Workflow Task Clustering for Best Effort Systems with Pegasus

Gurmeet Singh[1], Mei-Hui Su[1], Karan Vahi[1], Ewa Deelman[1], Bruce Berriman[2], John Good[2], Daniel S. Katz[3], and Gaurang Mehta[1]
[1]USC Information Sciences Institute, Marina Del Rey, CA 90292
{gurmeet, mei, vahi, deelman, gmehta}@isi.edu
[2]Infrared Processing and Analysis Center, California Institute of Technology, Pasadena, CA 91125
{gbb, jcg}@ipac.caltech.edu
[3]Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803
dsk@cct.lsu.edu

ABSTRACT

Many scientific workflows are composed of fine computational granularity tasks, yet they are composed of thousands of them and are data intensive in nature, thus requiring resources such as the TeraGrid to execute efficiently. In order to improve the performance of such applications, we often employ task clustering techniques to increase the computational granularity of workflow tasks. The goal is to minimize the completion time of the workflow by reducing the impact of queue wait times. In this paper, we examine the performance impact of the clustering techniques using the Pegasus workflow management system. Experiments performed using an astronomy workflow on the NCSA TeraGrid cluster show that clustering can achieve a significant reduction in the workflow completion time (upto 97%).

Keywords

Workflow clustering, task clustering, best effort systems, queue wait time.

1. INTRODUCTION

Large-scale applications in different scientific fields such as astronomy [1], biology [2], physics [3], earthquake-science [4] are often structured as a set of interdependent tasks also known as workflows. Due to the large resource requirements of these applications, they are often executed using resources from various collaborative organizations such as Open Science Grid (OSG) [5] and national supercomputing centers such as the TeraGrid [6]. These resources are shared and autonomous and often managed using queuing based resource management systems such as PBS [7], Condor [8] etc. The quality of service is often best effort in nature and the response time of the applications cannot be predicted in advance. Additionally due to the high utilization levels of these production resources [9], the applications experience large slowdown due to long queue wait times at the resources.

Pegasus [10] is a framework for mapping and executing workflows on distributed computational resources such as the TeraGrid and OSG. Pegasus has been used for enabling the execution of various scientific applications [1, 3, 4] on the national cyberinfrastructure. In this paper, we discuss the optimizations incorporated in the Pegasus system for improving the response time of workflows using a workflow restructuring technique that clusters workflow tasks. This technique is especially suited to fine granularity workflows where the runtimes of the tasks in the workflows are very small (seconds to few minutes). Yet, due to their data-intensive nature and overall computational needs, these workflows require resource-rich execution environments such as the TeraGrid. One such application is Montage [11], which is used to create custom science-grade mosaics of regions of the sky. A typical Montage workflow contains thousands of tasks most of which have a running time of minutes or less.

There are several problems in executing these workflows in their original form. The typical wait time experienced by the tasks in the resource job queue is often much more than their runtime leading to a workflow completion time that is significantly greater than what can be achieved on a dedicated system. These workflows have a high degree of parallelism and a large number of tasks can execute concurrently. When these tasks are submitted to the resource queues, they overload critical resources such as the main submission node. In order to cope, the resource management systems impose limitations on the number of tasks that a user can submit at a time thereby throttling the execution of the workflow. Additionally, resouces such as the TeraGrid have a processing and an accounting cost associated with the execution of each job.

The approach taken in Pegasus to deal with this issue is to group jobs into clusters and execute a cluster as a single task. With this approach, the number of tasks in the workflow is greatly reduced and the queue wait time of the cluster is amortized over all the tasks in the cluster leading to smaller completion times for the workflow. Additionally, the load on the head node of the remote clusters and the accounting costs are decreased due to the smaller number of jobs being executed. Additionally, Pegasus also supports overlay computing, where a set of nodes are temporarily acquired by the user from the remote resource by using middleware tools such as Condor Glidein and then the workflow is scheduled on these nodes bypassing the remote scheduler. This has the dual affect of incurring the queue wait time only once and reducing the scheduling load on the main submission node of the remote resource.

In Section 2 we describe the structure of the Montage workflow. Section 3 describes the various clustering techniques implemented in Pegasus and the issues associated with clustering. Section 4 describes the experimental results. Section 5 presents the case where the entire workflow is executed as a single cluster. Related work is presented in Section 6 followed by directions for future work in Section 7.

MG '08, January 29-February 3 2008, Baton Rouge, LA, USA

2. MONTAGE WORKFLOW

Montage is an application for constructing custom astronomical image mosaics of the sky [12, 13]. Figure 1 shows the structure of a small Montage workflow. The vertices represent the compute tasks and the edges represent the data dependencies between them. The number within the vertices represents the level of the task in the workflow. All tasks that have no parent tasks are at level one. The level of any other task is the maximum level of any of its parents plus one. All the tasks at the same level are independent of each other. The Montage workflow is such that each level consists of the same module working on different input data. For the experiments we use three different size Montage workflows: these workflows are used to create one, two and four square degrees mosaics of the M17 region of the sky (the number of tasks in the workflow increases with the number of degrees). Table 1 shows the name of the module and number of tasks at each level of the three montage workflows and the average runtime of each module. The working description of each of these modules can be found at [14].

