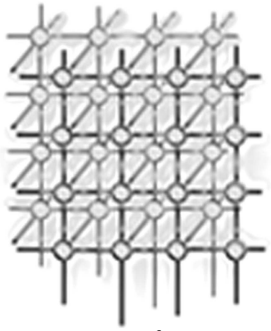
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Utility Functions for Adaptively Executing Concurrent Workflows



Kevin Lee[1], Norman W. Paton[1], Rizos Sakellariou[1] and Alvaro A. A. Fernandes[1]

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

SUMMARY

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 capabilities, 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 a utility function; (ii) a description of how the approach has been applied to support adaptive workflow execution in execution environments consisting of multiple resources, such as grids or clouds, in which adaptations are coordinated across multiple workflows; and (iii) an experimental evaluation of the approach with utility measures based on response time and profit using the Pegasus workflow system.

1. 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 [1, 2]. 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., [3, 4]) 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., [5]), or by dynamically revising compilation decisions that gave rise to a concrete workflow while it is executing (e.g., [6, 7, 8]).

In principle, any decision that was made statically during workflow compilation can be revisited at runtime [9]. Adaptations can be performed for different reasons, including i) changes to the executing environment, ii) changes to the workflow description and iii) adaptation to user (workflow submitter) requirements. Changes to the execution environment can be resources becoming available, computation bottlenecks being detected or resources failing. The workflow description can be changed in order to take advantage of new data or services becoming available. A user may decide to change the overall task to, for example, require providence data to be collected or use specific computational services. Furthermove, adaptations can be performed for different reasons, including prospective (to improve future performance), reactive (to react to previous results), and altrustic (to aid other areas of the workflow).

In common with adaptive and autonomic computing techniques in other areas [10], in this paper, adaptive workflow execution involves a feedback loop, the implementation of which differs from platform to platform, but in which various phases 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. Adaptive workflow execution techniques may differ in all of these phases [9]. In this paper: *monitoring* captures progress information in the form of job completion times and queue lengths; *analysis* identifies where monitoring information departs from expectations; *planning* uses utility functions to consider how different allocations of tasks to resources may give rise to higher utility (in the form of reduced response times or increased profits); and *execution* applies the updated resource allocations, reusing work carried out to date.

A software framework has been developed that assigns to each of these phases software components, built by the authors, that are generic by design. These components can then be instantiated to become the autonomic manager of a specific software artefact simply by being provided with specifications of what to monitor, how to analyse the monitoring information, how to plan an adaptation, and how to execute the latter. In the case of this paper, the specific managed artefact is a workflow engine, comprising a compiler from abstract to concrete workflows and a job manager. Therefore, the specifications that are passed into the adaptivity framework to make its behaviour specific to the managed artefact cause the resulting system to behave in an autonomic manner.

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, namely reponse time and profit, to capture QoS targets within a consistent framework. To the best of our knowledge, our work is the first to

make use of the combination of utility functions and optimization algorithms for adaptive workflow execution. In so doing, we bring to the table a declarative approach to dynamic scheduling in which an optimization algorithm proposes assignments of tasks to resources that maximise utility, following the strategy of Kephart *et al.* [11].

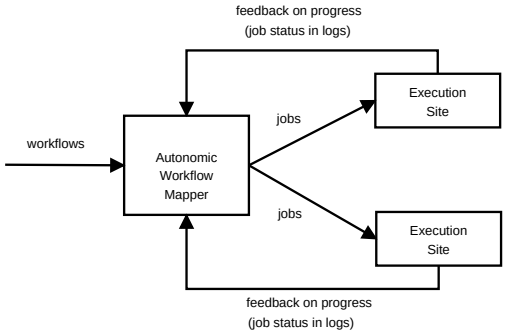
We note that the terms *utility* and *utility function* are used quite widely; in general terms, a *utility function* is a function that computes a value that represents the desirability of a state. However, approaches that seek to maximise some measure of utility differ in the way in utility informs decision making. For example, Huebscher and McCann [12] use utility functions to express application requirements for component selection, in a setting where each component provides the parameters that enable its utility to be computed (i.e. there is no search problem as such, the utility function is essentially metadata that informs component selection). By contrast, in workflow scheduling, Yu *et al.* [13], use a utility measure to give a value to assignments of tasks to resources, in an adaptive scheduling system where the overall problem is divided into a number of steps, which are addressed using heuristics or local searches. Thus, in contrast to the approach used here, neither of these strategies use optimization methods to identify results that maximise utility.

When a utility-based approach to autonomic computing is adopted following the strategy of Kephart *et al.* [11], the following steps are followed by designers:

1. Identify the property that it would be desirable to minimize or maximize – in the case of workflows, useful utility measures may be cast in terms of response time, number of QoS targets met, execution cost, etc.
2. Define a function $Utility(W, a)$ that computes the utility of an assignment of tasks to nodes, a, for a set of workflows 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.
3. 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.

The utility-based approach offers various benefits, in particular: (i) the objectives of the adaptation are stated explicitly and declaratively, thus separating out the specification of the objective of the adaptation from the code that implements decision making; (ii) the search for means of maximizing utility are able to make use of well established optimization techniques [14]; (iii) where utility functions are defined over collections of workflows, optimization considers the combined effect of all the adaptations together, thus reducing the risk, when workflows are adapted independently, that different adaptations might interfere with each other in undesirable ways; and (iv) it is often straightforward to revise a utility function to prioritize different goals.

Moreover, because of the generic nature of the adaptivity framework, designers are less encumbered when experimenting with adaptation strategies. The fact that the adaptive behaviour exhibited is specifiable (i.e., that the adaptive framework, as a piece of software, exposes an instantiation inlet) allows the designer to explore and experiment with variants of an adaptation strategy (i.e., what to monitor, what analyses to carry out, etc.) by simply changing the specifications. In other words, designers need not incur the high cost of intruding on code to modify it and hence need not raise the risk that such intrusions disrupt the behaviour of the autonomic manager in ways that are difficult to explain, let alone avoid or reverse.



[tb]

Figure 1. High level architecture.

The remainder of this paper is structured as follows. Section 2 places this work in the context of other results in decision making in autonomic computing and adaptive workflow execution. Section 3 describes utility functions that assign scores to workflow assignments on the basis of response times and cost incurred to meet target deadlines. Section 4 describes how a generic adaptive framework, based on a monitoring, analysis, planning ane execution functional decomposition, has been instantiated to yield the software architecture needed to support adaptation based on utility functions. Section 5 presents the results of an experimental evaluation of the utility-based approach in the context of the Pegasus [5] workflow management system. Finally, Section 6 concludes the paper.

2. Related Work

This paper investigates how adaptive workflow execution can be controlled in a way that reflects explicitly stated goals. To place this in context, this section reviews related work on *decision making in adaptive systems* and on *workflow scheduling and adaptive workflow execution*.

