Adaptive Workflow Processing and Execution in Pegasus

Kevin Lee[1], Norman W. Paton[1], Rizos Sakellariou[1],
Ewa Deelman[2], Alvaro A. A. Fernandes[1], Gaurang Mehta[2]

[1]School of Computer Science
University of Manchester
Oxford Road, Manchester M13 9PL, U.K
{klee, norm, rizos, alvaro}@cs.man.ac.uk

[2]University of Southern California
Information Sciences Institute
Marina Del Ray, CA 90292, USA
{deelman, gmehta}@isi.edu

Abstract

Workflows are widely used in applications that require coordinated use of computational resources. Workflow definition languages typically abstract over some aspects of the way in which a workflow is to be executed, such as the level of parallelism to be used or the physical resources to be deployed. As a result, a workflow management system has responsibility for establishing how best to execute a workflow given the available resources. The Pegasus workflow management system compiles abstract workflows into concrete execution plans, and has been widely used in large-scale e-Science applications. This paper describes an extension to Pegasus whereby resource allocation decisions are revised during workflow evaluation, in the light of feedback on the performance of jobs at runtime. The contributions of this paper include: (i) a description of how adaptive processing has been retrofitted to an existing workflow management system; (ii) a scheduling algorithm that allocates resources based on runtime performance; and (iii) an experimental evaluation of the resulting infrastructure using grid middleware over clusters.

1 Introduction

A number of workflow environments have been developed in recent years to provide support for the specification and execution of scientific workflows. We distinguish scientific workflows (as supported, for example, by Pegasus [9], Askalon [11], Taverna [18], Kepler [2] and Triana [19]), being typically compute and/or data intensive, as opposed to business workflows, which are beyond the scope of this paper. Workflow languages are used to provide a high-level characterization of the pattern of activities that need to be carried out to support a user task. Workflows written in such languages typically leave open a number of decisions as to how a workflow is enacted, such as where the workflow is to be run, what level of parallelism is to be used and what resources are to be made available to the workflow. As a result, a collection of decisions must be made before a workflow can be enacted, for example by a compilation process that translates a workflow from an abstract form into a more concrete representation that resolves various of the details as to how the workflow is to make use of available resources.

Most existing workflow systems provide static approaches for mapping (e.g. [5, 21]) on the basis of information that provides a snapshot of the state of the computational environment. Such static decision making involves the risk that decisions may be made on the basis of information about resource performance and availability that quickly becomes outdated. As a result, benefits may result either from incremental compilation, whereby resource allocation decisions are made for part of a workflow at a time (e.g. [9]), or by dynamically revising compilation decisions that gave rise to a concrete workflow while it is executing (e.g. [14, 10, 16, 22]). In principle, any decision that was made statically during workflow compilation can be revisited at runtime (e.g. [17]).

Proposals that describe adaptive approaches to mapping (e.g., [14]) are often quite intrusive, in that the adaptive behaviour of their engine exercises fine-grained control over the workflow engine, implying that significant effort may be require to incorporate such capabilities into existing mainstream workflow systems. In contrast, the work described in this paper implements adaptivity as a separate module which is loosely-coupled with an existing workflow system.

This paper describes an approach to adaptive resource allocation and scheduling in the Pegasus workflow management system [9]. Pegasus already accommodates uncertainty about the runtime environment by incremental compilation, which both defers certain decisions as to how workflow activities are mapped to resources and forms the basis for fault tolerance, whereby a workflow partition, which is the unit of incrementality, can be retried if it fails. In line with [14] our adaptive system is purely reactive in that it monitors information and reacts to it. Thus, the emphasis is on adaptations on the basis of specific observable behaviour rather than mechanisms to predict what future behaviour is going to be. Our objectives in this work have been: (i) to dynamically adjust resource allocation decisions in the light of run-time feedback on the performance of the clusters onto which workflows are being compiled; and (ii) to obtain that dynamic behavior through minimal intervention into the existing Pegasus infrastructure. As a case study for the evaluation of our adaptive system we consider resource allocation on clusters that might be used by several users at the same time. This allows us to introduce adaptivity into an environment whose performance is not well known in advance, and in which there is limited control over the execution of individual jobs; studies that focusing on the evaluation of scheduling heuristics usually require more information about the environment than is assumed here [20, 16, 22, 23]. Yet, without using sophisticated heuristics, our adaptive engine yields demonstrable benefits.

