9th IEEE/ACM International Symposium on Cluster Computing and the Grid

Utility Driven Adaptive Workflow Execution

Kevin Lee, Norman W. Paton, Rizos Sakellariou, Alvaro A. A. Fernandes
School of Computer Science, University of Manchester, U.K.

***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 map tasks within a workflow to the available resources. As workflows are typically run over shared resources, and thus face unpredictable and changing resource capabilties, there may be benefit to be derived from adapting the task-to-resource mapping while a workflow is executing. This paper describes the use of utility functions to express the relative merits of alternative mappings; in essence, a utility function can be used to give a score to a candidate mapping, and the exploration of alternative mappings can be cast as an optimization problem. In this approach, changing the utility function allows adaptations to be carried out with a view to meeting different objectives. The contributions of this paper include: (i) a description of how adaptive workflow execution can be expressed as an optimization problem where the objective of the adaptation is to maximize some property expressed as a utility function; (ii) a description of how the approach has been applied to support adaptive workflow execution in grids; and (iii) an experimental evaluation of the resulting approach for alternative utility measures based on response time and profit.**

I. Introduction

Workflow languages provide a high-level characterization of the pattern of activities that needs 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 concrete representation that resolves various of the details as to how the workflow is to make use of available resources.

Most existing workflow systems (e.g., [1], [2]) provide static approaches for mapping 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., [3]), or by dynamically revising compilation decisions that gave rise to a concrete workflow while it is executing (e.g., [4], [5], [6], [7]). In principle, any decision that was made statically during workflow compilation can be revisited at runtime [8].

In common with adaptive techniques in other areas [9], in this paper adaptive workflow execution involves a feedback loop, the implementation of which differs from platform to platform, but in which various themes recur: *monitoring* records information on workflow progress and/or the execution environment; an *analysis* activity identifies potential problems and/or opportunities; a *planning* phase explores alternatives to the current evaluation strategy; and, if adapting is considered beneficial, an *execution* step takes place whereby a revised evaluation strategy is adopted. In all such feedback loops, some form of decision-making process must establish which adaptation is likely to be effective in a given context. In the autonomic computing community, decision-making in autonomic systems has been classified into three types, which are referred to as *policies* [10]: *action policies*, in which the behavior of the system is captured using condition-action rules; *goal policies*, in which one or more desired states are identified and a planner identifies actions that should lead to that state; and *utility function policies*, in which the value of different outcomes is quantified, and an optimization activity seeks to identify actions that maximize utility. In previous work we described the use of an *action policy* to adaptively balance load during workflow execution [11]; this paper investigates the use of utility functions to support the adaptive execution of workflows on grid resources. Advantages of the utility-based approach include the declarative expression of the utility of an adaptation and the use of well founded mathematical optimization techniques (e.g., [12]) to identify adaptations that maximize the utility measure. This differentiates the approach followed in this paper from the large body of literature on mostly bespoke techniques for run-time, dynamic, or multi-criteria workflow scheduling and execution (e.g., [5], [13], [14]).

The context for this work is illustrated in Figure 1. In essence, workflows are submitted to an *autonomic workflow mapper*, which adaptively assigns the jobs in the workflows to execution sites. Each execution site queues jobs for execution on one or more computational nodes. Given some objective, such as to minimize total execution times or, more generally, to optimize for some Quality of Service (QoS) target, the autonomic workflow mapper must determine which jobs to assign to each of the available execution sites, revising the assignment during workflow execution on the basis of feedback on the progress of the submitted jobs. In this paper we describe two different utility measures within a consistent framework.

When a utility-based approach is adopted, the following steps are followed by designers: (i) identify the property that it would be desirable to minimize or maximize – in the

978-0-7695-3622-4/09 $25.00 © 2009 IEEE
DOI 10.1109/CCGRID.2009.15

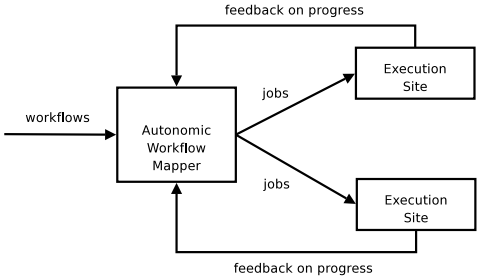

Fig. 1. High level architecture.

case of workflows, useful utility measures may be cast in terms of response time, number of QoS targets met, etc. (ii) define a function $Utility(w, a)$ that computes the utility of an assignment of tasks to nodes a for a workflow w expressed in terms of the chosen property – for workflow mapping, such a function can be expected to include expressions over variables V_E that describe the environment and variables V_M that characterize the mapping from abstract requests to jobs on specific execution nodes; (iii) select an optimization algorithm that, given values for V_E, searches the space of possible values for V_M with a view to maximising the utility function; one benefit of the utility-based approach is that standard optimization algorithms can be used to explore the space of alternative mappings. Several researchers have reported the use of utility functions in autonomic computing, typically to support systems management tasks (e.g. [15], [16]); to the best of our understanding this is the first attempt to deploy utility functions for adaptive workflow execution.