In the context of *decision making in adaptive systems*, adaptive behaviour stems from a feedback loop that changes the behaviour of the system in response to monitoring information, and thus 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* [11]: *action policies*, in which the behaviour 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.

Within this classification, typical *action policies* directly encode what action should be taken in response to monitoring information. For example, in workflow execution an action policy, when workload imbalance is detected, might directly compute a new workload allocation based on the queue lengths of the available resources (e.g. [15]), or might deploy heuristics to identify revised schedules

(e.g., [16]). By contrast, a typical *goal policy* has an explicit objective, and changes parameter values of the system being adapted, with a view to moving towards the objective. For example, applications of control theory to software systems are increasingly widespread [17], in which a model is developed that computes how to change the parameters of a system on the basis of monitored information. For example, control theory techniques have been applied to tasks such as load balancing (e.g., [18, 19]) and the configuration of block transfer sizes in web services [20]. To date, most results on *utility policies* have been in the context of systems management tasks, such as dynamic selection of web service implementations to support QoS goals [21], allocation of computational resources to applications with different priorities [22], balancing performance with power consumption [23], and network configuration [24]. To the best of our knowledge this paper, extending work initially presented in [25], reports on the first attempt to deploy utility functions with optimization algorithms for adaptive workflow execution.

In the context of *workflow scheduling and adaptive workflow execution*, there has been a vast amount of work on workflow scheduling. Whether this problem is studied specifically in the context of scientific workflows, or, more generally, in the context of directed acyclic graphs, numerous techniques have been proposed that attempt to optimize performance for a certain type of resources and workflow graphs; for representative work we refer to [3, 26, 27, 28, 29, 30, 31]. The common characteristic of these heuristics is that they base their decisions on statically available (or predicted) information, which ignores the state of the execution environment at run-time. Although it has been suggested that, when run-time changes are within certain bounds of what can be estimated, heuristics with good performance statically are more likely to perform better in the presence of run-time changes [16], such run-time changes may be unbounded and may dominate workflow execution.

The reason for such run-time changes is two-fold. On the one hand, it is highly difficult to predict accurately the execution time of individual jobs (tasks) composing the workflow. Good performance prediction has been a challenge that attracted lots of interest over the years [32, 33, 34], however, there can always be run-time-dependent scenarios that are not possible to predict in advance and make an impact on performance. On the other hand, efficient workflow execution requires the concurrent execution of potentially parallel jobs (different tasks of the workflow without any dependence between them) in a coordinated manner. This is again a difficult problem as the parallel jobs may run on different machines, belonging to different administrative domains, having different job queue sizes, job scheduling policies, etc. Progress in parallel job scheduling [35], including techniques for co-allocation and advance reservation [36, 37, 38], may provide a partial solution even with the risk of possibly adverse effects [39]. An extensive discussion is beyond the schope of this paper, however, such techniques still suffer because of the inherent lack of predictability of individual jobs.

As a result, to address the overall unpredictability of the environment, popular strategies have been to delay scheduling decisions until run-time [3, 40] (just in-time scheduling) or to re-evaluate the schedule at run-time and reschedule as needed [7, 8, 16, 15]. The common characteristic of all this body of work is that it makes use of bespoke techniques, which are often interlinked with both scheduling phases (static and dynamic). In contrast, our work decouples the problem and models the objectives of adaptation separately and in a systematic way using utility functions. This allows us to focus on workflow execution *per se* while, at the same time, we can also address challenging problems that involve multi-criteria decisions [41, 42, 43, 44, 13], where different factors may determine what is regarded as efficient execution; this includes multiple workflows competing for the same resources, budget constraints and external loads to these resources.

3. Utility Functions

3.1. Problem Statement

The task is to adaptively schedule a set of workflows W, $w \in W$, where each workflow w is a directed acyclic graph whose node set is a collection of tasks $w.tasks$ and whose 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, independently and autonomously varying) 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 here: (i) utility based on response time, so that utility is maximized when response time is minimized; and (ii) utility based on profit, so that utility is maximized when profit is maximized and 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 collection of workflows, the autonomic workflow mapper monitors their progress, and when an alternative assignment is predicted to improve utility, remaps the workflows to conform to the new assignments. The remapped workflows do not repeat any tasks completed using the previous mappings; the files containing the results of completed tasks are read directly by the remapped workflows.

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. It is assumed that nodes may be shared both by multiple workflows and by other jobs or tasks over which the autonomic workflow mapper has no control.

3.2. Top Level Utility Definitions

This section defines utility functions for response time and profit. In essence, the functions declaratively specify the value of an assignment for a set of workflows. It is then the role of an optimization algorithm to explore the space of possible assignments to identify those that maximize utility.

3.2.1. Utility Based on Response Time

Where utility is based on response time, the goal is to minimize the sum of the response times of the workflows in the workload.

The overall utility of an assignment a of tasks within the set of workflows W to a set of resources R is the sum of the utilities of the workflows in W, and is given by:

$$Utility_W^{RT}(W, a) = \sum_{w \in W} Utility_w^{RT}(w, a)$$

where for every $w \in W$ and every $j \in w.tasks$, there exists an assignment $j \rightarrow r$ in a for some $r \in R$; and $Utility_w^{RT}(w, a)$ is the utility of an individual workflow w for the given job assignment.

The utility of an individual workflow can then be represented as being in an inverse relationship with its response time:

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

where PRT estimates the predicted response time of the workflow given the assignment a, as described in Section 3.3.

As the utility functions for both response time and profit make use of PRT, we now introduce the definition of utility for profit, before returning to response time estimation.

3.2.2. Utility Based on Profit

Where utility is based on profit, it is assumed that each 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 as many response time goals as possible at minimum cost.

The overall utility of an assignment a of tasks from the set of workflows W to a set of resources R is the sum of the utilities of the workflows in W, and is given by:

$$Utility_W^{Profit}(W,a) = \sum_{w \in W} Utility_w^{Profit}(w,a)$$

where for every $w \in W$ and 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 an individual workflow w for the given job assignment. The utility of an individual workflow, representing the profit obtained from its evaluation, 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.

In principle, we could define the $UtilityCurve$ for a workflow w in such a way that it returns 1 if w 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 for lack of clearer signs of movement in the right or the wrong direction. As a result, we use a definition for $UtilityCurve$ that provides high and broadly consistent scores for meeting a response time target, and low and 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 centres [45] (a scaled version of which, for a target time of *50*, generates the curve illustrated in Figure 2):

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

[tb]

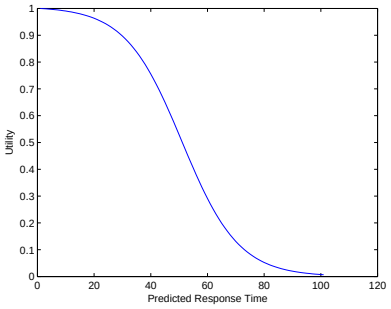


Figure 2. Utility for a target response time of *50*.

where $PRT(w, a)$ is the predicted response time, as defined in Section 3.3, 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 look-up 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.