The remainder of this paper is structured as follows. Section 2 provides the technical context for this work by describing the Pegasus workflow management system. Section 3 details both what adaptations are carried out and how these have been integrated with the Pegasus infrastructure. Section 4 describes the results of experiments conducted using both synthetic and real-world scientific workflows. Section 5 draws some overall conclusions.

2 Technical Context

The Pegasus Workflow Management System (Figure 1) consists of the Pegasus workflow mapper [9] and the DAGMan [13] workflow executor for Condor. The Pegasus mapper takes high-level descriptions of complex applications structured as workflows, automatically maps them to available cyberinfrastructure resources, and submits them to DAGMan for execution. Pegasus has been used in a wide range of applications including earthquake science and astronomy.

The *workflow mapping engine* is a compiler that translates between the high-level specifications and the underlying execution system and optimizes the executables based on the target architecture. The mapping includes finding the appropriate software and computational resources where the execution can take place, as well as finding copies of the data indicated in the workflow instance. The mapping process can also involve workflow restructuring geared towards optimizing the overall workflow performance as well as workflow transformation geared towards data management and provenance information generation. The result of the mapping process is an executable workflow, which can be executed by a

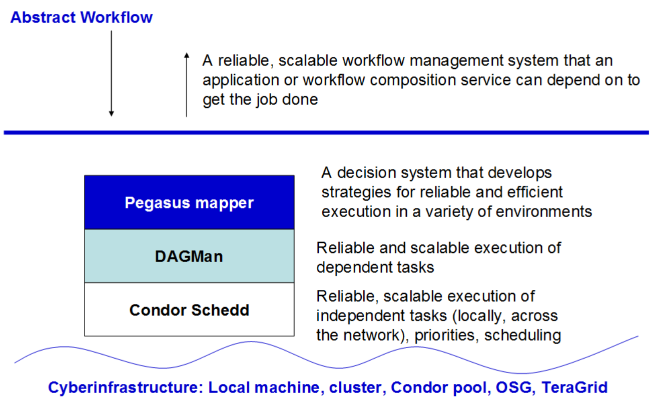
Abstract Workflow
A reliable, scalable workflow management system that an application or workflow composition service can depend on to get the job done
Pegasus mapper
A decision system that develops strategies for reliable and efficient execution in a variety of environments
DAGMan
Reliable and scalable execution of dependent tasks
Condor Schedd
Reliable, scalable execution of independent tasks (locally, across the network), priorities, scheduling
Cyberinfrastructure: Local machine, cluster, Condor pool, OSG, TeraGrid

Figure 1. Layered Architecture of the Pegasus Workflow Management System.

workflow engine that follows the dependencies defined in the workflow and executes the activities defined in the workflow tasks. DAGMan, our workflow engine relies on the resources (compute, storage, and network) defined in the workflow to perform the necessary actions. As part of the execution, data is generated along with associated metadata.

Mapping the workflow instance to an executable form involves finding the resources that are available and can perform the computations, the data that is used in the workflow, and the necessary software. We assume that data may be replicated in the environment and that users publish their data products into some data registry. Pegasus uses the logical filenames referenced in the workflow to query a data registry service, such as the Globus Replica Location Service (RLS) [6], to locate the replicas of the required data. Given the set of logical filenames, RLS returns a corresponding set of physical file locations. Optionally, Pegasus also adds tasks to the workflow to register the final and intermediate workflow data products into the registry. In this way, new data products can be easily discovered by the user, the community, or another workflow. In order to be able to find the location of the logical application component names (transformations) defined in the workflow instance, Pegasus queries the Transformation Catalog (TC) [8] and obtains the physical locations of the transformations (on possibly several systems) and the environment variables and libraries necessary for the proper execution of the software. Pegasus also supports staging of statically linked executables on demand. In that case, the executables are treated as input data for the corresponding workflow tasks. The executables are transferred to the remote grid sites along with other input data required.