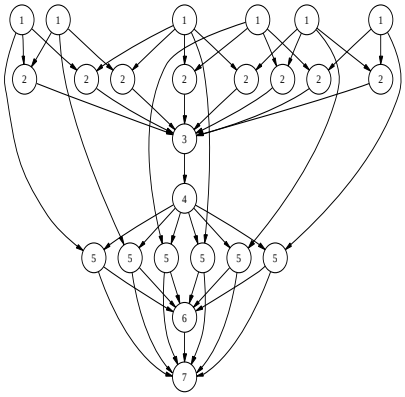

Figure 1. The structure of a small Montage workflow.

Table 1. Number of task per level and average runtimes of modules in three Montage workflows.

Level	Module	# tasks (1 sq deg)	# tasks (2 sq deg)	# tasks (4 sq deg)	Average runtime (seconds)
1	mProject	45	152	610	37
2	mDiffFit	107	410	1754	35
3	mConcat	1	1	1	15
4	mBgModel	1	1	1	10
5	mBackground	45	152	610	131
6	mImgtbl	1	4	16	10
7	mAdd	1	4	16	70
	Total	201	724	3008	

3. WORKFLOW CLUSTERING IN PEGASUS

Pegasus is a workflow management system [10, 15, 16] for mapping and executing complex scientific workflows on the Grid. It takes an input an abstract workflow and converts into an executable workflow by mapping tasks to Grid resources, transferring the task executables to those resources, discovering sources for input data and adding data transfer nodes to the workflow. The final executable workflow could be executed on a local condor pool or on remote resource using Condor-G [17] and Condor DAGMan[18]. Pegasus can also reduce workflows based on the data already materialized in the Grid. In this paper, we focus on the support for workflow clustering in Pegasus for minimizing the completion time of the workflows.

Pegasus currently implements level- and label- based clustering. In level-based clustering, tasks at the same level can be clustered together. The user can specify either the number of clusters to be created per level or the number of tasks to be grouped in a cluster. Figure 2 shows the Montage workflow in Figure 1 clustered with two clusters per level (left) and two tasks per cluster (right).

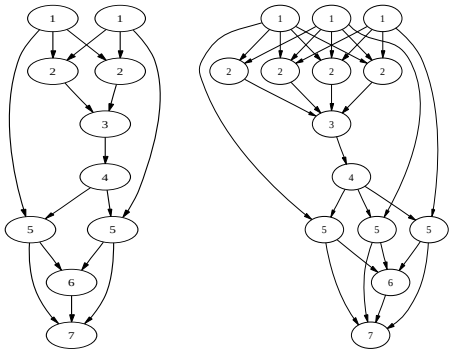

Figure 2. Montage workflow clustered with two clusters per level (left) and two tasks per cluster (right).

In label-based clustering, the user can label the tasks in the workflow to be clustered together. The tasks in the workflow with the same label are grouped into a single cluster. Figure 3(1) shows a workflow where tasks are labeled as *cluster_1* and *cluster_2* and the resulting clustered workflow is shown in Figure 3(2). Thus any clustering scheme can be implemented using an appropriate labeling program.

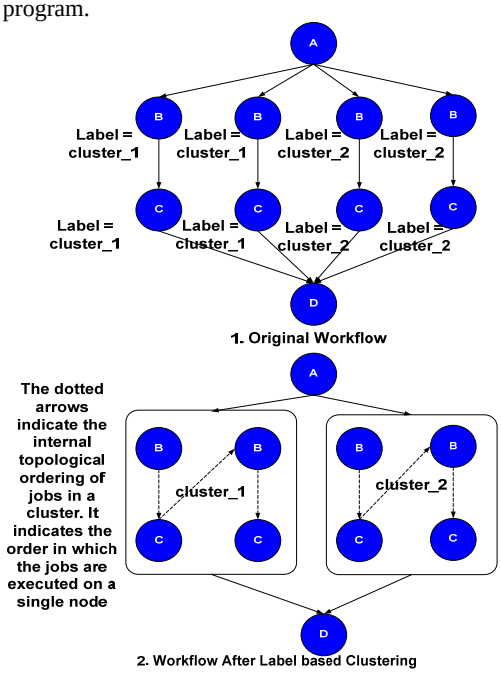

Figure 3. Example of label based clustering.

In some cases, a user may want to use a combination of level-based and label-based clustering techniques. Pegasus supports successive applications of clustering techniques. For example, a workflow can be clustered using label-based clustering and the resulting clustered workflow can be further clustered using level-based clustering. An example scenario is illustrated in **Figure 4** where the label clustered workflow of Figure 3(2) is further clustered by clustering tasks at level two into a single cluster.