3.3. Estimating Predicted Response Times

Both response time and profit-based utility models depend on the ability to predict the response time of a workflow for a given assignment. The predicted response time of a workflow for a given assignment is the predicted completion time of the last task in the workflow to complete.

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

$PRT(w, a') = ECT(start\_node(w), a') + \ AdaptationCost(w, CurrentAllocation(w), a')$,

where $AdaptationCost(w, CurrentAllocation(w), a')$ represents the cost of adapting from the current allocation associated with w to the candidate allocation a', *start_node(w)* is the root node of the workflow graph, and ECT estimates the expected completion time of a workflow rooted at a given task for a given assignment. The estimation of $AdaptationCost$ depends on the environment in which workflow execution is taking place.

To find $ECT(start\_node(w), a')$, we can use the following recursive formula that computes the expected completion time (ECT) of any workflow rooted at a 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 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, EQT, i.e., the time spent by a task in the queue, is neither constant nor known in advance, and is influenced both by external job assignments (e.g., tasks over which the autonomic workflow mapper has no control) to a node and specific assignments of tasks to machines. The next section describes how EQT can be estimated.

3.4. Estimating Queue Times

The queue times experienced by the jobs in a workflow are a crucial factor influencing the predicted response time of that workflow and can be estimated as described in this section.

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 the queue time on a node n during the period p' immediately following p experienced by a workflow w using an assignment a' can be computed as:

$$\begin{aligned} EQT(n, p, w, a') = \max(0,\ & (QueueTime(n, End(p)) + length(p) * \\ & (ExternalDemand(n, p) + \\ & CandidateDemand(n, w, a'))), \end{aligned}$$

where $QueueTime(n, End(p))$ is the (monitored) queue time on node n when adaptation is being considered (at $End(p)$), $length(p)$ is the period for which the demand levels applied, $ExternalDemand(n, p)$ is an estimate of the demand for node n 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 3.3 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 the queue time, definitions for $ExternalDemand$ and $CandidateDemand$ are now provided.

3.4.1. Estimating External Demand

The level of external demand on a node n during a period p can be estimated by taking into account the change in the queue time during p and information about the work assigned to n during p. The change in the queue time on a node, ΔQT, over a period p can be computed from available monitoring information about the queues on each node, as follows:

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

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 by the autonomic workload mapper 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.

3.4.2. Estimating $CandidateDemand$

The $CandidateDemand$ is an estimate of the demand placed on a node n by a candidate assignment a' of the tasks in a workflow w; the intuition is that the candidate demand depends both on the amount of work to be assigned to the node and the period during which the work will be carried out.

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

where $ET(Task, a')$ is the execution time of a task in a given assignment, $PreviousECT(w)$ is the estimated completion time for the workflow $ElapsedTime(w)$ is the time for which a workflow has been executing, and $Processors(n)$ is the number of processors available on node n. $PreviousECT(w)$ is the estimated completion time for the workflow using the current assignment, computed at the time of the previous adaptation using ECT from Section 3.3. An initial estimate for ECT is obtained using historical information about the queue times when the workflow is first compiled.

3.5. Optimizing Utility

This section has defined 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.

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:

1. generates an assignment;
2. calculates the value of the utility based on this assignment; and
3. 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; the specific optimization strategy used is described in Section 4.

4. Software Architecture

To evaluate the approach to adaptive workflow execution using utility functions, we utilize the Pegasus workflow management system [5]. Pegasus is a compiler that translates (maps) between the high-level specifications of an abstract workflow and the underlying execution system and optimizes the executables based on the target architecture. The translation 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. The result of the mapping process is an executable or concrete workflow, which can be executed by a workflow engine that follows the dependencies defined in the workflow and executes the activities defined in the workflow tasks. Pegasus uses the Directed Acyclic Graph Manager (DAGMan) workflow executor for Condor-G [46] to submit tasks in the concrete workflow in sequence. This sequence must respect task dependencies (certain tasks cannot start execution before other tasks have completed) and may order tasks following a scheduling algorithm, such as HEFT [31], which applies list scheduling principles to prioritize tasks on the critical path. In this way Pegasus takes high-level descriptions of complex applications structured as workflows (abstract workflows), automatically

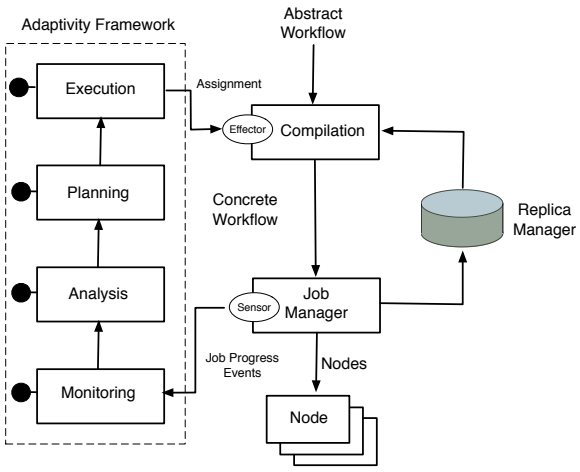


Figure 3. Adaptive Workflow Execution with Pegasus.

maps them to available computational resources (concrete workflows), and submits them to DAGMan for 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. The individual tasks are submitted using a DAG scheduling algorithm and Condor-G to computational nodes, until the workflow completes.

To enable adaptive behaviour, we extend Pegasus [5] with a generic adaptive framework that follows the *MAPE* functional decomposition proposed in [10]. This approach partitions an adaptation strategy into four phases (each corresponding, in our case, to distinct software components), viz., *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 of workflow adaptation [9].

All the software components in the framework are generic, in the sense that they expose an instantiation inlet that, upon being provided with a high-level specification of the required parameters, causes the component to display a specific behaviour which is appropriate for the engine that is being targeted for the role of managed component. As indicated in Figure 3, the framework is retrofitted to the existing Pegasus infrastructure to enable the desired behaviour. There are minimal touchpoints needed between the adaptive framework and the Pegasus workflow management system. A *sensor* collects the required data about the executing workflow and an *effector* enacts changes to the assignment of the workflow.

During workflow execution, log events are passed back to DAGMan via the Grid Site Scheduler and remote job manager. These events indicate the current status of each task's execution, e.g., if it is held, queued, executing, completed or failed. This log provides a snapshot of a point in the execution of the workflow. DAGMan uses this information to determine when to submit workflow tasks that have dependencies on other tasks, when the workflow has failed and when the workflow has completed.

