* is then used to eliminate unnecessary dependencies from the workflow such as between the task 3 and clean job C_{h0}.

This is because task S_{0h} is a child of task 3 and a parent of cleanup job C_{h0}. Thus a direct dependency between task 3 and C_{h0} is not required.

```
Method AddCleanUpJobs
    Assign 1 as level for the root jobs
    For every job j in the workflow in a topologically sorted order
        Level( j ) = Max( level( Parent( j ) ) ) + 1
    EndFor
    For every Resource i = 1 .. R
        Let $G_i = (V_i , E_i)$ be the sub graph induced by the jobs mapped to Resource i .
        Let $Leaf_i$ be the subset of $V_i$ that are the leaf nodes of $G_i$
        Create a empty max priority queue, PQ
        For each job J in $Leaf_i$
            Insert J in PQ with its level being its priority
        EndFor
        While PQ is not empty , extract a job j from PQ with the highest level.
            Create a cleanup job new_cleanup
            For every file f used/produced by j
                If f is not already set to be cleaned up by another job, set new_cleanup to clean f.
                Else add j as the parent of the cleanup job which was already set to clean f
                EndIf.
            EndFor
            If new_cleanup cleans up at least one file, add it as the child of job j
            Add all parents of job j in $G_i$ to PQ that don't already exist in PQ
        End While
        For every CleanupJob C  created for the resource i  in the above loop.
            Reduce_Dependencies ( C , Gi ).
        End For
    EndFor
End Method AddCleanUpJobs

Method ReduceDependencies ( CleanupJob C , Graph Gi  )
    Duplicates is an empty set of jobs.
    Mark all jobs in Gi as unvisited
    ListParents be the list of jobs that are direct parents of  C.
    For every Job J that is in ListParents
        If  J is not in Duplicates
            Initialize queue BFSq with all the parents of J.
            While BFSq is not empty
                Pop job uJ from BFSq.
                If uJ is not marked as visited
                    Mark uJ as visited.
                    If uJ is a member of ListParents
                        Add uJ to Duplicates
                    EndIf
                    Add all unvisited parents of uJ to BFSq.
                End If
            End While
        End If
    End For
    Remove all edges between CleanupJob C and the jobs in Duplicates.
End Method ReduceDependencies
```

Figure 5: Dynamic Data Cleanup Algorithm, With One Cleanup Job Per Computational Task.

The reason for the bottom-up traversal order of tasks in *AddCleanUpJobs* method is that we want to associate the cleanup job for a file with the last task in the subgraph that uses the file and this is best done in a bottom-up fashion. The complexity of this algorithm is *O(e * (e+n)),* where e is the number of edges in the workflow with cleanup jobs and n is the number of jobs .

4. Experiments

4.1. Setup

In this paper, we use two methods for evaluating our approach: simulation and runs of the real applications on the grid. For all the experiments, the workflows are specified in an abstract format and then mapped onto the resources by Pegasus in the case of real execution. In the case of the simulator these resources are simulated and in the case of the grid execution, we use the resources of the Open Science Grid.

Simulation

The simulations are performed using a Java-based grid simulator called GridSim [33]. This simulator can be used to simulate any number of grid resources and users. We added attributes to the task model in GridSim to explicitly model clean up jobs. In our experiments, each simulated resource is a cluster of homogeneous processors with space sharing and a First Come First Serve (FCFS) scheduling policy. The processing speed of the resources and the bandwidth between the users and resources can be configured and contention for network resources is ignored. For the simulation experiments described in this paper, each resource was a cluster of ten homogeneous processors.

Grid Execution

For our grid execution, we use the Pegasus [20, 22, 23] workflow mapping system to map a high-level workflow description onto the available resources and use the Condor DAGMan [1] workflow execution engine to execute the workflow on the mapped resources. Pegasus locates the available resources and creates an executable workflow where the executable tasks are assigned to resources, where there are data transfer nodes to stage data in and out of the computations, data registration nodes to register the intermediate data products into registries, and in cases of workflows with dynamic cleanup nodes, these will be included as well. The executable workflow is given to DAGMan for execution. DAGMan follows the dependencies defined in the workflow and releases the nodes that are ready to run to a local Condor queue. Tasks destined for the grid are sent to the grid resources using Condor-G [26]. The final scheduling of these tasks is done by the resource managers of the remote grid resources. As part of the execution, the data is generated along with its associated metadata and any provenance information that is collected.

4.2. Comparison of Cleanup Algorithms

In our first set of experiments, we compared the behavior of the two cleanup algorithms described in Section 3. We performed the evaluation via simulation and compared the number of cleanup jobs created by each algorithm and the number of dependencies added to the workflow. We performed the experiments using both the LSC and Montage workflows containing 164 and 731 compute tasks respectively. Algorithm I represents the algorithm where one cleanup job is created for every file, and Algorithm II represents the algorithm where at most one cleanup job per compute job in the workflow. Results in Table 2 show that Algorithm II obtains a data footprint similar to that of Algorithm I while also achieving a significant reduction in the number of cleanup tasks and dependencies. There is approximately a 40% reduction in the number of tasks and almost 30% reduction in the number of dependencies. All subsequent experiments described in this paper were done using Algorithm II and this algorithm is currently implemented in Pegasus.

Table 2: Comparison via Simulation of the Data Cleanup Algorithms, Showing the Reduction in the Number of Cleanup Tasks and the Number of Dependencies.

LSC workflow	Max Space Used (MBs)	No of CleanUp Jobs	No of dependencies
Algorithm I	1027.13	237	840
Algorithm II	1028.23	96	238
2-degree MONTAGE	**Max Space Used (MBs)**	**No of CleanUp Jobs**	**No of dependencies**
Algorithm I	2405.71	2029	4211
Algorithm II	2409.71	731	1296

4.3. Minimizing the Workflow Footprint With Dynamic Cleanup

In our initial simulations [34], we have shown the potential of using data cleanup techniques to reduce the data footprint for LSC-like workflows. Here we observe the performance of the data cleanup algorithms using real applications: Montage and LSC binary inspiral search workflow running on the Open Science Grid. For reasons of clarity and to reduce the effect of cross-site scheduling we focus our experiments on a single grid site. However, the algorithms are also applicable when workflows are scheduled across multiple sites.

Figure 6 shows the results of running the Montage workflow, which creates a one degree square mosaic of the M17 region of the sky. The graph shows the storage usage of the workflow over time for both the original workflow (without cleanup) and the workflow which dynamically cleans up redundant data.

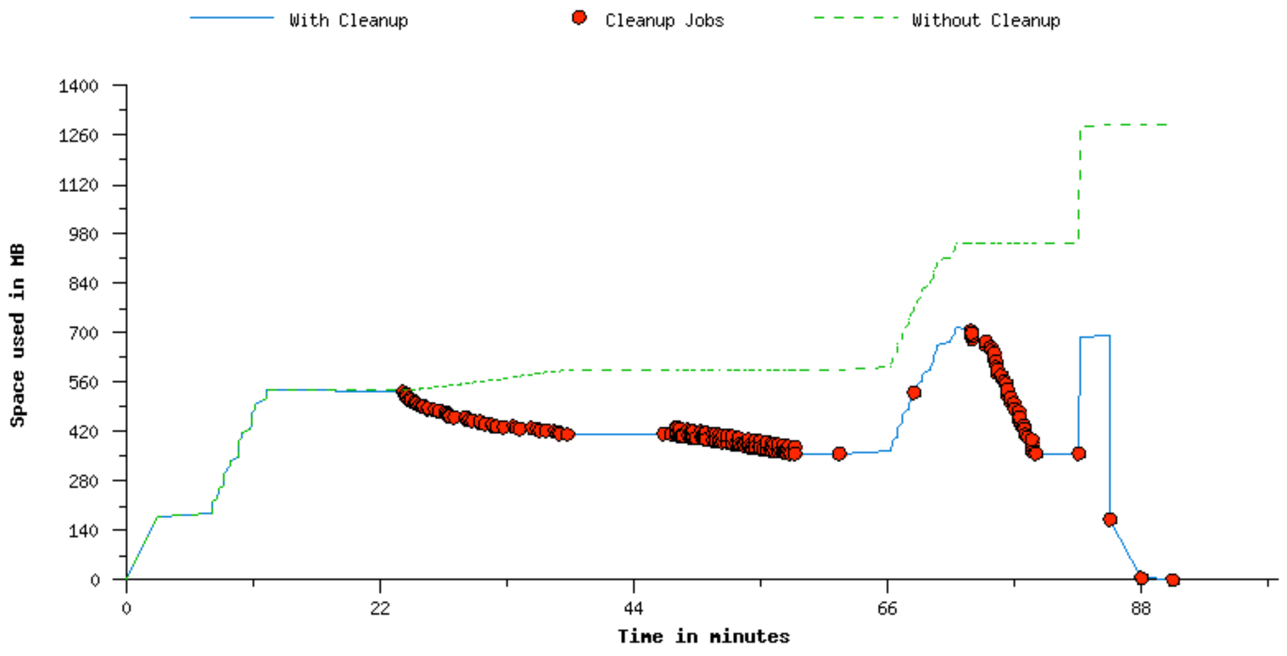


Figure 6. One Degree Square Montage Workflow Data Footprint over Time. The dots indicate completion of cleanup jobs.