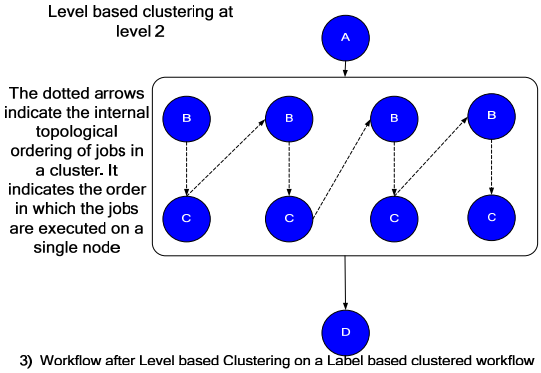

Figure 4. Overlaying clustering techniques.

Each cluster whether generated using level- or label- based clustering must satisfy the convexity requirement that dictates that all paths between any two tasks in a cluster must be completely contained within it. The cluster shown in Figure 5 is non-convex since the path from t1 to t3 through t4 is not contained within the cluster. The difficulty here is that t4 must start execution after t1 has completed and before t3 starts execution. Thus it creates co-scheduling requirements between clusters. However, due to the best effort nature of the execution environment, it is not possible to achieve co-scheduling without explicit resource control.

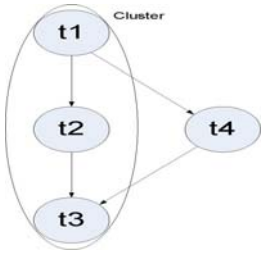

Figure 5. A non-convex cluster.

Pegasus does error checking to ensure that each cluster created by grouping the tasks with the same label satisfies the convexity requirement. Note that the clusters generated using level based clustering trivially satisfy the convexity requirement since all the tasks at a level are independent of each other and no path exists between them. Another restriction of clustering is that the tasks within a cluster be scheduled to the same resource.

A secondary issue after clustering has been done is to decide how to execute the tasks in the cluster. Note that the tasks in a cluster can represent a directed acyclic graph in case of label-based clustering. Our current approach for this case is to create a topological ordering of the tasks in the cluster and execute them sequentially based on this order. This entails a loss in parallelism since the clustered tasks can be potentially executed in parallel (e.g. level-based clusters). However, it greatly simplifies the design of the wrapper program used to execute the cluster and at the same time ensures that all the dependency requirements are met.

In case of level-based clustering, we have more flexibility in how to execute the jobs in the cluster. Since, the jobs in a level-based cluster are always independent of each other, order is not important. Hence, we can execute the jobs in parallel if required. In this case, the clustered job can be executed using *mpiexec*, a wrapper MPI program written in C that is distributed with Pegasus. The wrapper when invoked on the remote resource is run on every MPI process, with the first process being the master and the rest of the processes acting as workers. The number of instances of *mpiexec* that are invoked is equal to the number of nodes requested in the job submission description. The master distributes the constituent jobs to the workers.

4. EXPERIMENTS AND RESULTS

In order to evaluate the performance of the various clustering schemes, we executed the three Montage workflows described in Table 1 on the NCSA TeraGrid cluster using level- and label- based clustering.

4.1 Level-based Clustering

For the level based clustering experiments described in this section, the tasks in a cluster were executed sequentially. The requested wall clock time of a cluster was the sum of the wall clock times of the tasks in the cluster. The number of clusters per level of the workflow is referred to as the clustering factor.

To illustrate the differences between the execution profile of an unclustered and clustered workflow, Figure 6 shows the queued and running times of the tasks in an unclustered one degree Montage workflow (Table 1). The X-axis shows the progression of time after the workflow was submitted for execution. The Y-axis shows the task identifiers. For each task we plot the time when it was submitted to the NCSA TeraGrid queue, the time when it started running and when it finished running. As the figure shows, the tasks in the workflow experience queue delays that are significantly more than their running times. Figure 7 shows the execution of the same workflow after being clustered using level-based clustering with a clustering factor of 5. In this case, there are far fewer number of tasks (clusters) in the workflow and they experience relatively shorter queue delays leading to a faster completion time (both Figure 6 and Figure 7 are on the same time-scale).

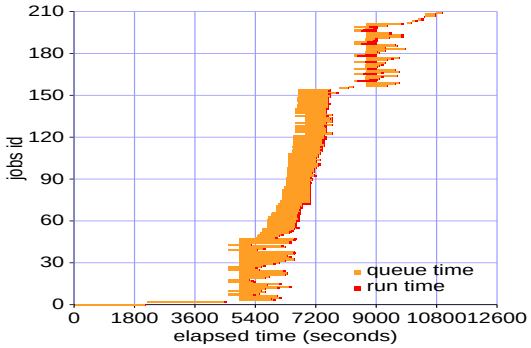

Figure 6. The submit, start and finish times of tasks in one degree workflow without clustering.