For the use of the framework in this paper, the job *queue*, *execute* and *termination* events are tracked by a sensor which delivers them to the *Monitoring* component as a stream of XML events. These events are generated in the normal process of executing DAG-based jobs on grid execution sites, collecting these events out-of-band of the execution produces no overhead and does not interfere with job execution. The Monitoring component derives current queue times from these events which are passed to the *Analysis* component. Analysis uses a stream query processor to compare these current queue times to the queue times predicted when the current plan was generated. When a sustained change is detected by Analysis between the actual and predicted queue times *Planning* is triggered. In the experiments, we use the STREAM system [47] as our stream query processor. Therefore, the substantive part of the XML document that instantiates the Analysis component is a continuous query, over the stream of data generated by the Monitoring component. The query is in CQL [48], the continuous query language of the STREAM system.

The *Planning* component, given the monitored information on queue times, searches for assignments that maximize the chosen Utility measure as summarized in Section 3.5. 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. When 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 (i.e., $AdaptationCost$ in Section 3.3) is taken into account at this stage, based on micro-benchmarks. In the experiments, we use the NOMADm [49] implementation of a Mesh Adaptive Direct Search (MADS) [50] 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 $Utility_w^{Profit}(w,a)$.

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.

During execution, information about the files produced during the evaluation of the *Concrete Workflow* is recorded by the *Replica Manager*. As intermediate products for already completed tasks are available from the *Replica Manager*, these tasks are not included in the *Concrete Workflow*.

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 the job manager to halt the current workflow; the job manager removes the jobs from the remote grid site queues. The new concrete workflow can then be submitted to DAGMan, which, following the scheduling policy used, submits the tasks appropriately to the execution nodes. This adaptation process could be repeated many times if the utility-based decision making process judges it beneficial to do so.

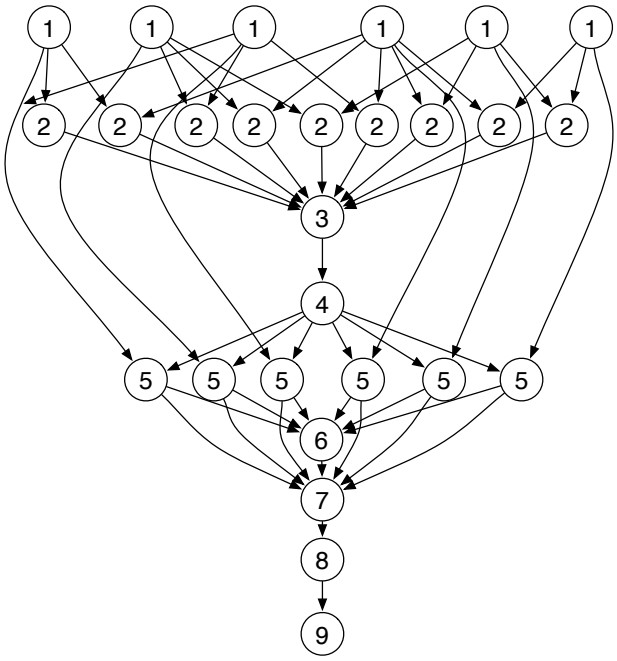


Figure 4. Montage Workflow used in the experiments.

5. Experimental Evaluation

5.1. Experiment Setup

The aim of the experiments is to explore the effect of the two utility functions on the execution of multiple workflows on different compute resources. The experiments use *Montage workflows*, which create large mosaic images from many smaller astronomical images [5]. These can be of varying sizes depending on the size of the area of the sky of the mosaic. The Montage workflow used in the experiments is adapted from [5] and illustrated in Figure 4. It contains 27 tasks and corresponds to a 0.2 degree area. The numbers indicate the level of each task in the overall workflow.

Four execution sites were used to execute the workflows. We designate these, *ES 1*, *ES 2*, *ES 3* and *ES 4*. *ES 1* consists of Intel 2Ghz Core 2 Duo CPUs with 2GB of RAM, whilst *ES 2*, *ES 3*, *ES 4* consists of Intel 2Ghz Pentium 4 CPUs with 1GB of RAM. The machines are connected together directly by Gigabit Ethernet and each have access to sufficient independent and shared disk storage.

All 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 WS-GRAM interfaces provided by the Globus Toolkit

Version 4.0.7. Under unloaded conditions, *ES 1* has an average queue time of 25 seconds, and the other execution sites have an average queue time of 35 seconds for a typical job.

During workflow execution, to introduce some uncertainty into the execution environment, a load is applied to *ES 2*. 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. This leads to varying queue waiting times.

The experiments compare three strategies for workflow execution. One strategy is entirely non-adaptive; tasks are statically scheduled using HEFT [31] and there are no reallocations from the execution sites where they are initially placed. The other two strategies are adaptive; one strategy adapts using the utility based on Response Time (Utility(RT), or simply U(RT), in the graphs), and the other strategy adapts using the utility based on Profit (Utility(Profit), or simply U(Profit), in the graphs).

For the purposes of calculating profit and assessing the behaviour of the adaptive strategy using the utility based on profit, we assumed a simple model where the (monetary) cost of executing jobs on each site was chosen on the basis of the site's characteristics – the faster site is the more expensive. Thus, the cost of executing a single job on *ES 1* is 2 units of currency and on each of *ES 2*, *ES 3*, and *ES 4* is 1 unit 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. Finally, the reward for meeting the target response time for a single workflow is 100 units of currency.

5.2. Experiment 1: Adaptive vs Non-adaptive

The aim of this experiment is to demonstrate the benefits of adaptivity. For this, we compare the standard Pegasus non-adaptive HEFT-based strategy for workflow execution with an adaptive strategy that makes use of the utility based on response time. Figure 5 shows, in detail, the execution as it progresses for each workflow task (job) of a single Montage workflow using a non-adaptive HEFT-based schedule on two execution sites, *ES 1* and *ES 2*, with *ES 2* being loaded with a periodic load as described in the previous section. The effect of the task dependencies is easily visible in the workflow execution. For example, task 16 (which is the only task at level 3 of the workflow) cannot start execution until all previous tasks have completed. The same is true for task 17 and so on. The presence of different queue lengths is due to the varying queue waiting times between execution sites and the load applied to *ES 2*.

Figure 6 shows the execution of the same Montage workflow, this time using the adaptive strategy with the utility based on response time. In this case, the adaptive strategy changes the workflow's schedule twice in the execution, at about 3 minutes and 8 minutes from the start of the execution.

The non-adaptive workflow completed in 34 minutes and 34 seconds, whereas the adaptive strategy with the utility based on response time completed in 21 minutes and 5 seconds. This 39% reduction in response time is attributed to the adaptive workflow moving task assignments from *ES 2*, which has relatively long queue times, to *ES 1* with shorter queue times. This results in an overall improved response time for the adaptive workflow.