Pegasus queries cyberinfrastructure monitoring services (e.g., the Globus Monitoring and Discovery Service (MDS) [12]) to find the available resources and their characteristics (machine load, scheduler queue length, available

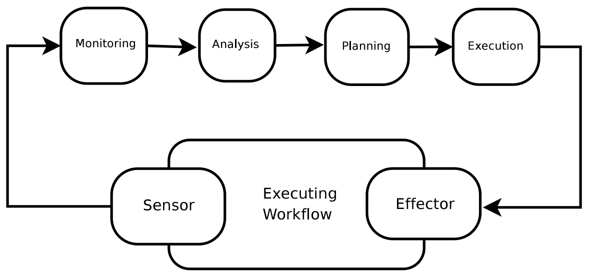

Figure 2. Pegasus Adaptive Support

disk space, and others). This information is combined with information from the Transformation Catalog to make scheduling decisions. Schedulers are one of the pluggable components of Pegasus. Up to now Pegasus included four different scheduling algorithms: random, round-robin, min-min [5], and HEFT [20]. In this work, we designed and incorporated a new scheduler into Pegasus. As opposed to the static nature of the existing four (and the large body of relevant work in the literature, e.g., [23]), the key feature of our new algorithm is that it takes into account runtime information.

Pegasus also uses information services to find the location of the data movement services (e.g., GridFTP [1] or SRB [4]) that can perform wide-area data transfers, job managers [7] that can schedule jobs on the remote sites, storage locations, where data can be pre-staged, shared execution directories, site-wide environment variables, etc. This information is necessary to produce the executable workflow that describes the necessary data movement, computation and catalog updates. Registries of code and data as well as information services allow Pegasus to provide a level of abstraction to the user and give the freedom to automatically optimize workflow execution.

3 Adaptive Pegasus

As stated in Section 1, the focus of this paper is on dynamically adjusting resource allocation decisions in response to feedback on the performance of workflow execution. The adaptive strategy used is structured around the *MAPE* functional decomposition [15] which partitions adaptive functionality into four areas, *Monitoring*, *Analysis*, *Planning* and *Execution*. The MAPE functional decomposition is a useful framework for systematic development of adaptive systems, and can be applied in a wide range of applications, including different forms to workflow adaptation [17]. The use of MAPE to structure the adaptive strategies in this paper is illustrated in Figure 2, which shows how it is retrofitted with minimal intervention to a Pegasus-planned executing workflow.

In the adaptation strategy described in this paper, an executing workflow instance is monitored for the relevant events at the assigned resources. These events are constantly analysed for patterns, which may lead to planning. Planning updates the information available to Pegasus, and reruns Pegasus on the current workflow. The revised plan for the work that remains to be done is compared with the current plan, and the new plan is adopted if it is predicted to give an improved overall response time. Changes to the workflow execution proposed by Planning are implemented in an execution step that removes and replaces the executing workflow. The following paragraphs discuss the components in Figure 2 in more detail.

Monitoring: To monitor the progress of an executing workflow, *job queue*, *execute* and *termination* events are tracked. These, respectively, indicate when Condor submits a task to the remote scheduler, when the remote scheduler indicates that the task has started to execute, and when the remote scheduler indicates that the task has completed. These are sensed using a *LogSensor* that polls for new entries in the DAGMan log file every 100 milliseconds. The DAGMan log file records all events about the execution of a workflow and its progress. Each entry of the log file is parsed to determine if it contains an event of interest. These events are passed to Analysis.

Analysis: The role of the analysis step is to establish whether the workflow is performing according to expectations when it was compiled. If expectations are not being met, then it may be possible to improve on the plan that is being pursued. To support the concise and declarative description of patterns in the monitoring data, the CQL continuous query language [3] is used to group and analyse the events produced by monitoring. The CQL queries that implement the analysis are given in Figure 3.

The queries look for a sustained substantial increase or decrease in batch queue (waiting) times per site compared to the job batch queue predictions created by the scheduler. If there is an output from this analysis, the planner is notified. In addition to determining if adaptations may be necessary, *Analysis* also generates average queue times for each available site for use by the scheduling algorithm. Queue times are derived using relevant event information from Monitoring.

Planning: When analysis detects a sustained change in batch queue times for a site, re-scheduling may need to be performed. To examine this, the Pegasus planner is called to propose an alternative schedule taking into account recent queue times.