The dots signify the completion of a cleanup job. For example, the third from last dot in Figure 6 follows a steep decline in the data footprint that signifies the cleanup done by the job represented by the dot. We can see that in this case we were able to reduce the overall workflow data footprint from 1291.583 MB down to 714.545 MB, a saving of 44.676%. We conducted similar experiments while increasing the size of the mosaics to 2 and 4 degree square. Figure 7 and Figure 8 show the results. For the 2 degree square mosaic the data footprint was reduced from 4.659 GB to 2.421 GB (48%) and for the 4 degree square mosaic from 16.24 to 9.687 GB (40.35%).

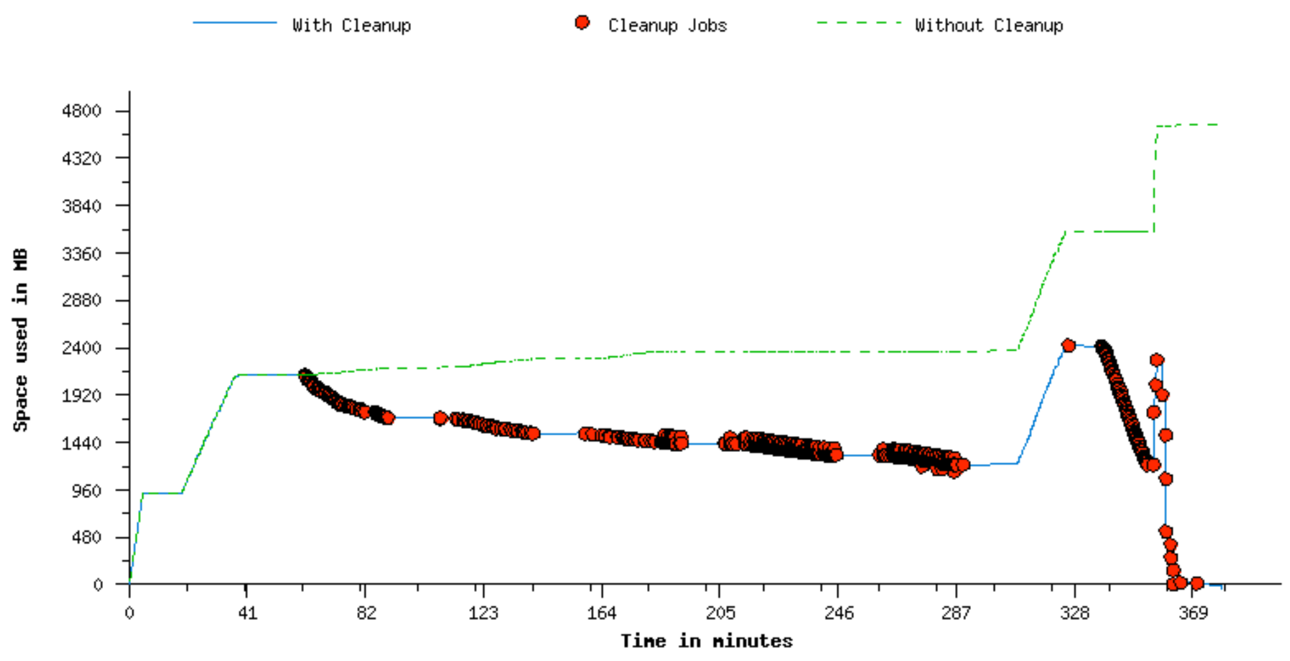


Figure 7: Two Degree Square Montage Workflow Data Footprint over Time. The dots indicate the completion of cleanup jobs.

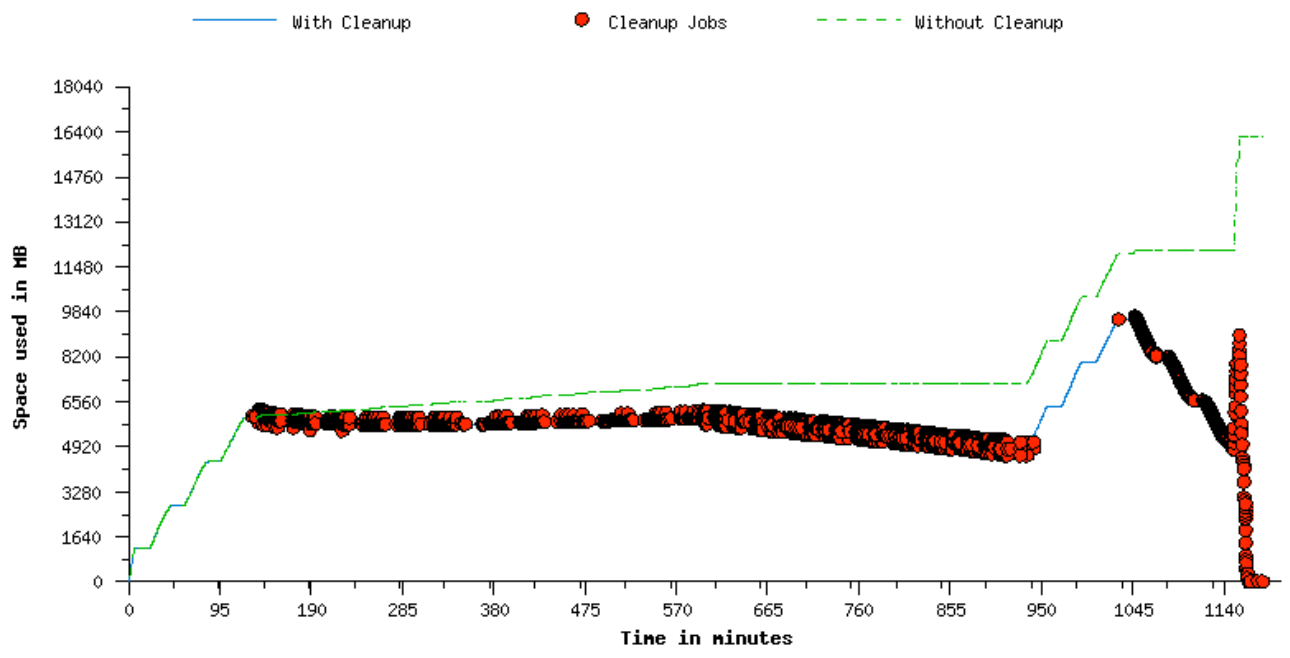


Figure 8: Four Degree Square Montage Workflow Data Footprint over Time. The dots indicate the completion of cleanup jobs.

We also tested our data cleanup algorithm with a small LSC workflow running on the OSG. Figure 9 shows the results. Although, based on our initial simulations [34] we expected a significant reduction in the workflow data footprint, we were not able to see the reduction when the actual workflow was executed on the OSG. We noticed that cleanup is happening in the workflow only after all the input data has already been staged-in. We also measured the runtime overhead of the cleanup tasks. The workflow without cleanup nodes ran in 135 minutes, whereas the workflow with cleanup nodes took 100 minutes. The difference in the runtimes can be attributed to the use of non-dedicated resources and network latencies when scheduling workflow tasks over a wide area network. In the next section we explore the reasons for not achieving a significant reduction in the data workflow footprint for the LSC workflow.