The remainder of this paper is structured as follows. Section II describes utility functions that assign scores to workflow assignments on the basis of response times and cost incurred to meet target deadlines. Section III describes how the search for a mapping that maximizes a utility measure can be cast as an optimization problem. Section IV presents the results of an experimental evaluation of the utility-based approach in the context of the Pegasus workflow management system [3]. Section V presents some conclusions.

II. Utility Functions

A. Problem Statement

A workflow w is a directed acyclic graph, where the nodes consists of a collection of tasks, $w.tasks$ and the edges represent dependencies between those tasks; a workflow is evaluated through an allocation of tasks to a set of nodes. The role of the autonomic workflow mapper is to adaptively assign the tasks to specific nodes. These nodes have different computational capabilities, and thus take different (and, thanks to their shared nature, changing) amounts of time to evaluate a job. As a result of the unstable environment, it is challenging to statically identify an assignment for the tasks in a workflow that remains effective throughout the lifetime of its execution.

The objective of the autonomic workflow mapper is to maximize a utility measure; two utility measures are considered: (i) utility based on response time, so utility is maximized when response time is minimized; and (ii) utility based on profit, where profit for a workflow is maximized by meeting a response time target while incurring minimal resource usage costs. In essence, during the evaluation of a workflow w, the autonomic workflow mapper monitors its progress, and when an alternative assignment is predicted to improve utility, remaps the workflow to conform to the new assignment.

Each execution node is assumed to support the submission of tasks, which are queued prior to execution, at which point the task obtains exclusive access to one of the processors of the node.

B. Utility Functions

1) Utility Based on Response Time: Where utility is based on response time, the objective is to minimize the response time of the workflow for an assignment a of $w.tasks$ to a set of resources. As a result, the utility of a workflow can be represented as being in an inverse relationship to its response time:

$$Utility_w^{RT}(w, a) = 1/PRT(w, a)$$

where, as described in Section II-C, PRT estimates the predicted response time of the workflow given the assignment a and for every $j \in w.tasks$, there exists an assignment $j \rightarrow r$ in a for some $r \in R$, where R is the set of available resources (execution sites).

2) Utility Based on Profit: Where utility is based on profit, it is assumed that a workflow w has a response time target that, when met, gives rise to a payment of value v. Furthermore, it is assumed that nodes charge different amounts for executing a job. The problem, then, is to meet the response time target at minimum cost.

The utility of an individual workflow for an assignment a of tasks to a set of resources can then be represented as:

$$Utility_w^{Profit}(w, a) = \\ (UtilityCurve(w, a) * v) - fcost(w, a)$$

where $UtilityCurve(w, a)$ estimates the success of the allocation a at meeting the response time target of w, $fcost(w, a)$ estimates the financial cost of the resources used, and v is the payment received for meeting the response time target for w. For every $j \in w.tasks$, there exists an assignment $j \rightarrow r$ in a for some $r \in R$; and $Utility_w^{Profit}(w, a)$ is the utility of the given job assignment a for an individual workflow w.

In principle, we can define the $UtilityCurve$ for a workflow w in such a way that it returns 1 if it meets its response time target, and 0 otherwise. However, such a definition is problematic during the search for effective assignments, as every candidate assignment that misses its target has the same utility of 0, no matter how near to or far from the target it is, and every assignment that meets the target has the same utility of 1 no matter how narrowly or comfortably the target is met. This makes it difficult for an optimization algorithm to rank alternative solutions effectively. As a result, we use a definition for $UtilityCurve$ that provides high and broadly

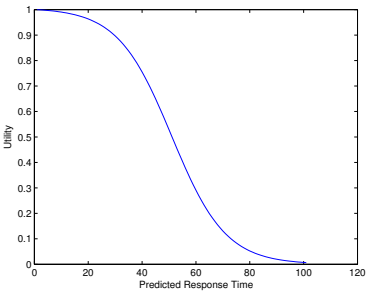

Fig. 2. Utility for a target response time of *50*.

consistent scores for meeting a response time target, and low but broadly consistent scores for missing a target, while also enabling improvements to be recognized during optimization.

We use a function definition from earlier work on resource allocation in data centers [16] (which generates the curve illustrated in Fig. 2 for a target time of *50*):

$$UtilityCurve(w,a) = \frac{e^{-PRT(w,a)+TT(w)}}{1+e^{-PRT(w,a)+TT(w)}}$$

where $PRT(w,a)$ is as defined in Section II-C, and $TT(w)$ returns the target response time of w.