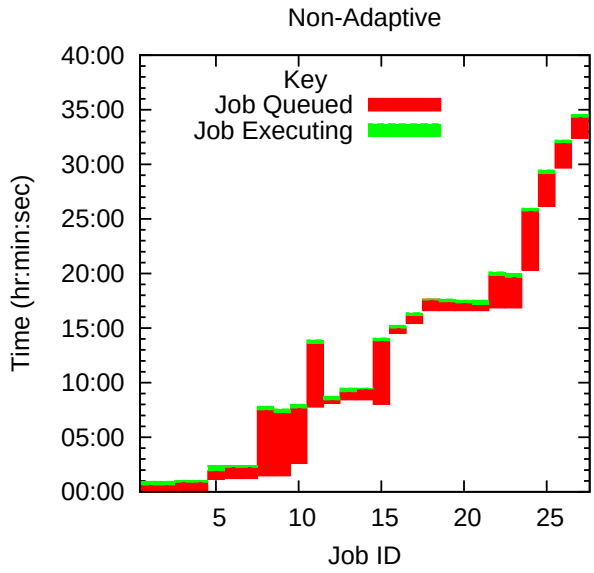


Figure 5. Experiment 1: Montage workflow progress plots, showing when each job (task) is queued and executed, for static (non-adaptive) HEFT execution.

5.3. Experiment 2: Single Workflow with different target response times

The aim of this experiment is to evaluate both the response time-based utility and the profit-based utility strategies against each other and the non-adaptive strategy. A single workflow is executed on execution sites *ES 1* and *ES 2*, the latter being loaded as described in the previous section. Each strategy is evaluated against three different target response times for the completion of the workflow execution: a *Loose* target, a *Mid* target and a *Tight* target. The *Loose* target is calculated from previous experiments as an easy to meet response time target, the *Tight* target is calculated from previous experiments as a hard to meet response time target, the medium being in between. For this experiment, the *Loose* target response time is 40 minutes, the *Mid* target response time is 30 minutes and the *Tight* target response time is 20 minutes.

The response time and profit for each of the three workflow execution strategies and the three target response times tried in this experiment are reported in Figures 7 and 8, respectively. They show that different response time targets have no effect on the execution strategy using HEFT or the utility based on response time, as neither of these strategies seek to meet specific response time targets. The workflow execution strategy using the utility based on response time always produces the best response time when compared to the HEFT and utility based on profit strategies.

The utility based on profit strategy has the aim of meeting the target response time at the lowest possible cost; if, during the execution of the workflow, it concludes that it is not possible to meet the

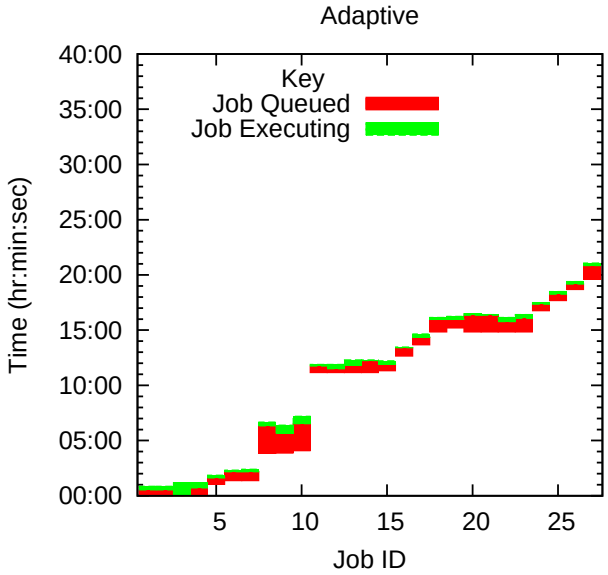


Figure 6. Experiment 1: Montage workflow progress plots, showing when each job (task) is queued and executed, with an adaptive workflow execution strategy using the utility based on response time.

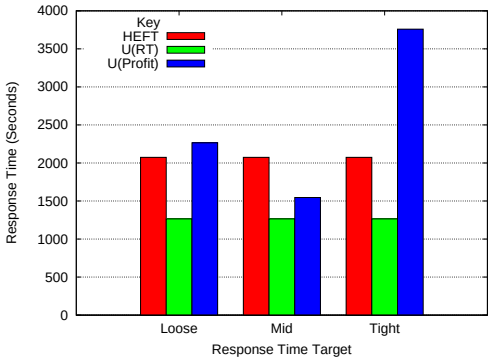


Figure 7. Response Time comparison for Experiment 2

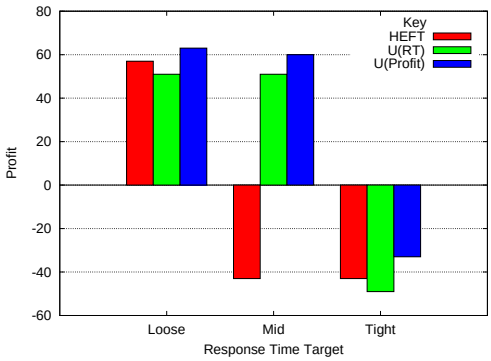


Figure 8. Profit comparison for Experiment 2

target response time it will just use the cheapest resources. All three strategies succeed in meeting the *Loose* target response time and return a profit; the utility based on response time is the fastest with the lowest profit; the utility based on profit uses the cheapest resources, and therefore takes the longest amount of time to complete, producing the most profit.

For the *Mid* target response time, the HEFT-based strategy 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 based on profit strategy. This is because the utility based on profit strategy, although using *ES 1* more often than for the *Loose* response time target, manages to meet the response time while using the inexpensive *ES 2* more than the utility based on response time.

For the *Tight* response time target, none of the strategies meet the target, and thus all make a loss. However, the utility based on 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 1*.

5.4. Experiment 3: Two Workflows

This experiment increases the complexity of the execution environment by considering two Montage workflows, which are planned and submitted for execution at the same time. As before, execution sites *ES 1* and *ES 2* are available and *ES 2* is loaded as described. For this experiment, the *High* target response time is 90 minutes, the *Mid* target response time is 60 minutes and the *Low* target response time is 30 minutes.

Figure 9 shows the individual response times for each workflow, each of the three strategies (HEFT, utility based on response time and utility based on profit) and each of the three response time targets. The first observation from the figure is that, as with Experiment 2, different response time targets clearly have no effect on the non-adaptive HEFT strategy or the utility based on response time; neither of these strategies seek to meet specific response time targets. The utility based on response time always produces the best response time when compared to HEFT and the utility based on profit. Figure

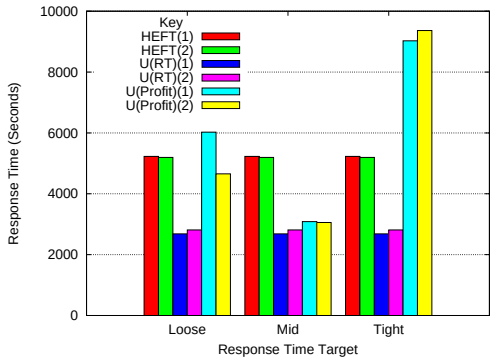


Figure 9. Response Time comparison for individual workflows in Experiment 3

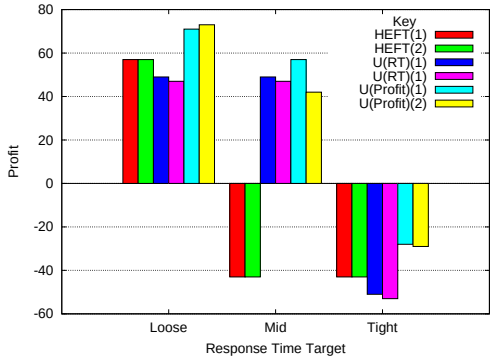


Figure 10. Profit comparison for individual workflows in Experiment 3

9 shows that there is some variation in the response times for each workflow even when they use the same strategy; however, this does not affect the overall result.