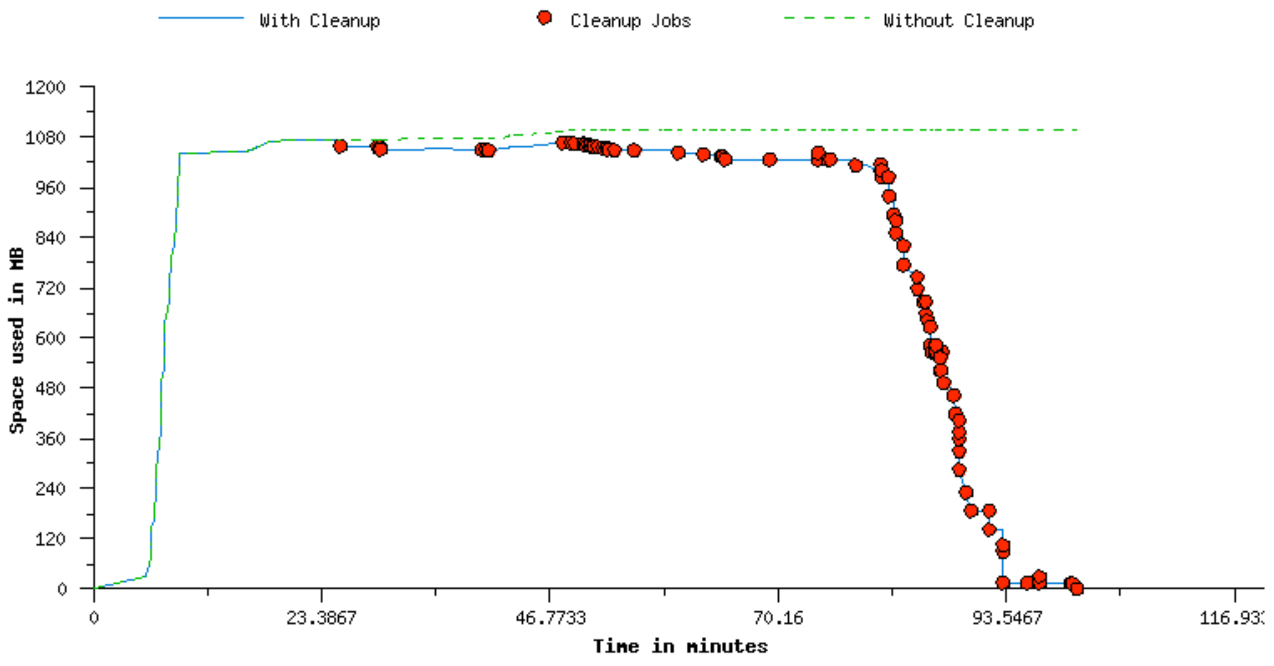


Figure 9: Small LSC Workflow Data Footprint Over Time with and without cleanup.

4.4. Contributions to the workflow footprint

When analyzing the Montage and LSC workflows we see that the opportunities for reducing the workflow footprint for Montage are inherent in the workflow whereas the ones in LSC workflow are limited. The reduction in the workflow footprint results when cleanup opportunities are present in the workflow before the maximum data footprint is reached. In order to illustrate the difference between the Montage and LSC workflow, we divide the workflow into levels. All the tasks in the workflow that have no parent dependencies are assigned level 1. The level of other tasks can be assigned by a breadth-first traversal where the level of any tasks is the maximum level of any of its parent tasks plus one. The number inside the vertices in Figure 1 shows the level number of the tasks in the Montage Workflow. The 2 degree square Montage workflow has 11 levels while the LSC workflow has 6.

Assuming that all the level *i* tasks are executed before tasks in level (*i+1*), Figure 10 and Figure 11 show the total workflow footprint (WFP) at the end of execution of each level without any cleanup and when cleanup is used for the LSC and Montage workflow respectively. The difference between the two represents the cleanup opportunity at each level of the workflow and is also shown in the figure. The difference between the maximum WFP with and without cleanup across all levels represents the maximum reduction in storage space for the workflow. For the LSC workflow (Figure 10), there isn't much difference between the maximum WFP with and without cleanup. The reason is that most of data used or generated by the workflow is already materialized by the end of level one and the cleanup opportunities are mostly absent until level five. For the Montage workflow, the WFP without cleanup gradually increases and cleanup opportunities exist across the workflow (Figure 11). Thus in this case, the maximum WFP with cleanup is only about 28% of the value without cleanup and represents a 72% possible reduction in the footprint of the workflow. In our experiments however, we do not achieve this ideal 72% reduction because the workflow is not scheduled evenly level by level and tasks from multiple levels might end up executing at the same time. Nevertheless, this analysis still gives an important insight into the storage requirements of a workflow and the reduction in the workflow footprint that can be achieved by using the cleanup algorithm alone.

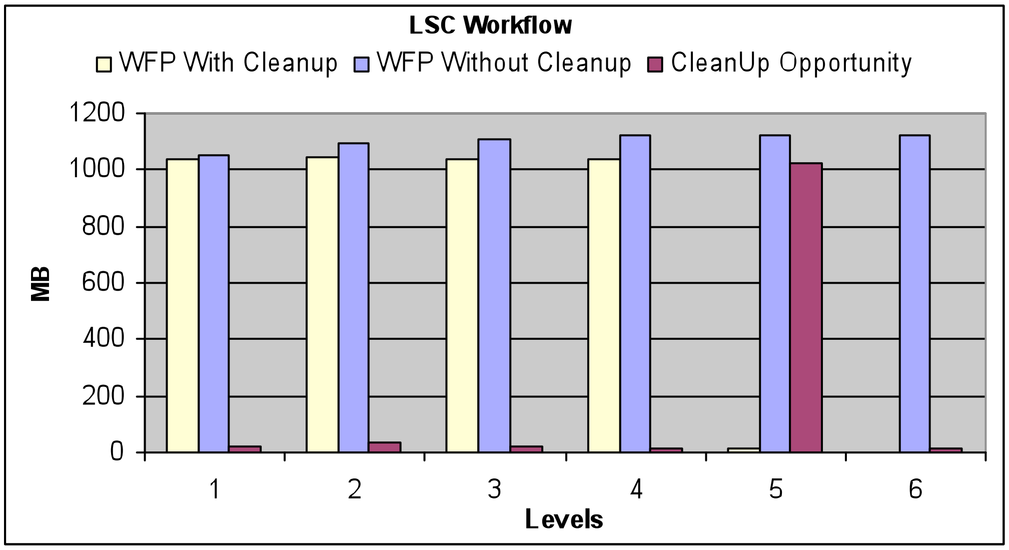


Figure 10: The Workflow footprint (WFP) with and without cleanup and the cleanup opportunities at each level of the LSC Workflow.

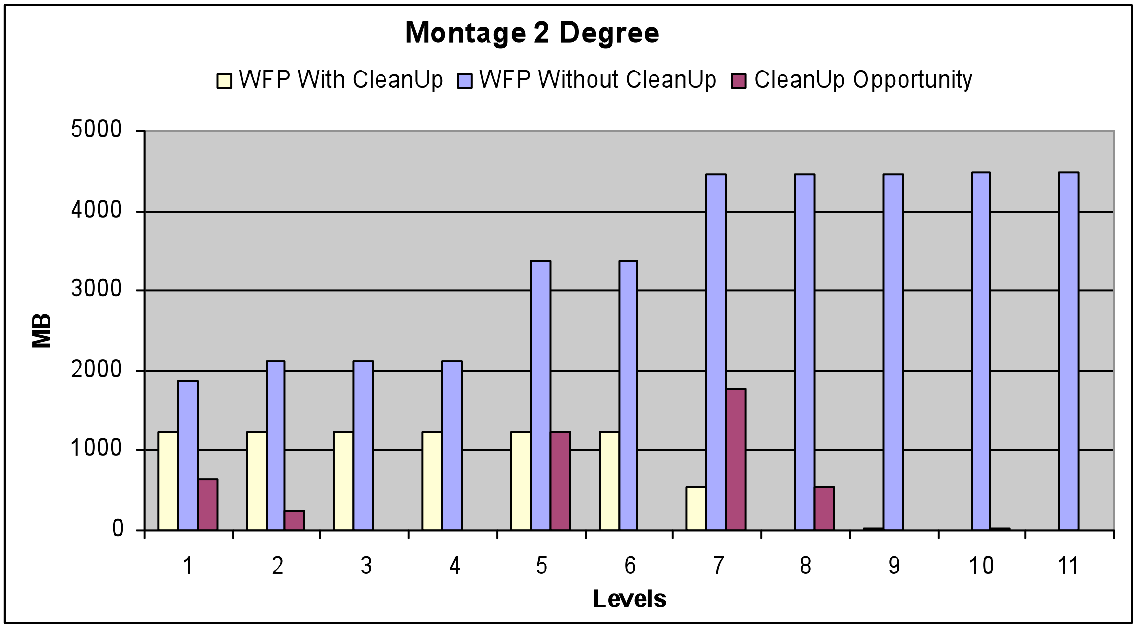
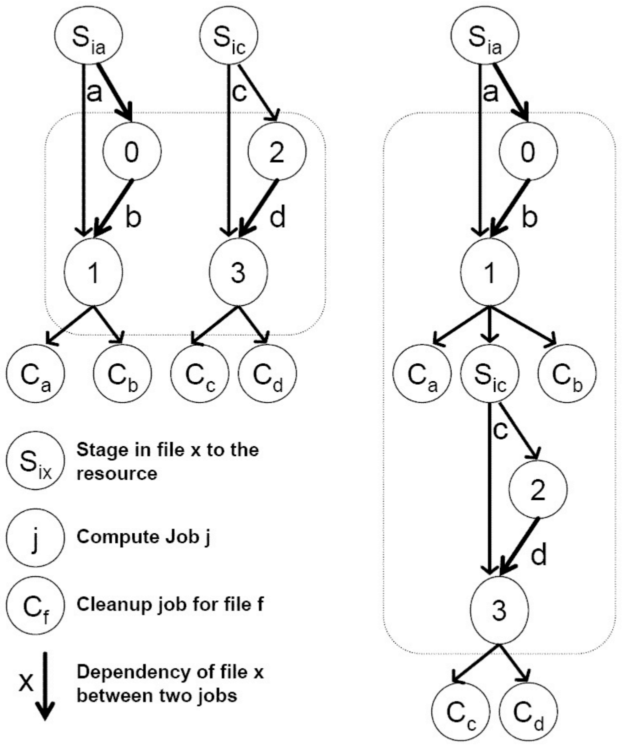


Figure 11: The Workflow footprint (WFP) with and without cleanup and the cleanup opportunities at each level of the Montage Workflow.