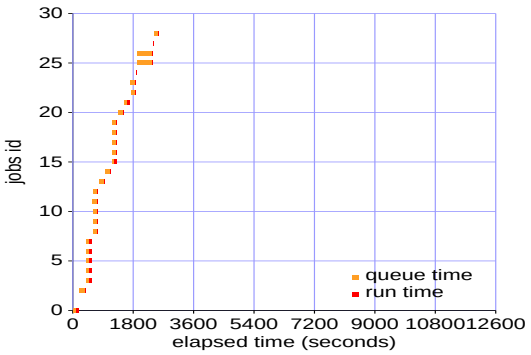

Figure 7. The submit, start and finish time of clusters in one degree workflow with clustering.

For the rest of the experiments in level-based clustering, we use a clustering factor of 1, 5, and 10 on 1, 2, and 4 square degree Montage workflows. Figure 8 shows the workflow completion times with different clustering factors and without clustering. The completion times are the average of three runs. The only exception is the 4 square degree Montage workflow which we could not execute more then once without clustering due to the significant number of tasks in the workflow 3008). The workflow completion times with clustering are considerably less then the unclustered completion time of the workflows. Taking the average over the three clustering factors, clustering reduced the workflow completion time by 68%, 72%, and 65% for the one, two, and four square degree Montage workflows respectively. In the best case (4 sq degree, 10 clustering factor), the reduction in time is 82%.

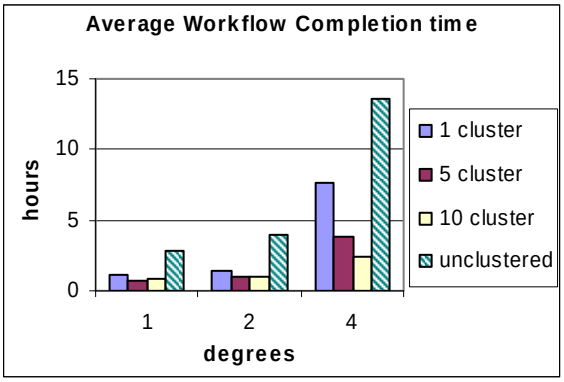

Figure 8. Workflow completion times with level based clustering.

Within the various clustering factors, there is little difference except for the 4 square degree workflow where reducing the clustering factor seems to increase the completion time of the workflow: as the clustering factor decreases, the requested wall clock time of clusters increasesand hence the TeraGrid scheduler has fewer opportunities to backfill them efficiently, or they get put into a slower queue on the resource resulting in longer queue wait time for these clusters.

We also plot the average slowdown of the tasks(clusters) for the same experiment in Figure 8. The slowdown is defined as *(queue wait time + runtime)/runtime* and is used to capture the impact of the queue wait times on the tasks.

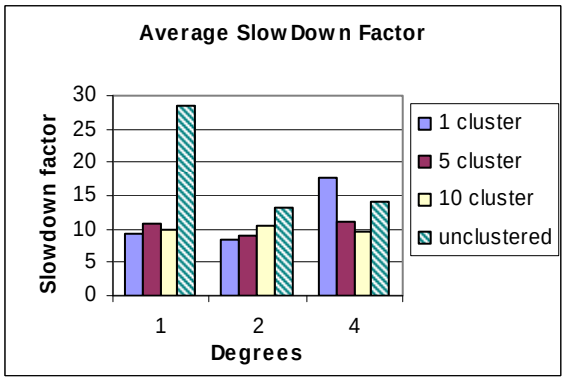

Figure 9. Average slowdown with level based clustering.

For the one square degree Montage workflow, the slowdown of the unclustered workflow is significantly larger than that of the clustered workflow. For the two and four square degree Montage workflows, the average slowdown with clustering about the same as that without clustering. Yet, the final completion time of the clustered workflows is much less than that of the unclustered ones (Figure 8), demonstrating the effectiveness of clustering tasks when both the clustered and unclustered tasks are getting similar quality of service from the resources. Within different clustering factors, there is little difference except for the 4 degree workflow where the slowdown decreases with increase in the clustering factor due to the reasons mentioned before.

4.2 Label-based clustering

In label-based clustering, we initially cluster using level-based clustering with clustering factors of 1,5, and 10; and then we collapse the clusters at levels 3 and 4 into a single cluster and that at levels 5,6, and 7 into another cluster. Thus the clustered workflows now have fewer levels than the level-based clustering only. Due to the resulting reduction in number of dependencies in the workflow, we anticipated that it would complete earlier than the workflows clustered using level-based clustering only. Figure 10 shows the workflow completion times with label and level based clustering. Each data point is the average of three clustering factors and three runs of each clustering factor. There doesn't appear to be much difference between the two clustering techniques for 1 degree workflow, but for the larger workflows the label-based clustering seems to perform better than the level-based only and the difference increases with the size of the workflow.