To ensure that jobs are not unnecessarily repeated, the replica catalogues used by Pegasus to share results within and between workflows are updated with results already produced by the workflow. This is because each job in a concrete workflow outputs its results as intermediate data in the form of a file. The relevant folders on the execution sites are scanned for intermediate results, which are added to the replica cata-

```
Input Streams:
events :
  int timestamp, char event, char job
assignments :
  char job, char site, int estimate

Queries:
jobqueued :
  select timestamp, job from events
  where event = 'ULOG_SUBMIT';
  register stream jobqueued
    (int timestamp, char job);

jobexecuted :
  select timestamp, job from events
  where event = 'ULOG_EXECUTE';
  register stream jobexecuted
    (int timestamp, char job);

queuedtime :
  select execute.timestamp − queued.timestamp,
    execute.job
  from jobqueued as queued, jobexecute as execute
  where queued.job = execute.job;
  register stream queuedtime
    (int queuetime, char job);

queuetimeandestimate :
  select queue.timestamp, queue.job, assignment.site,
    assignments.estimate
  from queuetime as queue,
    jobassignments as assignments
  where queue.job = assignments.job;
  register stream queuetimeandestimate
    (int timestamp, (char job, char site);

Analysis:
  select "LongQueue", site
  from queuetimeandestimate[Rows 3]
  where AVG(time − estimate) > threshold;
  select "ShortQueue", site
  from queuetimeandestimate[Rows 3]
  where AVG(estimate − time) > threshold;
```

Figure 3. Filtering monitoring events in CQL

logue.

As discussed in Section 2, Pegasus currently has four different schedulers which it uses to assign jobs to resources. However, these were designed to schedule statically using limited (or statically estimated) information about the performance characteristics of execution resources. To enable adaptive behavior, a scheduling algorithm is needed that takes account of information gleaned by Monitoring. To this end, we implemented a new scheduler which uses data collected about the average queue times of each available site to decide where to schedule each job in the workflow. Figure 4 shows this scheduling algorithm that enables adaptivity.

The scheduler depends on the presence of historic data containing the average queue times for each available site. This is generated by *Analysis*; when no prior data on average site queue times is available a default value of 0 is used.

```
Input:
   Workflow $W$
   List of Sites $S$
   List of Average Queue Times $SQ$

1. Calculate $PS_s$ the proportion of the workflow each site $s$
   should process.
   for Site $s \in S$
      $PS_s = (1/SQ_s)/\sum_{i \in S}(1/SQ_i)$

2. Calculate $Num_s$ the number of jobs each site $s$ should process.
   for Site $s \in S$
      $Num_s = PS_s * size(W)$

3. Create $AS$ a queue of assignable sites.
   for Site $s \in S$
      for Int $i = 1$ to $Num_s$
         $AS.push\_back(s)$

4. Randomise the list of assignable sites.
   $AS.randomise()$

5. Create $A_j$, the job to site assignment list.
   for Job $j \in W$
      $A_j = AS.pop\_front()$
```

Figure 4. Adaptive Scheduling Algorithm

The scheduler allocates work to each site in inverse relation to the average queue time since the start of the execution of the workflow. It is as follows: Step *1* calculates the proportion of a workflow (in number of tasks) that should be assigned to each site, based on average batch queue times. Step *2* calculates the number of jobs each site should process, by multiplying the number of jobs by the proportion each site should be assigned. Steps *3* and *4* create a randomised list of sites based on the number of jobs each site should be assigned from Step *2*. Step *5* creates the final job-to-site assignment list.

Not every new schedule proposed by the scheduler is deployed; new schedules are compared with the existing executing schedule to see if they are predicted to improve on the current plan. The cost of adaptation is also taken into account when deciding whether or not to deploy a new schedule. If it is decided to deploy it, the next component, execution, is called.

In addition to returning a list of job assignments to sites, the scheduler also generates a list of predicted batch queue times for each job on each site. These predictions are later used by *Analysis* to detect substantial deviations from actual running times. The predicted batch queue time is the average queue time for the site to which the job has been assigned. These are made available to *Analysis* in the form of the *assignments*, input stream in Figure 3.

Execution: At the stage that execution is called, there is a currently executing workflow. Execution stops the executing

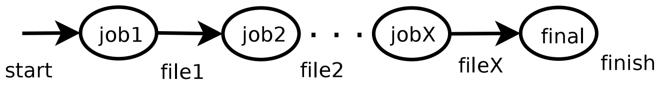

Figure 5. A Linear Workflow

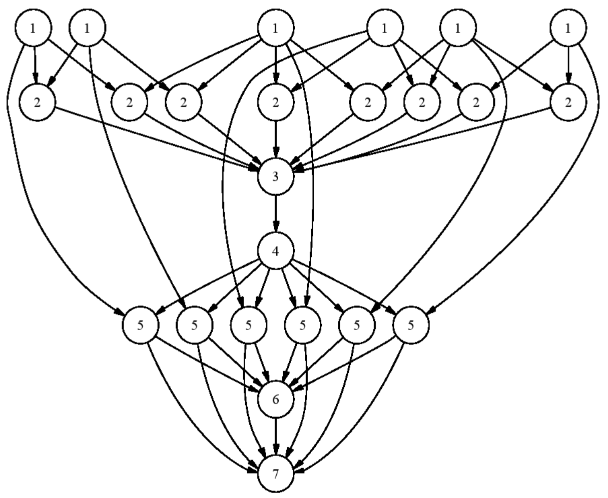

Figure 6. A Simple Montage Workflow [9]

workflow and deploys the new one using Pegasus commands.