5. Reducing the workflow footprint through workflow restructuring

The analysis of the LSC workflow revealed that the workflow was structured such that all the input data would be staged-in at the beginning of the run and not cleaned up until the level 5 of the workflow. One solution we explored was to restructure the workflow by adding extra dependencies so that the computation would progress in a depth-first manner and thus portions of the workflow would reach the computations and the cleanup nodes in level 5 of the original workflow before the remaining level 1 computations would start. By ordering the data stagein jobs for these remaining level 1 computations after the cleanup up jobs of the preceding level 5 tasks, we can achieve a significant reduction in the workflow footprint.

In order to illustrate the principle behind restructuring, Figure 12 shows a simple workflow containing two parallel chains with tasks {0,1} and {2,3} respectively mapped to a single resource. When run without restructuring both the files *a* and *c* would likely be staged in simultaneously and the footprint of the workflow will include both of these files present on the storage system at the same time. The restructuring adds a dependency between the last job of the first chain {1} and the stagein job of the second chain, S_{ic}. Moreover, the cleanup jobs have the highest priority among all jobs. Thus it is very likely that files *a* and *b* would be cleaned up by C_a and C_b before file *c* is staged to the resource by S_{ic}, thus reducing the footprint of the workflow. When these files are large, the restructuring would achieve a significant reduction in the footprint of the workflow.



Stage in file x to the resource
Compute Job j
Cleanup job for file f
Dependency of file x between two jobs

Figure 12: An example workflow before and after restructuring.

We note that the same effect can be achieved with careful scheduling of workflow tasks and data transfers. In some sense we refer to *workflow restructuring* as the ordering or sequencing of the execution of the tasks within the workflow (and this can be done in a resource-independent way), whereas *workflow scheduling* maps the tasks of a workflow onto two or more resources.

5.1. Moderate Workflow Restructuring

Initially, we only did a very limited restructuring of the LSC workflow. Figure 13(a) shows the small LSC workflow and Figure 13(b) shows the restructured workflow. We took the first independent cluster in the LSC workflow (shown by the oval) and made all the remaining jobs in the workflow dependent on it. This assures that this cluster executes first.

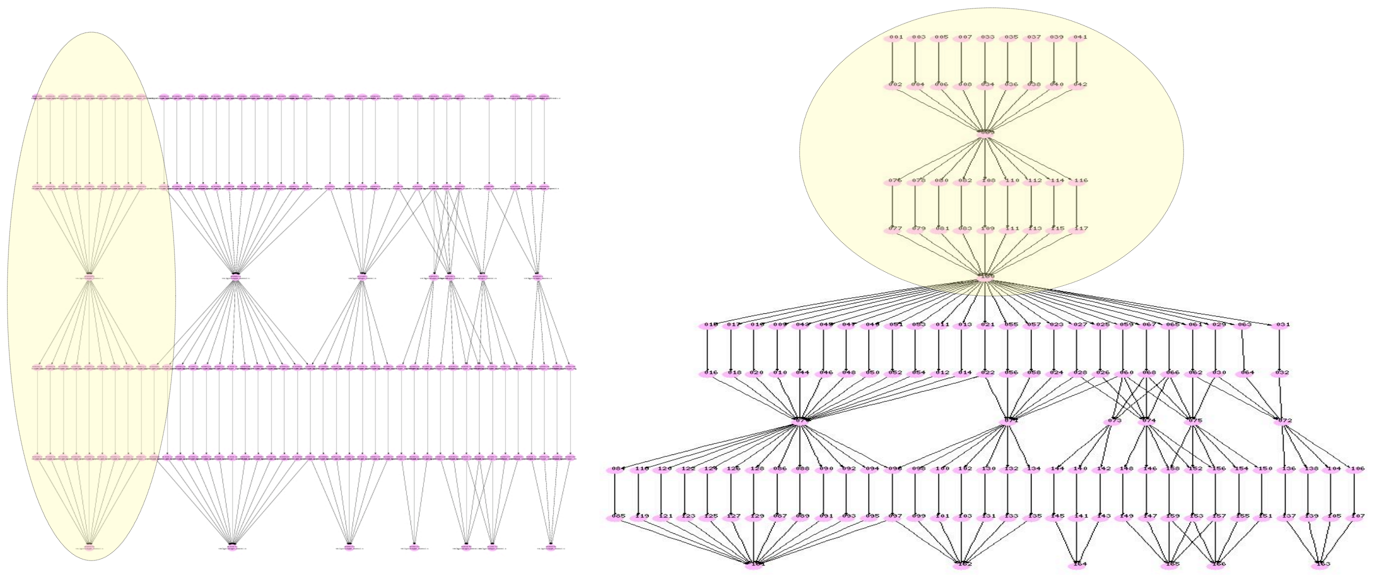


Figure 13: (a) The Original LSC Workflow. (b) The Moderately Restructured LSC Workflow.

This restructuring resulted in a "deeper workflow" of 12 levels instead of the original 6, but it also improved the workflow data footprint. Figure 14 shows the cleanup possibilities for the restructured workflow; they occur at levels 5 and 11. As a result of this distribution of cleanup opportunity, the maximum workflow footprint with cleanup is about 27% less than that without cleanup. We confirm this by both simulation and execution on the grid─Figure 15 and Figure 16 respectively. Our simulations (Figure 15) predicted a saving of 26.5 % in the workflow footprint for the moderately restructured LSC workflow.

An issue that arises in the grid execution is that the executable workflow generated by Pegasus includes data stage-in tasks that stage data into the first level of the workflow. These tasks are not dependent on any other task in the workflow and thus can usually proceed as soon as the workflow is started. However, this poses a problem, because we can easily saturate the data storage and increase the data footprint for workflows structured like LSC. To overcome this deficiency, we made these stage-in tasks dependent on the previous-level computational tasks as shown by an example in Figure 12. We then proceeded with the execution of the restructured LSC workflow on the grid. The actual grid execution (Figure 16) showed a reduction of 26% of disk space used (814.388 MB required for the workflow with cleanup, 1099.387 MB required for the workflow without dynamic cleanup), similar to that observed with the simulation.