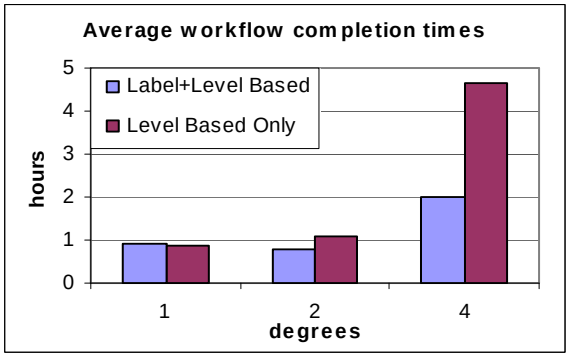

Figure 10. Comparison of label+level and level- based clustering.

The experiments described in this section were done on a shared operational execution environment (NCSA TeraGrid cluster). In such an environment the completion time of a workflow is highly dependent on the workload of the resources during the timeframe of execution of the workflow. Yet, we have tried to gain high-level insights into the performance of the various clustering techniques by repeating experiments multiple times, eliminating the execution records that differ widely from the rest of the records and then taking the average. Apart from performance, there are other factors in favor of clustering such as the reduced overhead associated with executing tasks remotely, and the reduced load that these tasks create on the common shared resources such as the cluster headnode.

5. OVERLAY COMPUTING

For the experiments described in the previous sections, the tasks or clusters were submitted to the queue of the NCSA TeraGrid cluster. Since there were multiple levels in the workflows with several tasks or clusters at each level, the effect of the queue wait times get compounded. In this section, we examine a different computing model where the user requests a certain number of processors for a certain duration from the remote resource and then uses special middleware tools to execute the workflow over the allocated processors without having to go through the remote queue again. This presents advantages for both the user and the resource owner. For the user, he/she has to go through the resource queue only once to get the acquired resources and thus the penalty of queue wait time is only incurred once (provided the workflow makespan is less than the maximum wallclock time at the resource). For resource owners, they are no longer responsible for scheduling the individual tasks in the workflow and hence the load on the resource scheduler is decreased. Moreover, the user can now actually schedule the workflow on the acquired processors in an intelligent fashion in order to minimize the completion time of the workflow. There has been a lot of research on scheduling task graphs on dedicated systems[19] that is relevant here.

One way to implement this approach is to submit a *placeholder* job into the resource queue. When this placeholder job starts execution, it allows the user to schedule tasks on the processors allocated to the placeholder by the resource scheduler using a pull- or a push-based mechanism. In fact placeholders can be submitted to multiple queues over multiple resources and when they start execution, they provide resources to the user that can be scheduled at his/her discretion. This user-level aggregation and scheduling of resources bypassing the remote resource schedulers, creating personal clusters [20], is also called overlay metacomputing [21].

One example of such placeholder technology is the *glidein* [22] feature of Condor [23]. Users can submit glidein jobs to the remote resources. When the glidein job starts execution, it starts worker daemons on the processors allocated to it and these worker nodes then report a Condor pool controlled by the user as shown in Figure 11. This Condor pool can consist of a single desktop machine owned by the user. When the glidein job is running, this pool is dynamically expanded to include worker nodes allocated to the glidein job. The user can then use standard Condor tools such as DAGMan for executing workflows on these resources. [24] reports another system called *TrellisDAG* for executing workflows using placeholders.

For the experiments in this section, we submitted Condor glidein jobs to the NCSA TeraGrid cluster requesting 8 processors. When the glidein job started execution, 8 worker nodes appeared in our Condor pool and then Condor DAGMan was used to execute the Montage workflows over these worker nodes. The duration of the requested processors was 10 minutes for the Montage 1 square degree workflow, 20 minutes for 2 square degrees and 30 minutes for 4 square degrees workflow. This was based on our observation that these workflows should not take more than this amount of time on 8 processors to complete execution. Then we measured the completion time of the workflow including the time spend by the glidein job waiting in the resource queue.

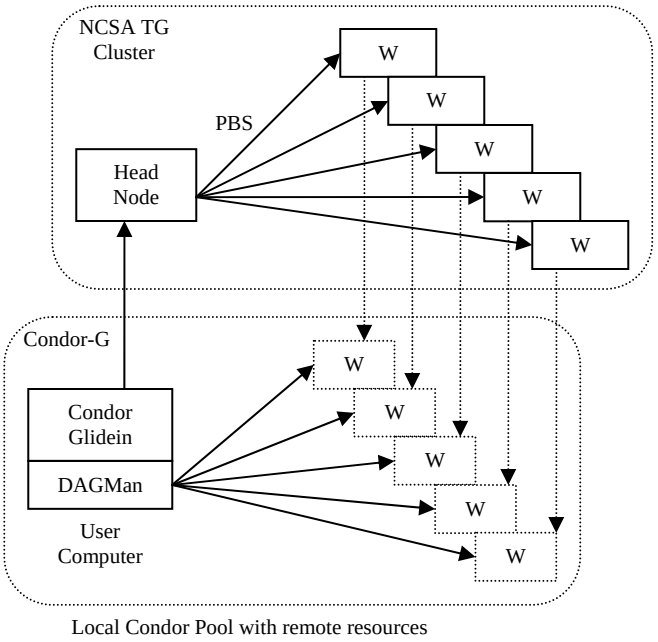

Figure 11. Adding remote resources to a local Condor pool.