The (expected) financial cost of evaluating a workflow on a set of resources can be obtained as follows:

$$fcost(w,a) = \sum_{t \in w.tasks} fcost(t,a),$$

that is, the financial cost of the workflow is the sum of the financial costs of all tasks it contains. To compute the financial cost of a single task we assume that there is a lookup table that gives the financial cost per execution time unit for each different site. Then, the financial cost of each task is the expected execution time for the task multiplied by the given financial cost per execution time unit for the site on which the task is to be run.

C. Estimating Predicted Response Times

The predicted response time of a workflow for a given assignment a can be estimated as follows:

$$PRT(w,a) = ECT(exit\_task, a),$$

where $exit\_task$ is the exit node of the workflow (it is assumed that the workflow has a single exit node; it is straightforward to convert any workflow to a workflow with a single exit node, by adding a dummy node with zero execution time), and ECT is the expected completion time of the exit task, as described in detail below.

Where adaptations incur a cost, the predicted response time of a workflow for a given assignment a can be estimated as:

$$PRT(w,a) = ECT(exit\_task, a) + AdaptationCost(w, CurrentAllocation(w), a),$$

where $AdaptationCost(w, CurrentAllocation(w), a)$ represents the cost of adapting from the current allocation associated with w to the candidate allocation a. The estimation of this cost depends on the environment in which workflow execution is taking place, and is discussed for our experimental context in Section IV.

To find $ECT(exit\_task, a)$, we can use the following recursive formula that computes the expected completion time (ECT) of any task, $task_i$, of the workflow:

$$ECT(task_i,a) = ET(task_i,a) + EQT(task_i,a) + \max_{task_j \in successors(task_i)}(ECT(task_j,a)),$$

where $ET(task_i,a)$ is the expected execution time of $task_i$, $EQT(task_i,a)$ is the estimated amount of time that $task_i$ will spend in a queue, and $task_j$ is bound in turn to all immediate successors of $task_i$ in the workflow (if a task $task_i$ has no immediate successors, then this component of the formula is zero).

In what follows, it is assumed that ET is given, that is, for each node, for each type of task, it is known how long that task type will take to execute on the node. However, the time spent by a task in the queue, EQT, is neither constant nor known in advance, and is influenced both by external task assignments (those over which the autonomic workflow mapper has no control) to a node and assignments of tasks to machines. The next section describes how EQT can be estimated.

D. Estimating Queue Times

In the following, we consider two time periods, p and p', such that $length(p) = length(p')$ and $End(p) = Start(p')$. Adaptation is being considered at $End(p)$; as a result, p is in the past, and we have access both to information about the assignment a of our workflow w to execution nodes during p and to monitoring information collected during p. In this context, we are interested in estimating the queue time during p' for potential future assignments a'.

The estimated (average waiting) queue time, EQT, during p' depends on: (i) the queue time at the start of p' (or the end of p); (ii) the demand for use of the node during p' as a result of workflow execution over which the autonomic workflow mapper has control (henceforth referred to as $AssignedDemand$); and (iii) the demand for use of the node during p' as a result of allocations of work to the node over which the autonomic workflow mapper has no control (henceforth referred to as $ExternalDemand$); we assume that the $ExternalDemand$ during p' is the same as the $ExternalDemand$ observed during p. By *demand* we mean the fraction of the available resource used during a given period. Thus if the $AssignedDemand$ is 0.5 then the amount of work assigned during p' is such as to fully occupy the node half of the time. If the $AssignedDemand$ is less than 1 and the $ExternalDemand$ is 0 during p' then either the EQT will be 0 (assuming that a node contains several processors) or the EQT at the end of p' can be expected to be less than at the start of p' (as the length of the queue will reduce during p').

An estimate for EQT during p' can be computed as:

$$EQT(n,p,w,a') = \max(0,$$

$$(QueueTime(n, End(p)) + length(p) * (ExternalDemand(n, p) + CandidateDemand(n, w, a')))$$

where $QueueTime(n, End(p))$ is the (monitored) queue time on node n at the time when adaptation is being considered ($End(p)$), $length(p)$ is the period for which the demand levels applied, $ExternalDemand(n, p)$ is an estimate of the demand for the node from tasks over which the autonomic workflow mapper had no control during the period p, and $CandidateDemand(n, w, a')$ is an estimate of the demand that will be placed on the node by the workflows w and candidate assignment a'. As such, the queue time increases if the demand for the resource is greater than the amount of resource available, and decreases if the demand for the resource is less than the amount of resource available.

Given the above definition of estimated queue time, the EQT for a task used in Section II-C can be defined as:

$$EQT(task, a') = \texttt{let } n = \texttt{the } node \texttt{ such that } (task \rightarrow node) \in a' \texttt{ in } EQT(n, p, Workflow(task), a')$$

where p is a configuration property that specifies the period for which monitoring information is to be used to inform queue estimation, and $Workflow(task)$ returns the workflow of which the given $task$ is a component.