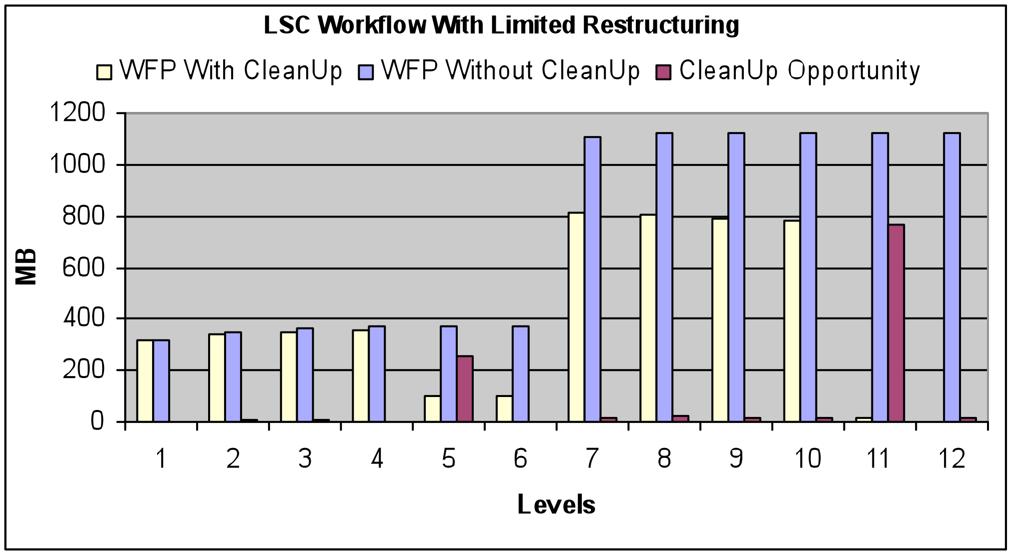
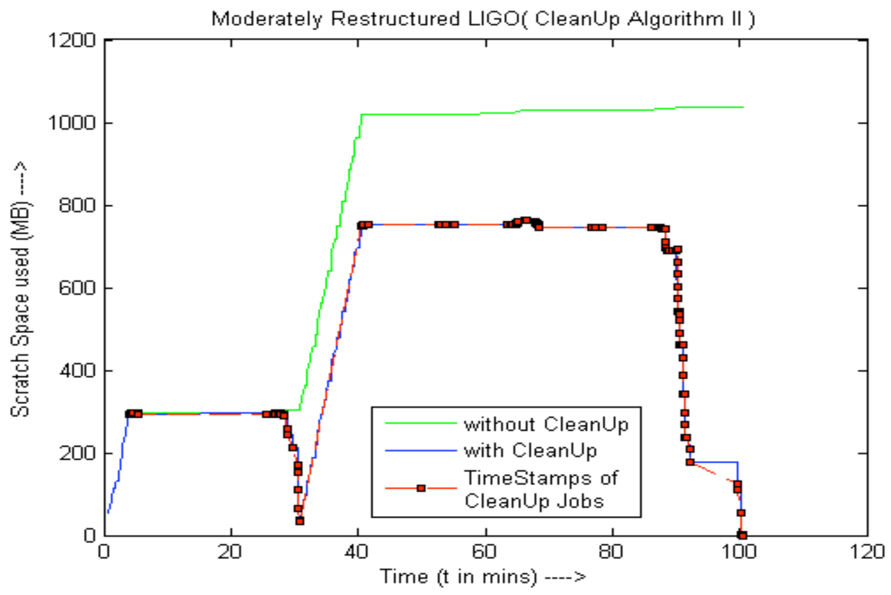


Figure 14: The Workflow footprint (WFP) with and without cleanup and the cleanup opportunities at each level of the Moderately-Restructured Workflow.



.

Figure 15: Simulated Behavior of the Moderately-Restructured Workflow.

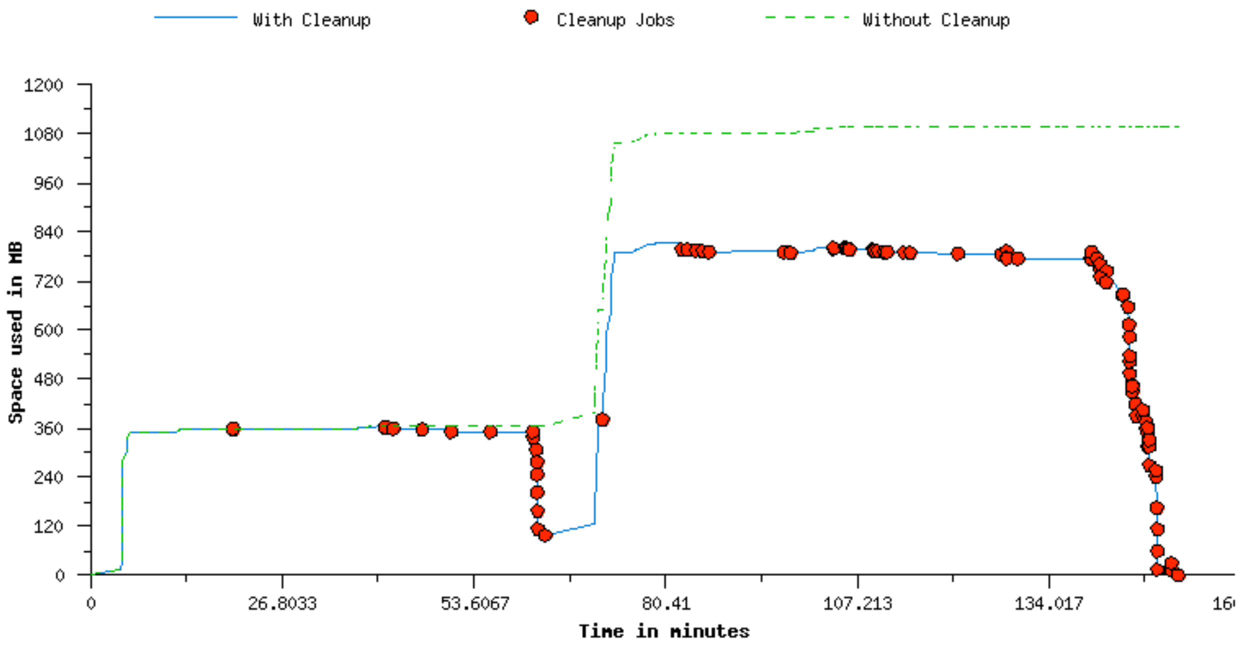


Figure 16: Behavior of the Moderately-Restructured Workflow on the OSG.

5.2. Full Workflow Restructuring

Upon examining the original LSC workflow we noticed additional opportunities for workflow restructuring. Figure 17 shows the most extreme restructuring we can design that would also not introduce cycles within the workflow and would potentially reduce the workflow data footprint.

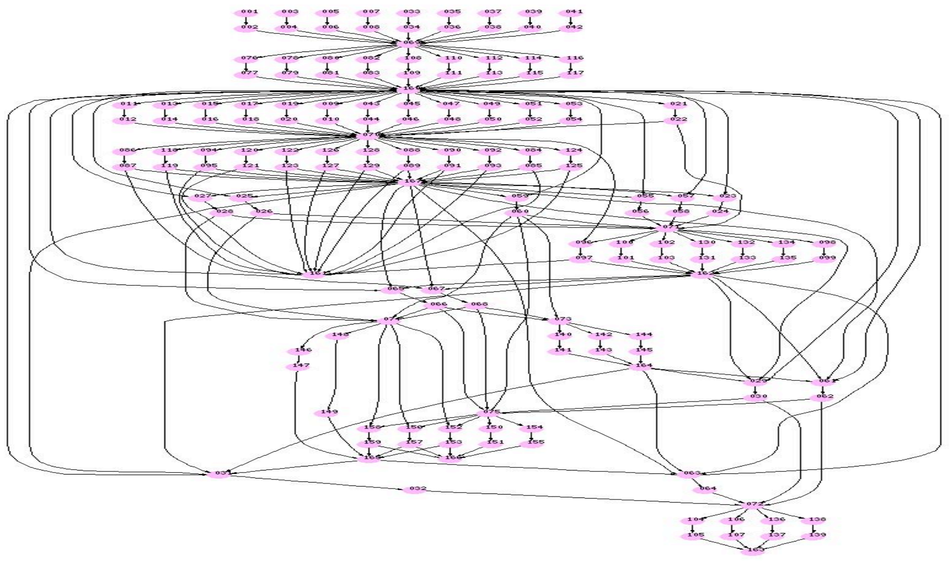


Figure 17: Extremely Restructured LSC Workflow.

Figure 18 shows the cumulative workflow footprint and the cleanup opportunities at each level within this extremely restructured workflow. The restructuring has increased the number of levels in the workflow from the original 6 to 36. However, the cleanup opportunities are more distributed than before and the maximum workflow footprint is decreased from approximately 1,125 MB without cleanup to approximately 500MB with cleanup.