Figure 10 shows the individual profit for each strategy and different target response times, with Figure 11 showing the total profit. The total profit is important as the utility based on profit aims to generate the most profit overall.

All three strategies succeed in meeting the *Loose* target response time and return a profit; the individual workflows for each strategy complete very closely as well. The utility based on response time uses the most expensive resources, and therefore returns the least profit. The utility based on profit uses only the resources necessary to meet the target response time, therefore it uses *ES 2* more often than the other strategies, reducing its response time, but increasing its profit while still meeting the

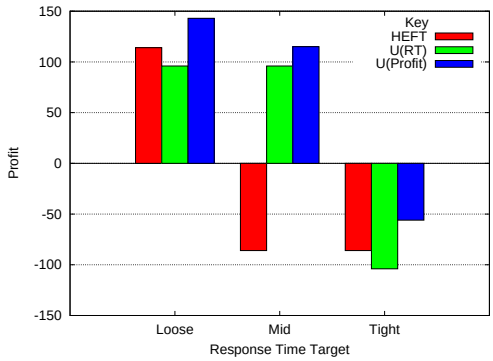


Figure 11. Total Profit comparison for Experiment 3

target response time. The standard HEFT non-adaptive strategy is between the two adaptive strategies in terms of both response time and profit, as it uses *ES 1* and *ES 2* mostly equally.

For the *Mid* target response time, the HEFT based strategy fails to meet the target response time and thus yields a significant loss (as in Experiment 2). Both utility strategies meet the response time target, and yield a profit, but the profit is greater with the utility based on profit strategy. This is because this strategy, although using *ES 1* more often than for the *Loose* response time target, manages to meet the response time while using the inexpensive *ES 2* more often than the utility based on response time. The response time and profit is mostly consistent for individual workflows, with the individual utility based on profit workflows having the most variation. This is a consequence of the utility needing to assign more tasks of one workflow to the faster *ES 1* in order to allow both workflows to complete within the target response time.

For the *Tight* response time target, none of the strategies meet the target, and thus all make a loss. The utility based on response time yields the largest loss as it uses the most expensive resources and still fails to meet the target response time. The utility based on 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 1*.

5.5. Experiment 4: Multiple Workflows

This experiment investigates the scalability of the approach in regards to managing more workflows by extending the number of workflows that need to be executed to 10. Thus, for this experiment ten workflows are planned and submitted for execution at the same time. As with previous experiments, *ES 1* and *ES 2* are available, and a load is applied to *ES 2*. As before, the experiment is repeated for the three strategies, HEFT, U(RT) and U(Profit), and for *Loose*, *Mid* and *Tight* response time targets. For this experiment, the *Loose* target response time is 180 minutes, the *Mid* target response time is 120 minutes and the *Tight* target response time is 60 minutes.

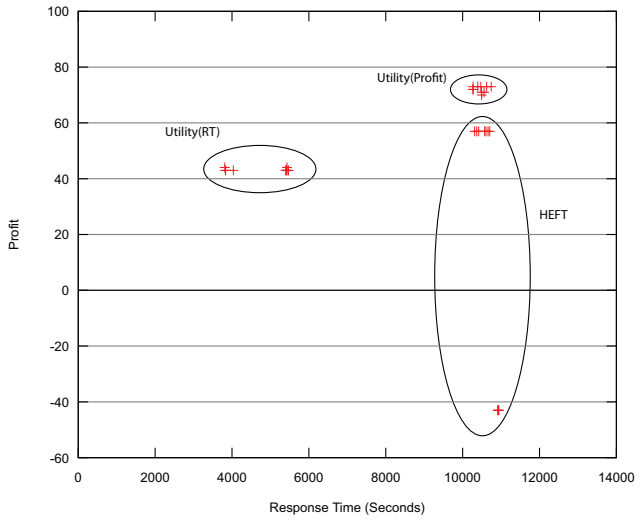


Figure 12. Scatter Graph of Profit versus Response Time for Experiment 4 (Loose target response time)

Figure 12 is a scatter plot showing the results for each workflow, using each of the three workflow execution strategies and a *Loose* target response time. All ten workflows are plotted with response time (Y-axis) against profit (X-Axis).

As can be seen, for the *Loose* target response time, the utility based on response time is by far the fastest strategy with an average response time of 4809 seconds and an average profit of 43.2; it does however have a wide spread of response times, ranging from 3811 seconds to 5468 seconds.

Using the utility based on profit execution strategy, all ten workflows succeed in narrowly meeting the target response time, yielding an average response time of 10459 seconds and an average profit of 72. The high average profit compared to the other approaches is due to the use of cheaper resources.

The non-adaptive HEFT approach fails to meet the target response time for three out of the ten workflows. For the workflows that succeed in meeting the target response time, the profit is higher than the utility based on response time, but lower than the utility based on profit even though their response time is similar. This is due to the utility based on profit adapting to use cheaper resources but the time cost of adaptation negating this effect slightly.

Figure 13 is a scatter plot showing the results of the three strategies with ten workflows and a *Mid* target response time. Both the utility based strategies return a profit, with the utility based on profit returning a larger profit with a slower average response time. All the HEFT workflows fail to meet the target response time, resulting in the slowest average response time and a large loss.