To complete the estimate for EQT, definitions for $ExternalDemand$ and $CandidateDemand$ are now provided.

1) Estimating $ExternalDemand$: The level of the external demand can be estimated as follows. The change in the queue time on a node, ΔQT, over a period p can be computed as follows from available monitoring information about the queues on each node:

$$\Delta QT(n, p) = QueueTime(n, End(p)) - QueueTime(n, Start(p)) \quad (1)$$

The change in the queue time also depends on the level of demand on a node during a period.

$$\Delta QT(n, p) = (ExternalDemand(n, p) + AssignedDemand(n, p)) * length(p) \quad (2)$$

which can be rewritten in terms of $ExternalDemand$ as:

$$ExternalDemand(n, p) = \frac{\Delta QT(n,p) - (AssignedDemand(n,p) * length(p))}{length(p)}$$

where ΔQT can be obtained from monitoring information and (1), and the duration of p is a configuration property.

The $AssignedDemand$ during p can be computed based on monitoring information concerning which tasks have been assigned to which machines and the capabilities of the machines:

$$AssignedDemand(n, p) = \frac{\sum_{task \in QueuedTask(n,p)} ET(task, a)}{length(p) * Processors(n)}$$

where $QueuedTask(n, p)$ identifies the tasks assigned to n during p, $ET(Task, a)$ is the execution time of a task in a given assignment, a is the assignment that applied during p, and $Processors(n)$ is the number of processors available on node n.

2) Estimating $CandidateDemand$: The $CandidateDemand$ is an estimate of the demand placed on resources by a candidate assignment a' of the tasks in a workflow w:

$$CandidateDemand(n, w, a') = \frac{\sum_{\{task \in w | (task \rightarrow n) \in a')\}} ET(task, a')}{(TT(w) - ElapsedTime(w)) * Processors(n)}$$

where $ET(Task, a')$ is the execution time of a task in a given assignment, $ElapsedTime(w)$ is the time for which a workflow has been executing, and $Processors(n)$ is the number of processors available on node n. In essence, the $CandidateDemand$ is a measure of the demand that must be placed on a node by the assignments to that node if target response times are to be met.

E. Summary

This section has described two utility functions, $Utility_w^{RT}(w, a')$ and $Utility_w^{Profit}(w, a')$ that can be used to compare the relative merits of different node assignments a' for tasks in w. Both definitions build on predictions of the estimated completion times of w given a', which in turn make use of predictions of average queue times. The queue time predictions principally take account of the impact of the change from the current assignment a to a new candidate assignment a' on the demand being placed on each node, using information that is readily available from the definition of the workflow and from monitoring of queue lengths. The exploration of alternative assignments is discussed in the next section.

III. Obtaining Plans from Utility Functions

In Section II, utility functions were defined that characterize and quantify desirable behaviors. This section describes how they can be used to decide on the benefits of adapting dynamically to changes in the execution environment. At any point in the evaluation of a workflow where adaptation is being considered, the process of obtaining an effective assignment of tasks to resources for a workflow w is a question of identifying assignments a' that maximize $Utility_w^{RT}(w, a)$ or $Utility_w^{Profit}(w, a)$.

To obtain an effective assignment a', a search algorithm can be used that (i) generates an assignment; (ii) calculates the value of the utility based on this assignment; and (iii) uses this value to inform the selection of an appropriate next assignment. The search continues until it converges on a specific value or a maximum number of iterations has taken place. For the purposes of the search, an assignment can be represented as a list of discrete categorical variables, each representing the assignment of a task to a specific execution node. In the experiments, we use the NOMADm [17] implementation of a Mesh Adaptive Direct Search (MADS) [12] algorithm; MADS is a class of nonlinear optimization algorithm that can be used

to maximize a black box function, such as $Utility_w^{RT}(w,a)$ or $Utility_w^{Profit}(w,a)$.

IV. Experimental Evaluation

A. Experimental Context

To evaluate the approach to adaptive workflow execution using utility functions, we have extended the Pegasus workflow management system [3] with a framework based around the *MAPE* functional decomposition [9] which partitions adaptive functionality into four components, *Monitoring*, *Analysis*, *Planning* and *Execution*. The principal components of relevance to the experiments, and their relationships, are illustrated in Figure 3. Pegasus takes as input an *Abstract Workflow*, which is compiled on the basis of metadata from registries and a replica manager to produce a *Concrete Workflow* that is explicit about where individual tasks are to be executed and which physical files are to be read and written by those tasks. Each *Concrete Workflow* is submitted to the DAGMan job manager of Condor-G [18], which in turn submits jobs to computational nodes for execution.