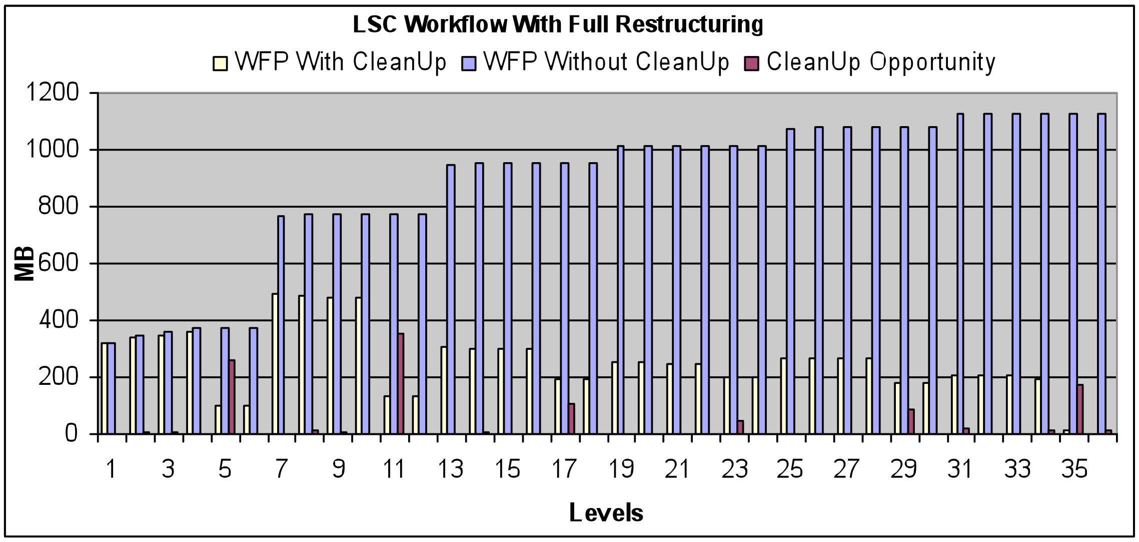


Figure 18: The Workflow footprint (WFP) with and without cleanup and the cleanup opportunities at each level of the Optimally Restructured LSC Workflow.

The corresponding simulation results are presented in Figure 19. The simulation shows a 59% improvement in the workflow data footprint. The actual grid runs shows a similar 56% improvement in data space usage (Figure 20).

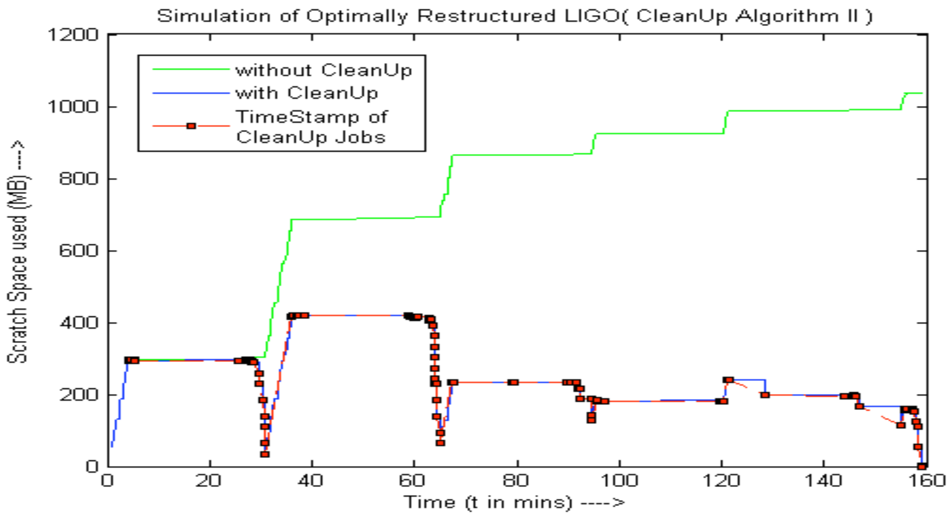


Figure 19: Simulated Behavior of the Optimally Restructured LSC Workflow.

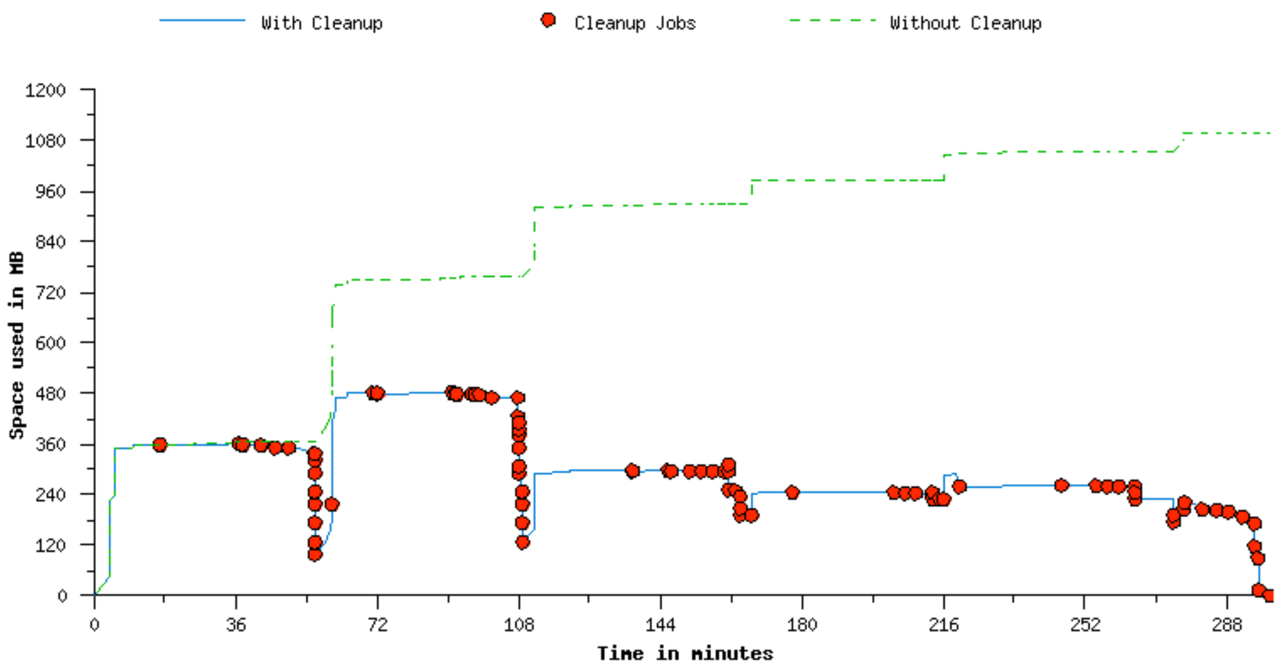


Figure 20: Actual Grid Behavior of the Optimally Restructured LSC Workflow.

Obviously there is a tradeoff between the data footprint and the workflow execution time which is reflected in the increased number of workflow levels. The increase is due to the reduction in the parallelism in the execution. The following figures show the number of workflow tasks running over time. Figure 21 shows the original LSC workflow which has a maximum and average execution parallelism of 36 and 15.8 respectively and a runtime of 100.8 minutes. Figure 22 shows the moderately restructured LSC workflow which exhibits a similar maximum parallelism but has a lower average parallelism of only 9.2. This workflow had an execution time of 152 minutes (an increase of 50% over the original workflow and proportional to the increase in the number of workflow levels). Figure 23 shows the task execution profile of the extremely restructured LSC workflow. The maximum parallelism is reduced to 13 and the average parallelism is 3.1. The workflow execution time is increased to almost 300 minutes—3 times the original workflow execution time reflecting the increase in the number of levels in the workflow to 36.

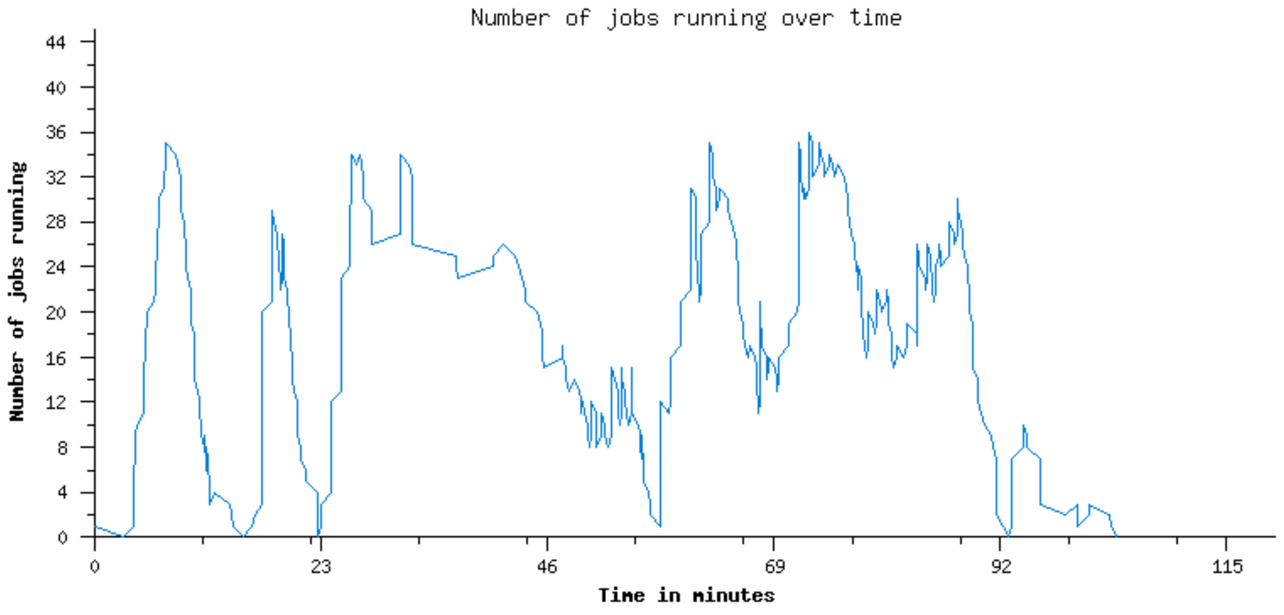


Figure 21: Task Execution Profile of the Original LSC Workflow.