Figure 14 is a scatter plot showing the results of the three strategies with ten workflows and a *Tight* target response time. In this case, all three strategies fail to meet the target response time. The utility based on response time produces the fastest target response time, with the greatest loss. The workflows executing with the non-adaptive HEFT strategy have an increased response time and a marginally

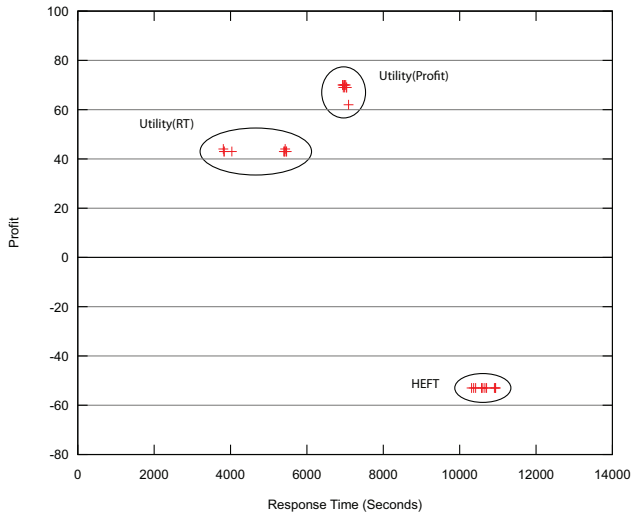


Figure 13. Scatter Graph of Profit versus Response Time for Experiment 4 (Mid target response time)

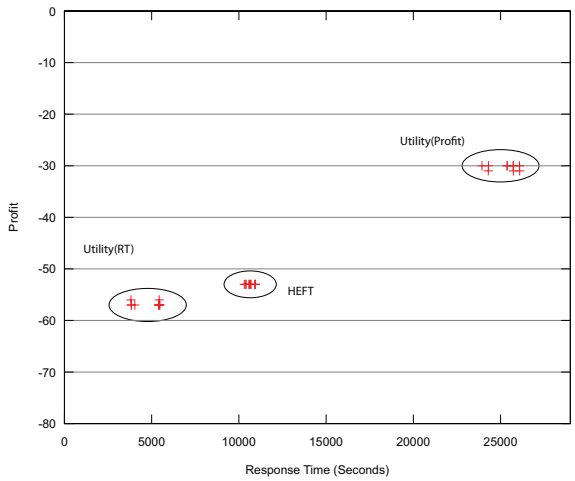


Figure 14. Scatter Graph of Profit versus Response Time for Experiment 4 (Tight target response time)

smaller loss. The workflows executing with the utility based on profit have the least loss and the highest average response time.

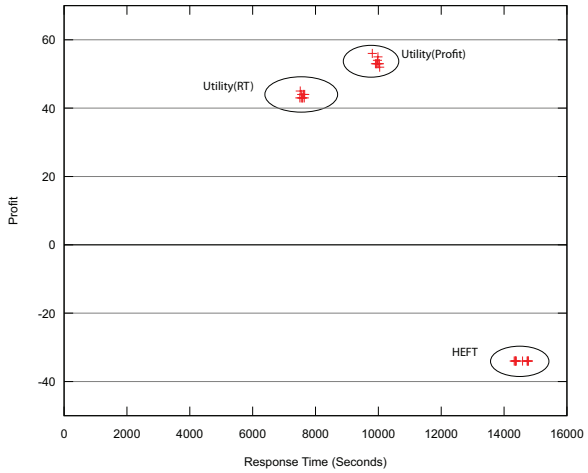


Figure 15. Response Time comparison for individual workflows in Experiment 5

5.6. Experiment 5: Additional Execution sites

This experiment is designed to evaluate the performance of the different approaches with additional execution sites. Four execution sites are used in this experiment, with the third and fourth execution sites, *ES 3* and *ES 4*, having identical specification and software to *ES 2*, as described in Section 5.1. The cost of executing a single job on either *ES 3* or *ES 4* is also equal to the cost of executing on *ES 2*, and it is 1 unit of currency. The same periodic load, as described before, is applied to *ES 2* only.

For this experiment, ten workflows need to be executed and all four sites are available. As with previous experiments, this experiment is repeated for the three strategies, HEFT, utility based on response time and utility based on profit. Also, a *Mid* target response time of 90 minutes is chosen. Each strategy is evaluated in terms of profit and response time as illustrated in Figure 15.

All three strategies have a tight clustering between individual workflows in terms of both profit and response time. The HEFT strategy fails to meet the target response time and thus yields a significant loss. The utility based on response time achieves the best response time, but at the cost of a reduced profit when compared to the utility based on profit which has a slower response time but yields the most profit.

This experiment demonstrates that our utility based adaptive strategies scale both in terms of the number of workflows and the number of execution sites. The spread of the results in this experiment shows that even under these conditions our strategies perform in a consistent manner and yield the desired results. As expected, the utility based on response time adaptive strategy produces the best response time and the utility based on profit adaptive strategy yields the highest profit.

6. Conclusions

We have presented an approach to adaptive workflow execution that: (i) adds adaptive scheduling based on utility functions 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 targets for the execution of workflows, even though the environment provides limited fine-grained control over the execution timing of individual jobs. Adaptive workflow execution 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. This is so because the software components that make up our MAPE-based adaptivity framework are generic and provide instantiation inlets that allow designers the flexibility, at low cost, of experimenting with different adaptation strategies for the same, or different, managed artefacts. This paper has shown the advantages of doing so in the context of workflow execution.

REFERENCES

1. Taylor IJ, Deelman E, Gannon DB, Shields M. *Workflows for e-Science. Scientific Workflows for Grids*. Springer, 2007.
2. Deelman E, Gannon D, Shields M, Taylor I. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems* 2009; **25**(5):528–540.
3. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K. Task scheduling strategies for workflow-based applications in grids. *CCGrid'05*, 2005; 759–767.
4. Wieczorek M, Prodan R, Fahringer T. Scheduling of scientific workflows in the ASKALON grid environment. *SIGMOD Record* September 2005; **34**(3):56–62.
5. Deelman E, Singh G, Su MH, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, *et al.*. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 2005; **13**(3):219–237.
6. Heinis T, Pautasso C, Alonso G. Design and evaluation of an autonomic workflow engine. *ICAC'05: Proceedings of the 2nd International Conference on Autonomic Computing*, IEEE Computer Society, 2005; 27–38.
7. Duan R, Prodan R, Fahringer T. Run-time optimisation of grid workflow applications. *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, IEEE Computer Society Press, 2006; 33–40.
8. Yu Z, Shi W. An adaptive rescheduling strategy for grid workflow applications. *IPDPS*, IEEE Press, 2007; 1–8.
9. Lee K, Sakellariou R, Paton NW, Fernandes AAA. Workflow adaptation as an autonomic computing problem. *WORKS'07: Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science*, ACM Press, 2007; 29–34.
10. Kephart JO, Chess DM. The Vision of Autonomic Computing. *IEEE Computer* 2003; **36**(1):41–50.
11. Kephart JO, Das R. Achieving self-management via utility functions. *IEEE Internet Computing* 2007; **11**(1):40–48.
12. Huebscher MC, McCann JA. An adaptive middleware framework for context-aware applications. *Personal and Ubiquitous Computing* 2006; **10**(1):12–20.
13. Yu J, Buyya R, Tham CK. Cost-based scheduling of scientific workflow application on utility grids. *e-Science*, IEEE Computer Society, 2005; 140–147.
14. Blum C, Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys.* 2003; **35**(3):268–308.
15. Lee K, Paton NW, Sakellariou R, Deelman E, Fernandes AAA, Metha G. Adaptive Workflow Processing and Execution in Pegasus. *3rd International Workshop on Workflow Management and Applications in Grid Environments (WaGe08), in Proceedings of the 3rd International Conference on Grid and Pervasive Computing Symposia/Workshops*, IEEE Computer Society Press, 2008; 99–106.
16. Sakellariou R, Zhao H. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming* 2004; **12**(4):253–262.