During execution, information about the files produced during evaluation of the *Concrete Workflow* is recorded by the *Replica Manager*, and job *queue*, *execute* and *termination* events are tracked by a *Monitoring* component. Current queue times are derived from these log entries and passed to the *Analysis* component which compares these to the times predicted when the current plan was generated. When a sustained change is detected between the actual and predicted queue times *Planning* is triggered. The *Planning* phase, given the monitored information on queue times, searches for assignments that maximize the chosen Utility measure as described in Section III; where an assignment is produced that is predicted to improve on the current assignment, the new assignment is passed to the *Execution* component. The estimated cost of performing an adaptation is taken into account at this stage, based on micro-benchmarks.

The *Execution* component calls Pegasus again to produce a new concrete workflow from the abstract workflow. A custom Pegasus site scheduler has been developed that uses the assignment produced by the *Planning* component. As intermediate products for already completed tasks are available from the *Replica Manager*, these tasks are not included in the *Concrete Workflow*, and thus completed tasks are not repeated. Once the new concrete workflow has been created with the new assignments and replicas, it can replace the currently submitted concrete workflow. A request is made to DAGMan to halt the current workflow; DAGMan in turn removes the jobs from the remote grid site queues. The new concrete workflow can then be submitted to DAGMan, which again follows the task dependencies and submits the tasks appropriately to the execution nodes. The adaptation process can repeat as many times as necessary.

B. Experimental Setup

The aim of the experiments is to explore the effect of the different utility functions on the execution of workflows on grid resources. The experiments use two abstract workflow styles. The first type is a *linear workflow* which is simply a Directed Acyclic Graph (DAG) were each task is dependent on the file created by the previous task. With these dependencies present, the tasks in the workflow will execute in series. In our experiments we consider linear workflows containing 50 tasks. The second workflow type is a *Montage workflow*, which creates a large mosaic image from many smaller astronomical images [3]. 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 4. The numbers represent the level of each task in the overall workflow. In our experiments we use a 0.2 degree area workflow which has 25 tasks.

Two execution sites were used to run the workflows which we designate *ES 1* and *ES 2*. *ES 1* consists of Intel 2Ghz Pentium 4 CPUs with 1GB of RAM, whilst *ES 2* consists of Intel 2Ghz Core 2 Duo CPUs with 2GB of RAM. The execution sites are connected together directly by Gigabit Ethernet and each have access to sufficient independent and

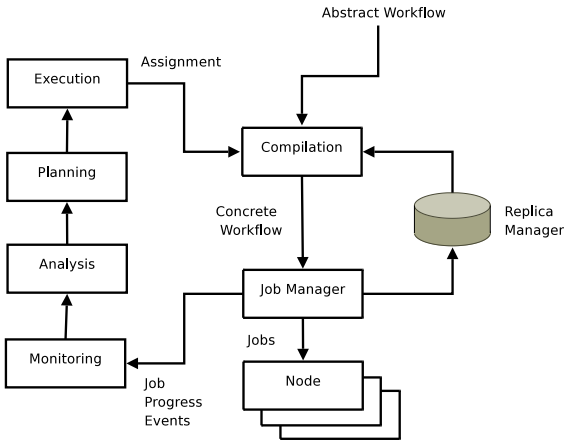

Fig. 3. Adaptive Workflow Execution with Pegasus.

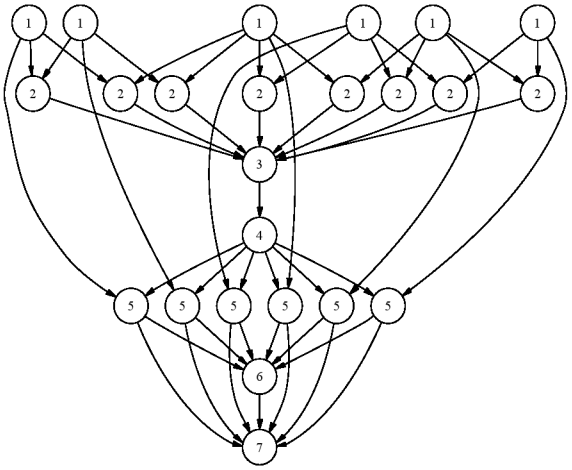

Fig. 4. A Simple Montage Workflow [3]

shared disk storage.

Both execution sites run Debian Linux, the Sun Grid Engine Version 6.1 Update 5 (as the site job scheduler) with its default scheduler, and expose WSGRAM interfaces provided by the Globus Toolkit Version 4.0.7. *ES 2* has double the number of cores of *ES 1*. Under un-loaded conditions *ES 1* has an average queue time of 35 seconds, and *ES 2* has an average queue time of 25 seconds for a typical job.

C. Response Time Utility Experiments

The aim of the first set of experiments is to compare the standard Pegasus non-adaptive approach with the response time utility approach. For these experiments, two separate runs are performed under the same environmental conditions, with adaptation switched off and adaptation switched on. Where adaptation is switched off, Pegasus uses simple round-robin scheduling, whereas when adaptation is switched on the optimization technique described in Section III seeks to maximize $Utility_w^{RT}$ as defined in Section II-B1.