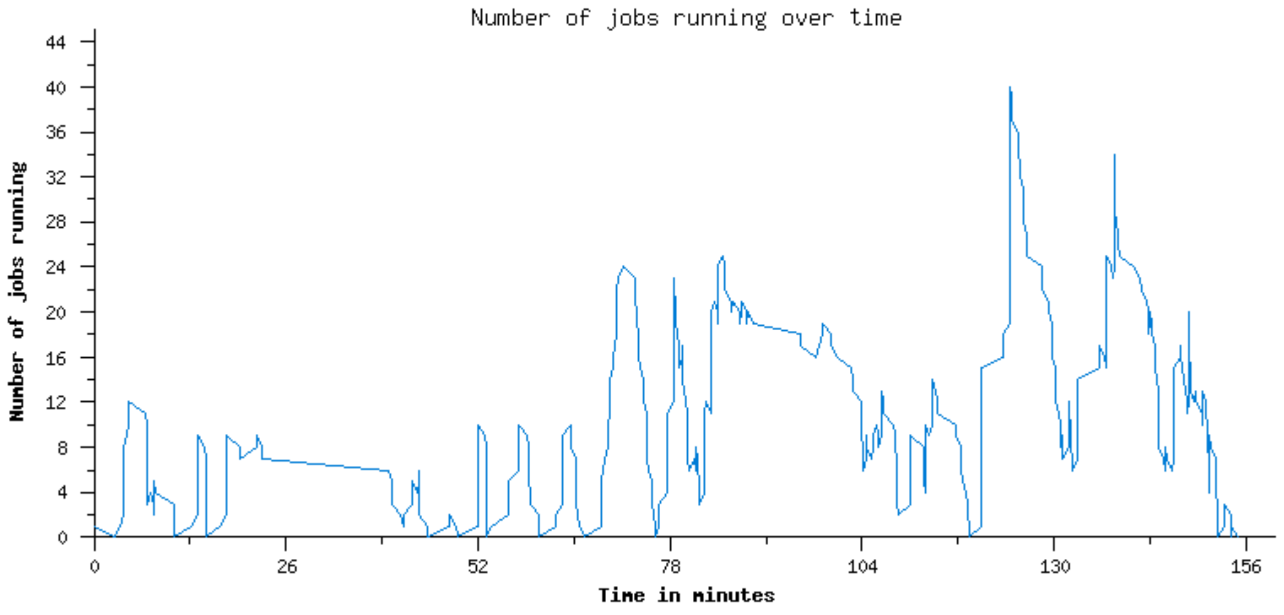


Figure 22: Task Execution Profile of the Moderately Restructured LSC Workflow.

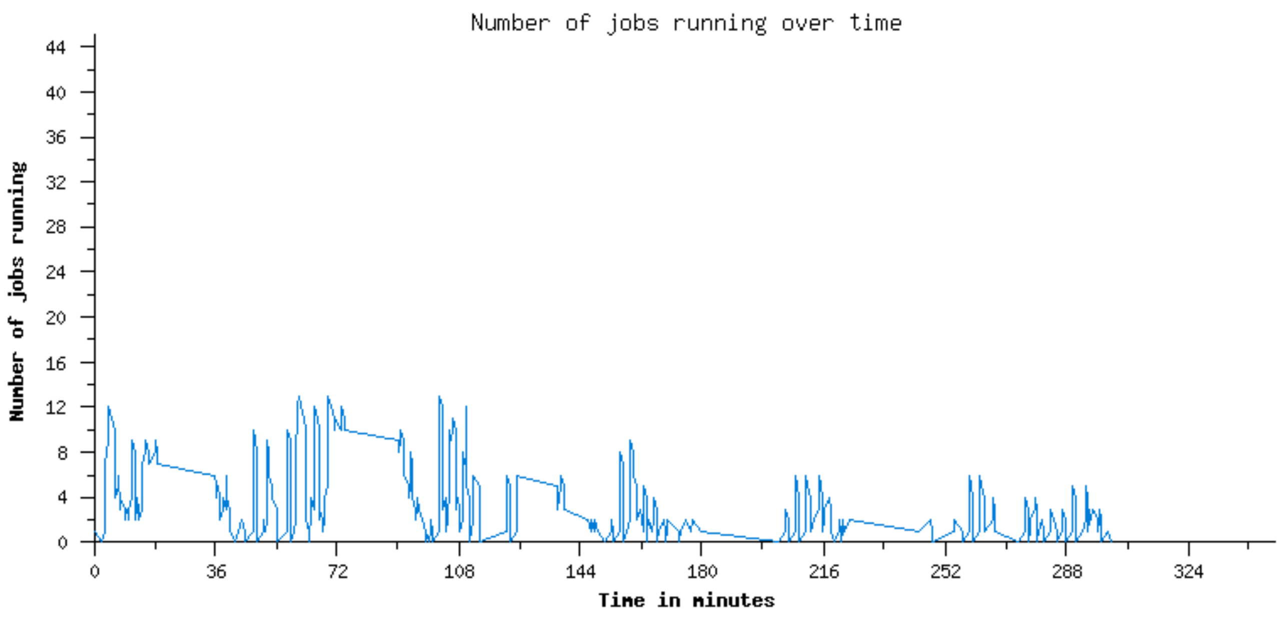


Figure 23: Task Execution Profile of the Optimally Restructured LSC Workflow.

The restructuring described in this section was done by a visual examination of the LSC workflow. Our future work consists of finding algorithmic techniques for restructuring workflows. Importantly, given the tradeoff between the workflow execution time and the data footprint of the workflow, we would like to examine techniques for restructuring a workflow to execute within a given space constraint while minimally affecting the execution time of the workflow.

6. Science drivers for workflow restructuring

In this paper, the LSC workflow considered is a subset of the large scale workflows used for production gravitational-wave data analysis. The workflow shown in Figure 13 contains 164 nodes, whereas full LSC workflows, such as those used to analyze the data taken by the LIGO detectors from November 2005 to November 2006, contain 185,000 nodes and 466,000 edges each. These workflows analyze 10 Tb of input data and produce approximately 1 Tb of output data. Significant restructuring of the workflow will be needed to run such analysis on compute resources with limited storage. The tools used to represent LSC workflows have been designed with flexibility to allow easy re-factoring of workflows to avoid being locked into a given environment or computing system.

As part of our work we noticed that the LSC workflows, as they are currently structured, impose challenges on the ability to minimize the workflow data footprint. LSC scientists have been aware of this and other issues relating to the structure of the workflow described in [7]. Consider the LSC workflow as shown in Figure 13(a). The first and second levels of the workflow analyze all available data from the three LIGO detectors, and then the third level applies consistency tests by comparing the analysis products from each detector for a given time period. The fourth and fifth levels of the workflow reanalyze the data, using information gained from these consistency tests, and then a final consistency test is applied at level 6. These consistency tests are applied to data in blocks on the order of a day. Once the second test is complete, that block of data has been completely analyzed [7] and the results are ready for the LSC scientist to review.

Since there are no direct dependencies between nodes at the first level of the workflow, all these nodes are submitted to compute resources before subsequent levels. These nodes, and subsequent level 2 nodes, prevent any consistency tests from being applied until the entire data set has been analyzed. Thus, results are not available until a majority of the data set has been analyzed. This structure was chosen for expedience in the early stage of development, but has drawbacks for large-scale offline batch processing. In our future work, we intend to work with LSC scientists to analyze possible restructuring of the LSC workflows to improve their efficiency. Minimizing workflow data footprint would allow to leverage Grid resources with limited storage and a restructured LSC workflow could be much more efficient, which is an advantage for obtaining staged results in the offline analysis of longer data sets.