4 Experimental Evaluation

4.1 Experiment Setup

The aim of the experimental evaluation is to explore the effect of the adaptive approach on response time in a range of scenarios. The experiments use two abstract workflow styles. The first type is a *linear workflow*, whose general form is illustrated in Figure 5. This is simply a DAG were each subsequent task is dependent on the file created by the previous task, and may contain any number of tasks. With these dependencies present, the tasks in the workflow will execute in series. In our experiments we considered an instance with 50 tasks. The second workflow type is that of a *Montage workflow*, which creates a large mosaic image from many smaller astronomical images [9]. These can be of varying sizes depending on the size of the area of sky of the mosaic. A simple Montage workflow is illustrated in Figure 6. The numbers represent the level of each task in the overall workflow. This corresponds to the size used in our experiments (25 tasks, equivalent to a 0.2 degree area).

In order to run the workflows, two clusters were used, which we designate *Cluster 1* and *Cluster 2*. The use of a modest number of clusters does not change the nature or value of the approach, as this level of resource availability is common. Cluster 1 has as submission site a 2.4Ghz Xeon with 2GB of RAM, and 8 worker nodes each with a 2.4Ghz Xeon with 2GB of RAM connected together by Gigabit Ethernet. Cluster 2 has as submission site a 2Ghz dual core Opteron with 4GB RAM, and 112 worker nodes each with a dual core 1Ghz P3 with 4GB RAM connected together by 100 Megabit Ethernet. All jobs are setup and submitted from the Cluster 1 submission site.

For each of the experiments, we submitted two workflows in parallel, a non-adaptive one and an adaptive one. The non-adaptive workflow uses simple round-robin scheduling, whereas the adaptive workflow uses the adaptive scheduling mechanism described in Section 3.

It should be noted that the resources, as detailed above, are not dedicated to the experiments in this paper, so they may be influenced by submissions from other users. However, in order to test the effect of the adaptivity strategy better, in some experiments, we also introduced additional (controlled) loads to the clusters. Thus, we group the results of the experiments according to the model for additional external load considered:

- **No Additional External Load:** For the purposes of the experiment, no external load is applied; the clusters are still, however, subject to third-party external load.
- **Constant Additional External Load:** For the duration of the experiment, additional linear workflows are submitted to a cluster. This has the effect of providing a constant additional external load above any third-party load on the clusters.
- **Temporary Additional External Load:** For a period of time specified in each experiment, linear workflows of a specified size and number are submitted to a cluster, creating a temporary increase in load.

At the end of each experiment, the log files were parsed to produce the results. In order to illustrate long waiting times in the queue for individual jobs of a workflow, the graphs presented plot both the queue time and execution time for each job separately; even though this distinction may not be immediately obvious in the case of experiments using workflows with a relatively large number of tasks, the graphs still indicate trends. The vertical axis of the graphs shows wall-clock time in the form *hours:minutes:seconds*. For each experiment, graphs for non-adaptive and adaptive workflow execution are plotted side-by-side to allow comparison.

4.2 No External Load

Experiment 1: The objective of this experiment is to compare the adaptive and non-adaptive approaches where no additional external load has been submitted to the clusters and there is no historical information on cluster performance.