During workflow execution, to induce some uncertainty into the execution environment, a load is applied to *ES 1*. This load consists of a process that submits short jobs (of around 20 seconds duration) every 15 seconds for 5 minutes, then sleeps for 2 minutes before continuing, leading to varying queue times.

Experiment 1. The results for non-adaptive and adaptive execution for the linear workflow are presented in Figure 5, which shows that the adaptive workflow mapper performs much better than the non-adaptive one. With the adaptive mapper, adaptation occurs twice around the times when jobs with *Job ID*s 3 and 7 are executing, respectively, giving rise to the gap in execution whilst data is moved around and the workflow is resubmitted to the grid sites. The adaptations occur early in the execution as knowledge about the workflow execution environment is gained. The gain made by adapting easily makes up for the cost of adapting.

Experiment 2. The results for non-adaptive and adaptive execution for the Montage workflow are presented in Figure 6, which again shows that the adaptive workflow mapper performs better than the non-adaptive one. In this case, the adaptive workflow mapper adapts once early in workflow execution. The significant gap in execution for the adaptive case results from the fact that job queue and execution times are shown only for completed jobs. During this experiment, adaptation wasn't triggered until jobs from the loaded *ES 1* began execution, and these were not able to complete before the adaptation occurred. After the large queue times were detected, the resulting assignment maps the majority of the remaining jobs to *ES 2*.

D. Profit Utility Experiments

The aim of the second set of experiments is to compare the Standard Pegasus non-adaptive approach, the response time utility approach (*U(RT)*) and the profit utility (*U(Profit)*) approach with respect to their ability to maximize profit. Profit is maximized by meeting response time targets without resorting to the use of unnecessarily expensive resources. In the experiments, the target response time is varied, with relatively high (easy to meet), medium (somewhat challenging to meet) and low (very challenging to meet) values for target response time. For each target response time, three separate runs are performed under the same environmental conditions as in Experiments 1 and 2 (that is, a load is applied to *ES 1* as described in Section IV-C).

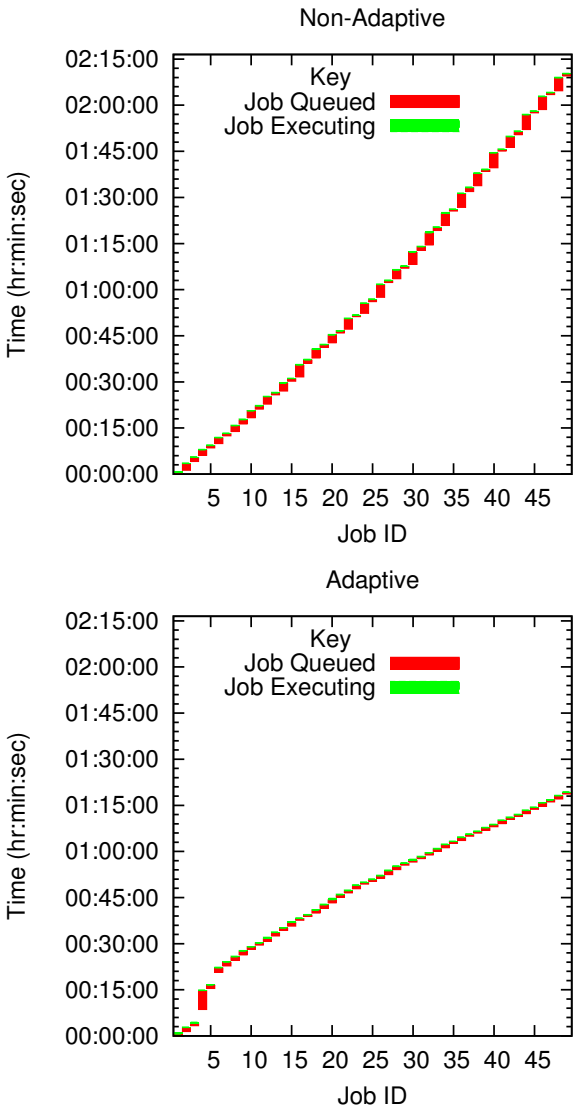

Fig. 5. Experiment 1: Workflow progress plots, showing when each job is queued and executed, for non-adaptive execution and the response time utility approach with a linear workflow.