7. 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 [27]. In recent years, there has been a revival of interest in the context of problems especially motivated by scientific workflow execution and heterogeneous environments [16, 31, 41, 42] . In the majority of these works, data used or materialized in the workflow affects resource selection only to the extent of minimizing the data transfer time between resources with the goal of minimizing the overall workflow execution time. No work has taken into account the available data storage when selecting resources, which can be a critical factor when executing data-intensive workflows. However, in a national level Grid infrastructure, unavailability of storage space was cited as the significant cause of job failures [24].

The most interesting work in the context of this paper, which considers data placement, has been presented in [35, 36]. 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. The scheduling algorithm of [35, 36] has been extended to a heterogeneous resource environment in [40]. This has been further extended in [39] for situations where a task requires data from multiple sources. However the focus is resource selection with the goal of minimizing the task completion time and the storage capacity of the resources is not a candidate for consideration.

In our previous work, we presented algorithms for scheduling workflows on storage constrained resources [34]. Storage-aware scheduling was found to be more reliable and perform better than scheduling algorithms that do not consider storage constraints. A related problem is to be able to provision storage resources in advance. Without the ability to guarantee certain storage availability at a resource, scheduling algorithm that operate using storage constraints would have limited effectiveness. Recently, low level mechanisms for allowing user level allocation of storage resources have been proposed [38]. This work also shows that the resource providers can benefit from storage allocation in the form of increased output under high utilization conditions by isolating the users from one another. Storage management systems such as Storage Resource Manager (SRM) [37], Storage Resource Broker (SRB) [11], and NeST [12] allow users to request storage allocation.

8. Conclusions

In this paper, we have described two algorithms for reducing the data footprint of workflow type applications. We have evaluated the performance of these algorithms using both simulation and actual workflow execution on the Open Science Grid. In the case of Montage, we were able to obtain a reduction of approximately 48% in the amount of disk space used. However, in order to be able to achieve a significant reduction in the footprint for the LSC workflow (56%), we needed to restructure it in a way that impacted the parallelism in the workflow. Although for this paper, we restructured the workflow by hand, we plan to investigate automated techniques in the future. Workflow restructuring is only an element in the footprint reduction, further improvements can potentially be gained by using data-aware workflow scheduling techniques. We explored some of these techniques via simulation and plan to evaluate them in real grid deployments. The data cleanup algorithm described in this paper has been implemented in the Pegasus workflow management system and is being used for real applications.

Acknowledgements

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 also thank the Open Science Grid for resources used for the motivation of this work. K. Blackburn and D. Meyers were supported by the LIGO Laboratory and NSF grants PHY-0107417 and PHY-0326281. The work of D. Brown was supported by the LIGO Laboratory and NSF grant PHY-0601459. The work of S. Fairhurst was supported by the LIGO Laboratory and NSF grant PHY-0326281 and PHY-0200852. LIGO was constructed by the California Institute of Technology and the Massachusetts Institute of Technology and operates under cooperative agreement PHY-0107417. This paper has been assigned LIGO Document Number LIGO-P070017-00-Z. Montage was supported by the NASA Earth Sciences Technology Office Computing Technologies (ESTO-CT) program under Cooperative Agreement Notice NCC 5-6261. 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] "Condor DAGMan." http://www.cs.wisc.edu/condor/dagman.
[2] "Cray Fortran Reference Manual."
[3] "Montage," in *http://montage.ipac.caltech.edu*.
[4] "Open Science Grid."
[5] "The Two Micron All Sky Survey at IPAC (2MASS)," in *http://www.ipac.caltech.edu/2mass/*.
[6] *Workflows in e-Science*: Springer, 2006.
[7] B. Abbott, "Search for gravitational waves from binary inspirals in S3 and S4 LIGO data," under preparation.
[8] B. Abbott, et al., "Search for gravitational waves from binary black hole inspirals in LIGO data," *Physical Review D (Particles, Fields, Gravitation, and Cosmology)*, vol. 73, pp. 062001-17, 2006.
[9] B. Abbott, et al., "Search for gravitational waves from primordial black hole binary coalescences in the galactic halo," *Physical Review D (Particles, Fields, Gravitation, and Cosmology)*, vol. 72, pp. 082002-8, 2005.
[10] B. C. Barish and R. Weiss, "LIGO and the Detection of Gravitational Waves," *Physics Today*, vol. 52, pp. 44, 1999.
[11] C. Baru, et al., "The SDSC Storage Resource Broker," presented at Proc. CASCON'98 Conference, 1998.
[12] J. Bent, et al., "Flexibility, manageability, and performance in a Grid storage appliance," presented at 11th IEEE International Symposium on High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings., 2002.
[13] B. Berriman, et al., "Montage: A Grid-Enabled Image Mosaic Service for the NVO," presented at Astronomical Data Analysis Software & Systems (ADASS) XIII, 2003.
[14] G. B. Berriman, et al., "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand," presented at SPIE Conference 5487: Astronomical Telescopes, 2004.
[15] G. B. Berriman, et al., "Generating Complex Astronomy Workflows," in *Workflows for e-Science*, I. J. Taylor, et al., Eds.: Springer, 2006.
[16] J. Blythe, et al., "Task Scheduling Strategies for Workflow-based Applications in Grids," presented at CCGrid, Cardiff, UK, 2005.

[17] D. 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*, I. J. Taylor, et al., Eds.: Springer, 2006.
[18] E. Deelman, et al., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," presented at 11th Intl Symposium on High Performance Distributed Computing, 2002.
[19] E. Deelman, et al., "Workflow Management in GriPhyN," in *Grid Resource Management*, J. Nabrzyski, et al., Eds.: Kluwer, 2003.
[20] E. Deelman, et al., "Pegasus : Mapping Scientific Workflows onto the Grid," presented at 2nd EUROPEAN ACROSS GRIDS CONFERENCE, Nicosia, Cyprus, 2004.
[21] E. Deelman, et al., "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance tracking: The CyberShake Example," presented at 2nd IEEE International Conference on e-Science and Grid Computing, Amsterdam, The Netherlands, 2006.
[22] E. Deelman, et al., "Pegasus: Mapping Large-Scale Workflows to Distributed Resources," in *Workflows for e-Science*, D. Gannon, et al., Eds.: Springer, 2006.
[23] E. Deelman, et al., "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming Journal*, vol. 13, pp. 219-237, 2005.
[24] I. Foster, et al., "The Grid2003 production grid: principles and practice," presented at 13th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2004.
[25] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, pp. 115-128, 1997.
[26] J. Frey, et al., "Condor-G: A Computation Management Agent for Multi-Institutional Grids," presented at 10th International Symposium on High Performance Distributed Computing, 2001.
[27] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors " *ACM Comput. Surv.* , vol. 31 pp. 406-471 1999
[28] A. Lathers, et al., "Enabling Parallel Scientific Applications with Workflow Tools," presented at Challenges of Large Applications in Distributed Environments (CLADE), Paris, 2006.
[29] M. Litzkow, et al., "Condor - A Hunter of Idle Workstations," in *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988, pp. 104-111.
[30] P. Maechling, et al., "SCEC CyberShake Workflows---Automating Probabilistic Seismic Hazard Analysis Calculations," in *Workflows for e-Science*, D. Gannon, et al., Eds.: Springer, 2006.
[31] A. Mandal, et al., "Scheduling Strategies for Mapping Application Workflows onto the Grid," presented at The 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), 2005.
[32] V. Nefedova, et al., "Automating Climate Science: Large Ensemble Simulations on the TeraGrid with the GriPhyN Virtual Data System," presented at e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on, 2006.
[33] M. M. Rajkumar Buyya, "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.
[34] A. Ramakrishnan, et al., "Scheduling Data-Intensive Workflows onto Storage-Constrained Distributed Resources," presented at 7th IEEE International Symposium on Cluster Computing and the Grid - CCGrid, 2007 (to appear).
[35] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications.," presented at International Symposium for High Performance Distributed Computing (HPDC-11), Edinburgh, 2002.
[36] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a High Performance Data Grid," presented at International Workshop on Grid Computing, 2001.

[37] A. Shoshani, et al., "Storage resource managers: Middleware components for grid storage," presented at Nineteenth IEEE Symposium on Mass Storage Systems, 2002.
[38] D. Thain, "Operating System Support for Space Allocation in Grid Storage Systems," presented at 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, 2006.
[39] S. Venugopal and R. Buyya, "A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids," presented at 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, 2006.
[40] S. Venugopal, et al., "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 685-699, 2006.
[41] M. Wieczorek, et al., "Scheduling of scientific workflows in the ASKALON grid environment " *SIGMOD Rec.* , vol. 34 pp. 56-62 2005
[42] J. Yu and R. Buyya, "A Budget Constraint Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms," presented at Workshop on Workflows in Support of Large-Scale Science (WORKS06), Paris, France, 2006.