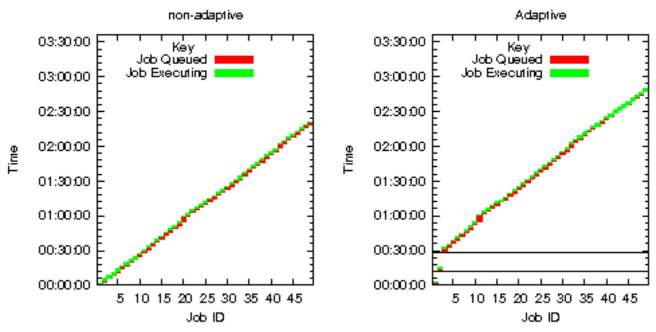

Figure 7. Results of Experiment 1

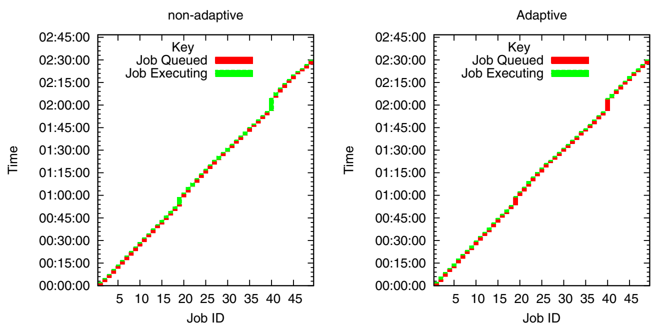

Figure 8. Results of Experiment 2

Adaptive and non-adaptive linear workflows (50 tasks each) are submitted in parallel, with access to Clusters 1 and 2, with no additional external load. The results are presented in Figure 7, which shows that the adaptive workflow performs less well than non-adaptive one. This is because it has to build up knowledge about the execution environment that can form the basis for informed adaptations. When enough knowledge has been gained, an adaptation is performed, which is visible on the graph as a gap in the linear workflow execution. The point in time when, as a result of an adaptation, a new schedule is applied is denoted with an horizontal line in the graph. The adaptive workflow adapts twice. The gains that result from adaptation are too modest to make up for the cost of adapting. This is because the clusters are performing similarly and consistently across the execution, and thus the original non-adaptive schedule is efficient.

Experiment 2: The objective of this experiment is to compare the adaptive and non-adaptive approaches where no additional external load has been submitted to the clusters and historical information on cluster performance is available.

The same workflows are submitted as in Experiment 1. With prior knowledge about the environment (from Experiment 1), the results are as shown in Figure 8. No adaptations are carried out for this run, and the adaptive and non-adaptive workflows perform similarly.

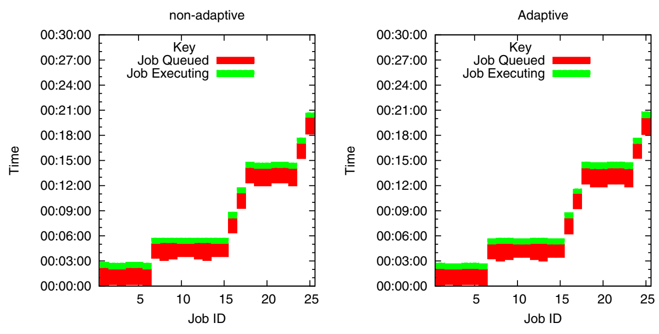

Figure 9. Results of Experiment 3

Experiment 3: The objective of this experiment is to compare the adaptive and non-adaptive approaches with no additional external load in the presence of historical information with a more complex workflow.

Adaptive and non-adaptive Montage workflows are submitted in parallel, with access to Clusters 1 and 2, with no additional external load. Prior knowledge is available about the environment (from Experiment 1). The results are shown in Figure 9, which indicates that the clusters act as expected and no adaptations are carried out for this run. Where tasks are run in parallel in Figure 9, this reflects the inherent parallelism of Montage (see the graph in Figure 6).

Summary: Once the adaptive infrastructure has been primed with current information about the environment, it correctly refrains from performing adaptations where none are required. The remainder of the experiments assume the availability of historical information about the clusters.

4.3 Constant External Load

Experiment 4: The objective of this experiment is to compare the adaptive and non-adaptive approaches with additional external load on the smaller cluster.

The same linear workflows are submitted as in Experiment 1, with additional constant external load supplied by the submission of 50 linear workflows (100 tasks each) to Cluster 1 at the start. The results are presented in Figure 10, which shows that the adaptive workflow changes its schedule early in the workflow execution, leading to a significant improvement in response time of the adaptive workflow compared to the non-adaptive workflow. The adaptive response time is 17% less than that in the non-adaptive case.

Experiment 5: The objective of this experiment is to compare adaptive and non-adaptive approaches with constant external load on a small cluster with a complex workflow.