As profit is a cost-based metric, the (monetary) cost of executing jobs on resources and the reward for meeting a target response time must be known. For these experiments, the costs of executing jobs on each site were chosen on the basis of their characteristics – the faster site is also the more expensive. The cost of executing a single job on *ES 1* is 1 unit of currency and

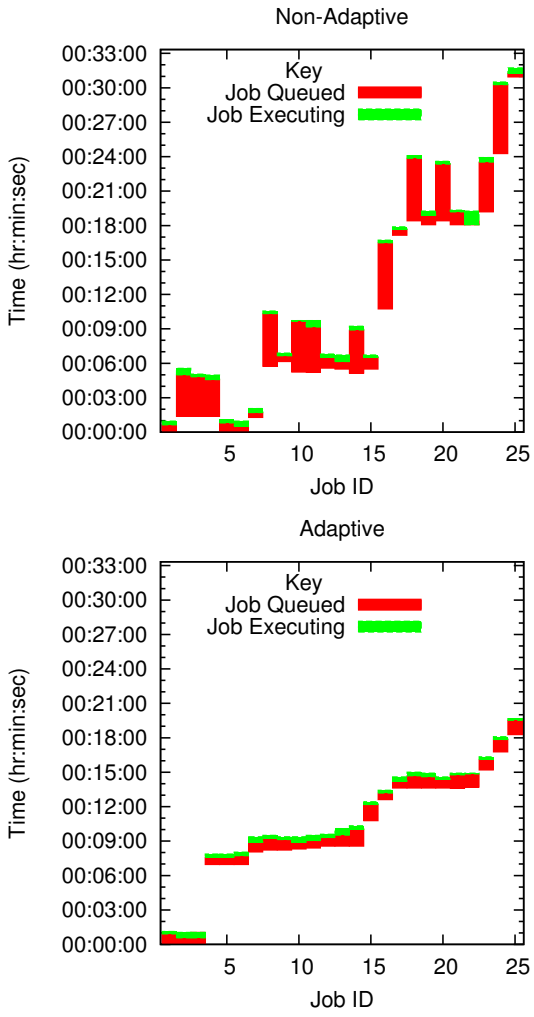

Fig. 6. Experiment 2: Response time utility and Montage workflow.

on *ES 2* is 2 units of currency. These costs are only incurred for jobs that have partially or completely run on a site, and not for jobs that are only queued. The reward for meeting the target response time is 100 units of currency.

Both profit and response times are reported for each workflow run. Profit is calculated as follows:

$$Profit(w) = (TargetMet(w) * reward) - Cost(w)$$

where $TargetMet$ is either 1 or 0 depending on whether the workflow execution meets the target response time or not; $reward$ is 100 units of currency; and $Cost$ is the overall financial cost of running the workflow, calculated after it has completed (this is, essentially, the number of tasks run on *ES 1* plus two times the number of tasks run on *ES 2*).

Experiment 3. The profit and response time results for linear workflow runs are reported in Figures 7 and 8, respectively. In this experiment, the *High* response time target is 3 hours (10800 seconds), the *Mid* target response time is 2 hours (7200 seconds) and the *Low* target response time is 1 hour (3600 seconds).

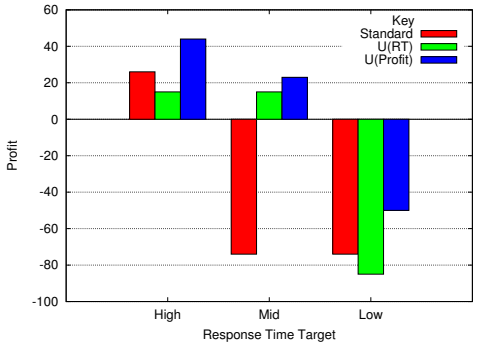

Fig. 7. Profit comparison for Experiment 3

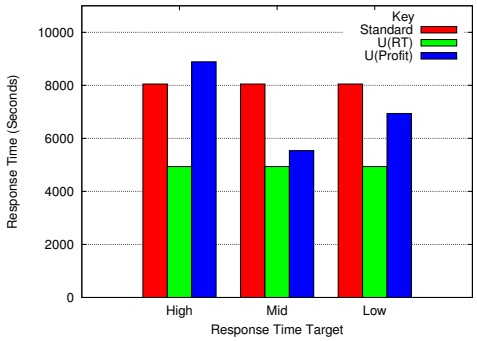

Fig. 8. Response Time comparison for Experiment 3

For the *High* response time target, all three strategies meet the target, and return a profit. The utility response time strategy completes with the lowest response time, but obtains the good response time by making extensive use of (expensive) *ES 2*. The utility profit approach provides the slowest response time but the highest profit, by only using *ES 2* when this was strictly necessary to meet the response time target. The non-adaptive approach allocates jobs evenly between the two execution sites, and thus comes in between the two utility approaches in terms of both response time and profit.

It can be seen from the response time results in Figure 8 that the non-adaptive and utility response time strategies yield the same response times in all cases, as neither explicitly seeks to meet response time targets.