Figure 12 shows how the workflow completion times using Glidein compares with the best clustering results obtained in the previous section (the label+level clustering shown in Figure 10). The figures shown are the average of 3 runs. The figure shows that overlay computing can reduce the completion time of the workflow significantly. The 4 square degree Montage workflow now completed on average in 24 minutes as compared to 816 minutes with the unclustered case (Figure 8), a reduction of 97%.

We can still use clustering on the top of overlay computing to decrease the execution overhead of the workflow. This overhead is due to the dependency management in Condor DAGMan, the time to submit a task to the Condor queue, and the time taken by Condor to find an execution host for the task and starting the task on that host. For example, when Glidein was used for the experiments shown in Figure 12, the workflow was clustered with a clustering factor of 8. This was because there were only 8 worker nodes and we wanted to have 8 clusters at each level that can execute in parallel. Once the clusters were mapped to the worker nodes, the wrapper program that implements the clusters can execute the tasks in the cluster sequentially with minimal overhead.

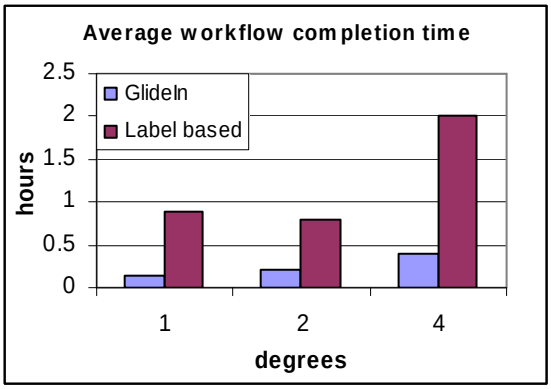

Figure 12. Comparison of Glidein and the best clustering results.

While it would seem that overlay computing is much superior to submitting tasks to the resource queue, it is not always an available option. For example, firewalls can prevent the worker nodes from joining the user Condor pool. Moreover, the resource utilization might be lower than going through the resource queue. This is due to the fact not all the allocated processors will be working for the entire duration for which the processors are allocated due to changes in parallelism at various levels of the workflow. For example, at levels 3, 4, 6, and 7 of the Montage workflow as shown in Figure 1, the parallelism of the workflow is 1. Thus at these levels only one task would be executing on one processor while the rest of the processors would be idle. In comparison, when individual tasks are submitted to the remote resource, the processors allocated to these tasks are busy the entire time the task is executing. In a previous work, we had looked at the problem of selecting the proper request shape taking into consideration the resource utilization [25].

The shape of the placeholder to request is an optimization problem in itself. The placeholder can be thought as a malleable job. For example, the 4 square degrees Montage workflow could be executed using 2 processors in under 2 hours, using 4 processors in under 1 hour, using 8 processors in under 30 minutes and so on. However, while 2 processors might be available immediately, there might be wait time involved in getting 8 processors. Conversely, 8 or more processors might be available immediately for the next 30 minutes but requesting a larger duration would involve wait time. Thus we need to decide which of these to use in order to minimize the sum of the queue wait time of the request and the execution time of the workflow. However, this is a difficult question to answer because the queue wait times of the requests can not be predicted accurately in advance. There are queue prediction services installed on the TeraGrid cluster [26] that try to predict the queue wait time of a particular request. However, there is usually a trade-off between the accuracy of the estimate and its conservativeness. In the future, we plan to investigate using the queue wait time predictions for minimizing the workflow completion times.

Another approach for selecting the placeholder shape is to examine the current resource availability of the cluster. The NCSA TeraGrid cluster allows users to examine the currently available resources using the *showbf* (show backfill window) command [27]. The output of a showbf command is shown in Figure 13. It shows that 8 processors (called *Tasks*) are available for 38 minutes while 6 or less processors are available for little more than one hour. Thus in this situation requesting 8 processors for 30 minutes would not involve any queue wait time and the request would be able to start immediately. However, this is only a guideline and the resource scheduler might enforce additional constraints such as limiting the total number of jobs concurrently running for a particular user, etc.

```
$$> showbf
Partition   Tasks  Nodes  StartOffset  Duration
---------   -----  -----  -----------  --------
ALL             8      4     00:00:00   0:38:52
ALL             6      3     00:00:00   1:01:16
ALL             2      1     00:00:00   1:09:45
```

Figure 13. Output of *showbf* command.

Executing a workflow using a placeholder as explained here is logically equivalent to clustering the whole workflow into a single cluster. This is because we are submitting a single task (glidein) to the remote resource queue and this task is used to execute the entire workflow. Since we are executing a workflow instead of a set of independent tasks, we have to use Condor DAGMan instead of the simple wrapper program that we had been using earlier. Thus we have covered both the extreme ends of the spectrum. At one end, the workflow is unclustered and each task can be thought to be a single cluster. At another end, the entire workflow is a single cluster. In between there are various possibilities as explored by the various label and level based clustering techniques as discussed in the previous section.