Adaptive and non-adaptive Montage workflows are submitted in parallel to Clusters 1 2, with additional constant external load supplied by submitting 50 linear (100 task each)

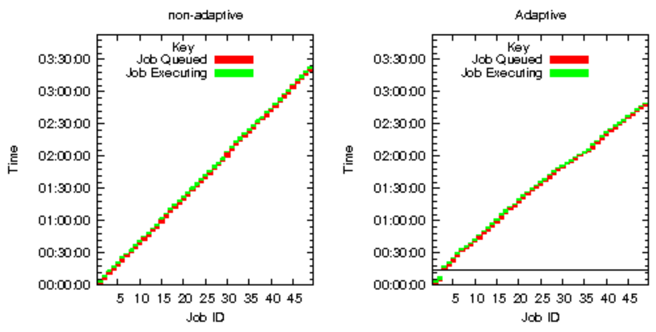

Figure 10. Results of Experiment 4

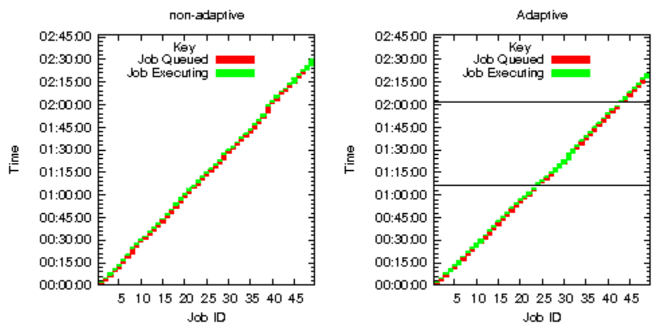

Figure 12. Results of Experiment 6

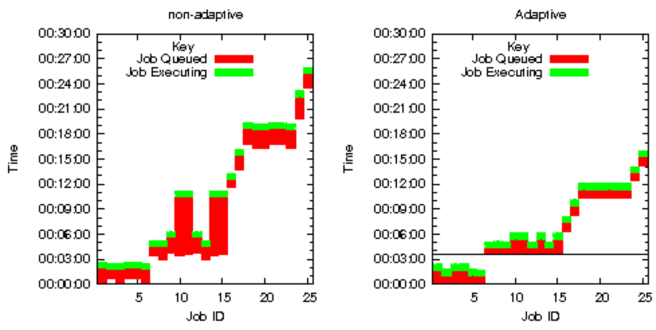

Figure 11. Results of Experiment 5

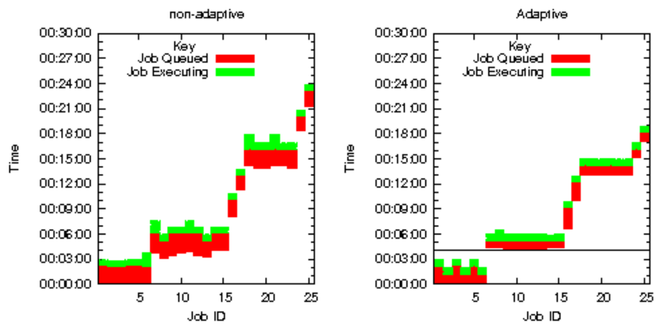

Figure 13. Results of Experiment 7

workflows to Cluster 1 at the start. The results are presented in Figure 11, which shows that the adaptive workflow changed the schedule early on in the workflow execution, leading to a significant improvement in performance when compared to the non-adaptive workflow. By moving work away from the heavily loaded Cluster 1, long queue times have been avoided, especially for the jobs with *Job Id 10*, *11*, *14* and *15*. The adaptive response time is 38% less than that in the non-adaptive case.

Summary: The constant external load is handled well by the adaptive scheduling scheme; few adaptations are required, but these provide lasting benefits, and significant response time improvements are observed.

4.4 Temporary External Load

Experiment 6: The objective of this experiment is to compare adaptive and non-adaptive approaches with temporary external load on a small cluster with a linear workflow.

The same workflows are used as in Experiment 1, with a temporary external load supplied by submitting 50 linear (10 tasks each) workflows to Cluster 1 at 60 minutes into the experiment. The results of the experiment are shown in Figure 12, in which one adaptation is performed just after 60 minutes and another when the temporary workflows complete after 120 minutes. The adaptation has reduced average queue times during the time of additional load, by moving jobs away from the heavily loaded cluster. The adaptive response time is 7% less than that in the non-adaptive case.

Experiment 7: The objective of the experiment is to compare adaptive and non-adaptive approaches with temporary external load on a small cluster with a complex workflow.