For the *Mid* response time target, the non-adaptive approach fails to meet the target, and thus yields a significant loss. Both utility strategies meet the response time target, and yield a profit, but the profit is greater for the utility profit approach. This is because the utility profit approach, although using *ES 2* more than for the *High* response time target, manages to meet the response time target while using the inexpensive *ES 1* more than the utility response time approach.

For the *Low* response time target, none of the approaches meet the target, and thus all make a loss. However, the utility profit strategy makes the smallest loss because, realising that the response time target is not going to be met, it avoids extensive use of *ES 2*.

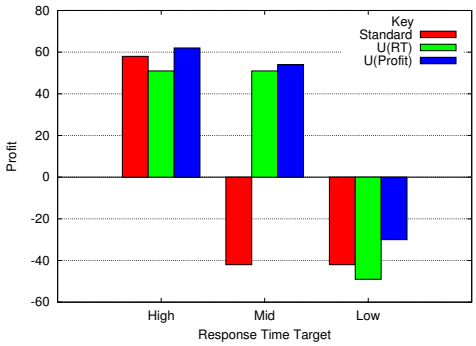

Fig. 9. Profit comparison for Experiment 4

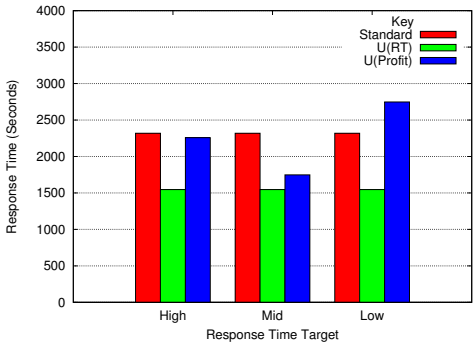

Fig. 10. Response Time comparison for Experiment 4

Experiment 4. The profit results for the standard Pegasus non-adaptive approach, the response time utility approach and profit utility approach with varying target response times for a Montage workflow are presented in Figure 9. The corresponding response times are presented in Figure 10. For this experiment, the *High* target response time is 40 minutes, the *Mid* target response time is 30 minutes and the *Low* target response time is 20 minutes.

Experiment 4 confirms the results of Experiment 3 with a different type of workflow. All three approaches succeed in meeting the high target response time. However, the profit utility approach consistently adapts in a way that yields the best profit (or smallest loss), while never improving on the response time of the utility response time approach.

E. Summary

Experiments 3 and 4 show the clear differences between the approaches when costs are associated with the use of resources, and indicate the power of utility functions. As the non-adaptive approach is unaware of response times or resource costs, it is consistently least competitive with respect to response time. By contrast, the response time utility approach consistently achieves the lowest response times, though at the cost of using the most expensive resources. In both experiments, the profit-based utility approach successfully trades off the costs and the benefits of the resources to yield the best financial results.

V. Conclusion

This paper presents a utility-based approach for adaptive workflow execution, which has been used in an autonomic computing framework and tested in the context of the Pegasus workflow management system. Our results demonstrate that the design of appropriate utility functions can enable optimization for different QoS targets in Grid workflow execution. As opposed to bespoke approaches to address similar problems, the main lesson from our work is that utility-based approaches allow adaptive workflow management system developers to cast the problems in a systematic way that can be addressed with well founded mathematical optimization techniques. In future work, we aim to expand our experiments, targeting multiple QoS targets and concurrently running workflows.

References

[1] 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 *CCGrid'05*, pages 759–767, 2005.

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

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

[4] 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.

[5] 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.

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

[7] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 12(4):253–262, 2004.

[8] K. Lee, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes. Workflow adaptation as an autonomic computing problem. In *Proc. 2nd Workshop on Workflows in Support of Large-Scale Science*, pages 29–34. ACM Press, 2007.

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

[10] J.O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[11] K. Lee, N.W. Paton, R. Sakellariou, E. Deelman, A. A. A. Fernandes, and G. Metha. Adaptive workflow processing and execution in pegasus. In *3rd Intl Workshop on Workflow Management and Applications in Grid Environments (WaGe08), in Proc. 3rd Intl. Conf. on Grid and Pervasive Computing Symposia/Workshops*, pages 99–106. IEEE Press, 2008.

[12] C. Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. on Optimization*, 17(1):188–217, 2006.

[13] M. Wieczorek, A. Hoheisel, and R. Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25(3):237–256, 2009.

[14] J. Yu and R. Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14:217–230, 2006.

[15] W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proc. ICAC*, pages 70–77. IEEE Press, 2004.

[16] M.N. Bennani and D.A. Menasce. Resource allocation for autonomic data centres using analytic performance models. In *Proc. 2nd ICAC*, pages 229–240. IEEE Press, 2005.

[17] M. A. Abramson, C. Audet, and J. E. Dennis. Nonlinear programing with mesh adaptive direct searches. *SIAG/Optimization Views-and-News*, 17(1):2–11, 2006.

[18] 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.