6. RELATED WORK

There has been a lot of work done on clustering task graphs on multiprocessing systems in order to minimize completion time [28]. The tradeoff there is between the concurrency and communication. When tasks that can execute in parallel are clustered, they are constrained to execute serially on the same processor. Thus there is a loss of concurrency. However, the communication cost between the clustered tasks is zero. There are marked differences between this clustering and our clustering. In our implementation, the clusters need to be convex and the tradeoff is between concurrency and queue wait times. Clustering does not affects the communication costs since the tasks (clustered or otherwise) communicate using files and thus via NFS. Even the loss of concurrency due to serial execution in our case can be overcome by executing each cluster using multiple processors (*mpiexec*) in order to exploit the parallelism between the clustered tasks. However, in the general case each cluster would require a dependency manager to execute the tasks in the cluster.

Overlay metacomputing [20, 21] has been recogonized as a means of creating a user level aggregation of resources from multiple providers. [24] describes a system for scheduling and executing DAGs on such overlays. However, the functionality provided is similar to that of Condor glidein and DAGMan that were used by us for our experiments. [20] describes another system called MyCluster, currently installed on the TeraGrid system that creates a user level personal cluster by submitting glidein jobs to the underlying resource queues.

The request selection problem in order to minimize the completion time for moldable jobs have been studied earlier in parallel computing [29-31]. [31] introduces an application scheduler that chooses on the behalf of the user, which request to submit for a particular job. It also uses a simulation in order to determine the

best request. But the simulation is done by the user instead of the scheduler. Their work is mainly focused on the selection between the various possible requests. In [25] we extend this work to workflows and cast a workflow as a moldable job. In [25] we consider the resource utilization of the resulting schedule as an explicit metric to be optimized. When the user-level overlay can be formed from multiple resource requests instead of a single one, [32] describes a system for intelligent selection of resources constituting such overlays in order to minimize the allocation cost and the completion time of the workflows. In this paper, we go beyond simulations and use a real world application over an operational Grid infrastructure in order to showcase the benefits of clustering for workflows.

Another related aspect is efficient scheduling and execution of the workflows tasks over the allocated resources when using overlay computing. Due to the distributed nature of the resources, the large number of tasks in the workflow and the dependencies between the tasks, there can be a non-negligible overhead involved in executing the workflow over the acquired resources [33, 34]. In [33], we have done a detailed analysis of the execution overhead when using the Condor system. We have used the insights gathered from that study for efficient execution of the Montage workflows as described in this paper.

7. CONCLUSION AND FUTURE WORK

We have shown that task clustering and overlay computing techniques can be very beneficial when scheduling large-scale fine-computational-granularity workflows onto the national cyberinfrastructure resources. We presented different clustering techniques incorporated into the Pegasus workflow mapping system and shown the results of scheduling and running an astronomy application on the TeraGrid. For this application, we were able to reduce the overall workflow runtime by 97%. We can expect to see reduced benefits of clustering when workflows are composed of longer duration tasks.

In the past year, a lot of work has been done in predicting queue wait times based on statistical models for the TeraGrid sites [35]. We plan to interface Pegasus with these queue prediction services, in order to automatically determine the clustering granularity for level-based clustering. As our experiments indicate, clustering is good for running short duration jobs. It is important that the clustered job does not become too large with a long requested wall clock time that results in the job waiting in the queue for a long period. Interfacing with queue prediction services will allow us to optimize the clustering so that the queue wait time is minimized.

Currently, in the case of label-based clustering we can only execute jobs sequentially. However, this is inefficient for large clusters as the loss in parallelism might offset any gain in performance due to clustering. We propose to investigate the possibility of using a dependency manager such as Condor DAGMan [18] in order to execute tasks within a cluster on multiple nodes.

ACKNOWLEDGEMENTS

This work was supported by NSF under OCI- 0722019. We thank TeraGrid for the use of their resources.

REFERENCES

[1] D. S. Katz, N. Anagnostou, G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, M. 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. T. Yang, and H. Zima, Eds.: American Scientific Publishers, 2006.

[2] A. Lathers, M.-H. Su, A. Kulungowski, A. W. Lin, G. Mehta, S. T. Peltier, E. Deelman, and M. H. Ellisman, "Enabling parallel scientific applications with workflow tools," presented at Challenges of Large Applications in Distributed Environments, 2006 IEEE, 2006.

[3] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, "Workflow Management in GriPhyN," in *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds.: Springer, 2003.

[4] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, and L. Zhao, "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example," presented at Second IEEE International Conference on e-Science and Grid Computing, 2006.

[5] "The Open Science Grid Consortium," http://www.opensciencegrid.org.