Adaptive and non-adaptive Montage workflows are submitted in parallel, with access to Clusters 1 and 2, with temporary external load supplied by submitting 50 linear (10 task) workflows to Cluster 1 at 10 minutes into the experiment. The results of the experiment are shown in Figure 13. The results show that an adaptation is performed only once, after 3 minutes. The adaptive workflows jobs are then typically subject to shorter queue times than the non-adaptive one. Even after the temporary workflows are complete no more adaptations are performed due to the jobs performing well on Cluster 2 (which has a generally lower queue time when neither cluster is loaded). The adaptive response time is 21% less than that in the non-adaptive case.

Summary: A temporary external load impedes the progress of a static workflow less than one that is present all the time, so the potential improvements available from the adaptive techniques are reduced compared with the constant external load case. However, adaptation takes place when the temporary external load is introduced, and in one case when it is removed, providing significantly reduced response times.

5 Conclusions

We have presented an approach to adaptive workflow processing that: (i) adds adaptive scheduling to an existing workflow infrastructure with minimal intrusion; (ii) illustrates the use of the MAPE functional decomposition from the autonomic computing community in a new setting, including the use of stream queries for identifying patterns of interest in monitoring events; and (iii) demonstrates significant performance improvements in experiments involving different forms of imbalance and workflows, even though the environment provides limited fine-grained control over the execution timing of individual jobs. Adaptive workflow processing promises to provide more robust performance in uncertain environments. Our experiments also indicate that workflows with a higher degree of inherent parallelism, such as Montage, may benefit more from adaptation. Finally, our work has demonstrated that effective adaptation can be added to an established grid workflow infrastructure at modest development cost, making use of existing facilities for monitoring and control.

Acknowledgments: We acknowledge the support of the UK Engineering and Physical Science Research Council. The ISI work was supported by the National Science Foundation under grants: CNS-0615412 and OCI-0722019.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing Journal*, 28(5):749–771, 2002.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflow. In *in 16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM'04), 21-23 June*, 2004.

[3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[4] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *CASCON'98*, 1998.

[5] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *IEEE Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[6] A. Chervenak et al. Giggle: A framework for constructing sclable replica location services. *Proceedings of Supercomputing 2002 (SC2002)*, November 2002.

[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459:62–82, 1998.

[8] E. Deelman et al. Transformation catalog design for griphyn. *Technical Report, GriPhyN-2001-17, www.griphyn.org*, 2001.

[9] E. Deelman et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[10] R. Duan, R. Prodan, and T. Fahringer. Run-time optimisation of grid workflow applications. In *Proc. Intl. Conference on Grid Computing*, pages 33–40. IEEE Press, 2006.

[11] T. Fahringer et al. Askalon: A development and grid computing environment for scientific workflows. In *in Workflows for eScience, Scientific Workflows for Grids*. Springer Verlag, ISBN: 978-1-84628-519-6, 2005.

[12] S. Fitzgerald. Grid information services for distributed resource sharing. In *Proc. 10th IEEE Intl Symposium on High Performance Distributed Computing*, 2001.

[13] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *HPDC*, pages 55–63, 2001.

[14] T. Heinis, C. Pautasso, and G. Alonso. Design and evaluation of an autonomic workflow engine. In *2nd International Conference on Autonomic Computing*, pages 27–38. IEEE Computer Society, 2005.

[15] J. Kephart and D. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[16] J.-H. Lee, S.-H. Chin, H.-M. Lee, T. Yoon, K.-S. Chung, and H.-C. Yu. Adaptive workflow scheduling strategy in service-based grids. In *GPC*, pages 298–309. Springer, 2007.

[17] K. Lee, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes. Workflow adaptation as an autonomic computing problem. In *2nd Workshop on Workflows in Support of Large-Scale Science (Works 07) in Proceedings of HPDC*, pages 29–34, 2007.

[18] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–54, 2004.

[19] I. Taylor, M. Shields, I. Wang, and A. Harrison. The triana workflow environment: Architecture and applications. In *Workflows for e-Science*, pages 320–339, 2007.

[20] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002.

[21] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. In *in SIGMOD Record Volume 34(3)*, 2005.

[22] Z. Yu and W. Shi. An adaptive rescheduling strategy for grid workflow applications. In *IPDPS*, pages 1–8. IEEE Press, 2007.

[23] H. Zhao and R. Sakellariou. Advance reservation policies for workflows. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 47–67. LNCS 4376, Springer-Verlag, 2006.