17. Hellerstein JL, Diao Y, Parekh S, Tilbury DM. *Feedback Control of Computing Systems*. Wiley, 2004.
18. Diao Y, Hellerstein JL, Storm AJ, Surendra M, Lightstone S, Parekh S, Garcia-Arellano C. Incorporating Cost of Control into the Design of a Load Balancing Controller. *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004; 376–387.
19. Yfoulis C, Gounaris A, Paton NW. An Efficient Load Balancing LQR Controller in Parallel Database Queries under Random Perturbations. *IEEE Multi-conference on Systems and Control*, 2009.
20. Gounaris A, Yfoulis C, Sakellariou R, Dikaiakos MD. A control theoretical approach to self-optimizing block transfer in web service grids. *ACM Transactions on Autonomous and Adaptive Systems* 2008; **3**(2).
21. Menascé DA, Dubey V. Utility-based QoS Brokering in Service Oriented Architectures. *ICWS'07: IEEE International Conference on Web Services*, 2007; 422–430.
22. Walsh WE, Tesauro G, Kephart JO, Das R. Utility functions in autonomic systems. *ICAC'04: Proceedings of the 1st International Conference on Autonomic Computing*, IEEE Press, 2004; 70–77.
23. Kephart JO, Chan H, Das R, Levine DW, Tesauro G, Rawson F, Lefurgy C. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. *ICAC'07: Proceedings of the 4th International Conference on Autonomic Computing*, 2007; 24.
24. Kumar V, Cooper BF, Schwan K. Distributed Stream Management Using Utility-Driven Self-Adaptive Middleware. *ICAC'05: Proceedings of the 2nd International Conference on Autonomic Computing*, 2005; 3–14.
25. Lee K, Paton NW, Sakellariou R, Fernandes AAA. Utility driven adaptive workflow execution. *CCGrid 2009*, 2009.
26. Kwok YK, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys* 1999; **31**(4):406–471.
27. Sakellariou R, Zhao H. A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. *13th Heterogeneous Computing Workshop*, IEEE Computer Society, 2004; 111–123.
28. Nurmi D, Mandal A, Brevik J, Koelbel C, Wolski R, Kennedy K. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. *SC'06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ACM: New York, NY, USA, 2006; 119.
29. Canon LC, Jeannot E, Sakellariou R, Zheng W. Comparative Evaluation of the Robustness of DAG Scheduling Heuristics. *Integrated Research in Grid Computing, CoreGRID Integration Workshop*, Hersonissos, Crete, Greece, 2008; 63–74.
30. Mandal A, Kennedy K, Koelbel C, Marin G, Mellor-Crummey J, Liu B, Johnsson L. Scheduling Strategies for Mapping Application Workflows onto the Grid. *HPDC'05: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, IEEE Computer Society: Washington, DC, USA, 2005; 125–134.
31. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 2002; **13**(3):260–274.
32. Nadeem F, Fahringer T. Predicting the execution time of grid workflow applications through local learning. *SC'09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ACM: New York, NY, USA, 2009; 1–12.
33. Jarvis SA, He L, Spooner DP, Nudd GR. The impact of predictive inaccuracies on execution scheduling. *Performance Evaluation* 2005; **60**(1-4):127 – 139. Performance Modeling and Evaluation of High-Performance Parallel and Distributed Systems.
34. Adve VS, Bagrodia R, Browne JC, Deelman E, Dube A, Houstis EN, Rice JR, Sakellariou R, Sundaram-Stukel DJ, Teller PJ, *et al.*. Poems: End-to-end performance design of large parallel adaptive computational systems. *IEEE Transactions on Software Engineering* 2000; **26**:1027–1048.
35. Feitelson DG, Rudolph L, Schwiegelshohn U. Parallel job scheduling – a status report. *JSSPP'04: Proceedings of the 10th international workshop on Job scheduling strategies for parallel processing*, Springer-Verlag: Berlin, Heidelberg, 2004; 1–16.
36. Decker J, Schneider J. Heuristic scheduling of grid workflows supporting co-allocation and advance reservation. *Cluster Computing and the Grid, IEEE International Symposium on*, IEEE Computer Society: Los Alamitos, CA, USA, 2007; 335–342.
37. Kuo D, Mckeown M. Advance reservation and co-allocation protocol for grid computing. *E-SCIENCE'05: Proceedings of the First International Conference on e-Science and Grid Computing*, IEEE Computer Society: Washington, DC, USA, 2005; 164–171.
38. Zhao H, Sakellariou R. Advance reservation policies for workflows. *JSSPP'06: Proceedings of the 12th International Workshop on Job scheduling Strategies for Parallel Processing*, Springer-Verlag: Berlin, Heidelberg, 2006; 47–67.
39. Margo M, Yoshimoto K, Kovatch PA, Andrews P. Impact of reservations on production job scheduling. *JSSPP'07: Proceedings of the 13th International Workshop on Job scheduling Strategies for Parallel Processing*, Springer-Verlag: Berlin, Heidelberg, 2007; 116–131.
40. Qin J, Wieczorek M, Plankensteiner K, Fahringer T. Towards a Light-weight Workflow Engine in the Askalon Grid Environment. *CoreGRID Symposium*, Springer, 2007; 239–251.

41. Duan R, Prodan R, Fahringer T. Performance and cost optimization for multiple large-scale grid workflow applications. *SC*, 2007; 12.
42. Wieczorek M, Hoheisel A, Prodan R. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems* 2009; **25**(3):237–256.
43. Yu J, Buyya R. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming* 2006; **14**:217–230.
44. Sakellariou R, Zhao H, Tsiakkouri E, Dikaiakos MD. Scheduling workflows with budget constraints. *Integrated Research in Grid Computing*, Springer, 2007; 189–202.
45. Bennani MN, Menascé DA. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. *ICAC'05: Proceedings of the Second International Conference on Autonomic Computing*, IEEE Computer Society Press, 2005; 229–240.
46. Frey J, Tannenbaum T, Livny M, Foster I, Tuecke S. Condor-G: A computation management agent for multi-institutional grids. *HPDC'01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, 2001; 55–63.
47. Arasu A, Babcock B, Babu S, Datar M, Ito K, Nishizawa I, Rosenstein J, Widom J. STREAM: The Stanford Stream Data Manager. *SIGMOD Conference*, 2003; 665.
48. Arasu A, Babu S, Widom J. The CQL continuous query language: semantic foundations and query execution. *VLDB J.* 2006; **15**(2):121–142.
49. Abramson MA, Audet C, Dennis JE. Nonlinear programing with mesh adaptive direct searches. *SIAG/Optimization Views-and-News* 2006; **17**(1):2–11.
50. Audet C, J E Dennis J. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization* 2006; **17**(1):188–217.