[6] C. Catlett, "The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility," presented at Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002, 2002.

[7] R. L. Henderson, "Job Scheduling Under the Portable Batch System " in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing* Springer-Verlag, 1995 pp. 279-294

[8] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," presented at Distributed Computing Systems, 1988., 8th International Conference on, 1988.

[9] A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters, "How are Real Grids Used? The Analysis of Four Grid Traces and its Implications," presented at 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, 2006.

[10] E. Deelman, G. Mehta, G. Singh, M.-H. Su, and K. Vahi, "Pegasus: Mapping Large-Scale Workflows to Distributed Resources," in *Workflows for e-Science: Scientific Workflows for Grids*, I. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds.: Springer, 2007.

[11] "Montage Project." http://montage.ipac.caltech.edu.

[12] G. B. Berriman, E. Deelman, J. Good, J. Jacob, D. S. Katz, C. Kesselman, A. Laity, T. A. Prince, G. Singh, and M.-H. Su, "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand," presented at SPIE Conference 5487: Astronomical Telescopes, 2004.

[13] "Montage," in *http://montage.ipac.caltech.edu*.

[14] "Montage Components." http://montage.ipac.caltech.edu/docs/components.html.

[15] 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*, vol. 13, pp. 219-237, 2005.

[16] Pegasus, "http://pegasus.isi.edu."

[17] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: a computation management agent for multi-institutional grids," presented at High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on, 2001.

[18] "Condor DAGMan." http://www.cs.wisc.edu/condor/dagman.

[19] H. Topcuouglu, S. Hariri, and M.-y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13(3), pp. 260-274, 2002.

[20] E. Walker, J. P. Gardner, V. Litvin, and E. L. Turner, "Creating Personal Adaptive Clusters for Managing Scientific Jobs in a Distributed Computing Environment," presented at Workshop on Challenges of Large Applications in Distributed Environments (CLADE), 2006.

[21] C. Pinchak, P. Lu, and M. Goldenberg, "Practical Heterogeneous Placeholder Scheduling in Overlay Metacomputers: Early Experiences," in *Job Scheduling Strategies for Parallel Processing*, D. G. F. a. L. R. a. U. Schwiegelshohn, Ed.: Springer Verlag, 2002, pp. 205--228.

[22] Condor_Glidein, "http://www.cs.wisc.edu/condor/glidein."

[23] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, pp. 323-356, 2005.

[24] M. Goldenberg, P. Lu, and J. Schaeffer, "TrellisDAG: A System for Structured DAG Scheduling," in *Job Scheduling Strategies for Parallel Processing*, D. G. F. a. L. R. a. U. Schwiegelshohn, Ed.: Springer Verlag, 2003, pp. 21--43.

[25] G. Singh, C. Kesselman, and E. Deelman, "Performance Impact of Resource Provisioning on Workflows," University of Southern California available at http://www.cs.usc.edu/Research/TechReports/05-850.pdf 05-850, 2005.

[26] D. Nurmi, R. Wolski, J. Brevik, and G. Obertelli, "QBETS: Batch Queue Prediction System," presented at TeraGrid Conference, Madison, 2007, available at http://www.teragrid.org/events/teragrid07/archive/presentations/wednesday/QBETS.pdf.

[27] ShowBF, "Maui User Manual, available at http://www.clusterresources.com/products/maui/docs/commands/showbf.shtml."

[28] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Survey*, vol. 31, pp. 406-471, 1999.

[29] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling " in *Proceedings of the Job Scheduling Strategies for Parallel Processing* Springer-Verlag, 1997 pp. 1-34

[30] A. B. Downey, "Using Queue Time Predictions for Processor Allocation " in *Proceedings of the Job Scheduling Strategies for Parallel Processing* Springer-Verlag, 1997 pp. 35-57

[31] W. Cirne and F. Berman, "Using Moldability to Improve the Performance of Supercomputer Jobs," *Journal of Parallel and Distributed Computing*, vol. 62, pp. 1571-1601, 2002.

[32] G. Singh, C. Kesselman, and E. Deelman, "A Provisioning Model and its Comparison with Best-Effort for Performance-Cost Optimization in Grids," in *Proceedings of the 16th International Symposium on High Performance Distributed Computing (HPDC)*. Monterey, California, USA: ACM Press, 2007, pp. 117-126.

[33] G. Singh, C. Kesselman, and E. Deelman, "Optimizing Grid-Based Workflow Execution," *Journal of Grid Computing*, vol. 3(3-4), pp. 201-219, 2005.

[34] F. Nerieri, R. Prodan, T. Fahringer, and H.-L. Truong, "Overhead Analysis of Grid Workflow Applications," presented at 7th IEEE/ACM International Conference on Grid Computing, 2006.

[35] J. Brevik, D. Nurmi, and R. Wolski, "Predicting Bounds on Queueing Delay in Space-Shared Computing Environments," presented at IEEE International Symposium on Workload Characterization, 2006.