Real time policy based management of workflows for SLA Adherence

Daniel Spaven*, Madhavi Rani†, Sumit Kumar Bose†, Mike Fisher*, Srikanth Sundarrajan†
*BT Innovate and Design †SETlabs-Infosys Technologies

Abstract

The need for service providers to offer SLAs to gain consumers confidence has become increasingly apparent. Our work focuses on providers who manage the execution of long-running data-driven grid workflows which are often characterized by user specified deadline constraints. Service providers responsible for managing the execution of such workflows tend to overprovision system resources to reduce the possibility of deadline violations. Further, uncertainty in reliably estimating the QoS parameters makes advanced reservation mechanisms less feasible. The paper proposes an alternate real time management strategy that supports just-in time resource allocation for executing workflows. Towards this end, the paper outlines a scalable and responsive architecture that incorporates policy based management components for allocating resources and integrates monitoring into the workflow itself. The proposed architecture allocates additional resources from the grid pool to the data-parallel operations of the workflow at run-time, when execution delays are observed. These dynamic allocations of resources balance the providers need to satisfy customer SLAs with that of efficiently managing their infrastructure. An experiment was undertaken on an implementation of the architecture to evaluate whether these benefits were realized.

Keywords
Grid workflow, Policy based system, SLA

1 Introduction

Businesses are always looking for ways to improve their productivity and efficiency. Workflows are becoming an increasingly popular method for achieving these goals by specifying, automating and optimising tasks of work. The definition of a workflow according to the Workflow Management Coalition (WfMC) is: "*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant* to another for action, according to a set of procedural rules. *participant = resource* (*human or machine*)". Workflow management systems (WFMS) enable the creation of workflows and facilitate their repeated execution. As workflows have gathered more mainstream popularity the need for service level agreements (SLAs) and their QoS obligations have become more prevalent.

For this work we consider workflows in the scientific domain that comprise of a series of computationally intensive components, analysing large data sets. These workflows are usually long in duration and contain parallelisable components. Our primary focus is real-time management of such workflows so that time-based QoS demands can be fulfilled. We describe our system and a generic architecture that utilizes policy based management to address this problem. Monitoring is integrated into the workflow in the form of events which, when generated, trigger policies that invoke management actions. This enables resource allocation decisions to be made at the most appropriate point; when the resource is actually needed. Management actions control the resource allocation of the workflow and can take advantage of parallelisable aspects of components to maintain QoS adherence.

The structure of the paper is as follows: section 2 explains the motivation for this paper. Section 3 defines the problem, section 4 presents our architecture and describes a reference implementation based on this architecture. Section 5 outlines the related work and section 6 describes our testing scenario and presents the results. Section 7 explains further work that we are interested in exploring. Section 8 concludes the paper with a summary of our findings.

2 Motivation

This section describes why SLAs for long running workflows are beneficial and the importance of managing uncertainty in the WFMS for achieving these benefits.

2.1 Effective SLA management for workflows

To understand the benefits of effective SLA management it is important to explore the motivations of why the customer and the provider would enter into such an agreement. The customer has a need for a service for their own employees or customers. They look to a hosting provider because they don't have the resource capacity in-house, it's not cost effective or perhaps they lack the expertise to manage the service. For the provider the SLA facilitates a business model where the provider can charge

the customer for a managed workflow execution. Their goal is to maximize the difference between the perceived value for the customer and the cost of providing the service. The content of the SLA plays a large part in creating the perceived value. Work by Phillip Masche et. al. [1] describes that the non-functional (QoS) requirements of the service can act as the key differentiator in service choice. Penalties are also identified as being very important in building the user confidence and establishing trust.
The dilemma the provider faces is that they cannot include satisfactory QoS requirements and the worthwhile penalties to differentiate their service because they lack the capability of managing the workflow effectively. To do so they would either have to over provision (reducing their margins) or leave themselves exposed to high penalty payments. Alternatively they are forced to weaken their QoS guarantees and compete on price further reducing their profit margins.
Ideally the provider should be able to monitor the QoS and dynamically change resource allocations to maintain SLA adherence and maximize resource utilisation. They can then compose strong SLAs which will differentiate their service in the market, while retaining high profit margins.

2.2 Uncertainty in QoS estimation

Before a provider can compose a SLA for a workflow the provider should be able to estimate the QoS they can deliver. Any uncertainty that exists in the WFMS can adversely affect these estimates. Uncertainty can be introduced from a number of sources:

Workflow Components: Before constructing the SLA for the whole workflow the provider must break the problem down and consider the QoS they can offer for each workflow component. Performance models can be created that generate probabilities of certain outcomes given input parameters [2]. However, they are limited because even the optimal model only replicates the uncertainty that exists in the real world. This uncertainty may come from non-deterministic algorithms, input parameters only decided at run-time, varying load (user interaction or the size of a data-set). QoS is also heavily reliant on the resources available to the provider, if they are heterogeneous in nature and frequently changing, yet more uncertainty is added to the system.

3^{Rd} party providers: Often workflows are built using third party components to obtain functionality that is not supported in-house. The provider then loses control of governing application level QoS. Trust and reputation mechanisms have been proposed to solve the problem [3] of third party providers failing to meet QoS requirements because of selfish intentions or incompetence.

Network latency and bandwidth: Network performance is often not taken into account by resource managers at the time of deploying applications with QoS requirements. When considering time-based QoS; latency associated with data transfer between components or from external sources, communication latency and deployment location are important considerations that must be taken into account.

System Load: The provider's resource pool has to be managed according to the current demand. When SLAs are offered to the system and capacity is limited, the provider will want to prioritize SLAs that align with their business level objectives.

Lack of control of these factors makes the mapping of tasks to resources very difficult due to inaccuracies when estimating component performance. It becomes apparent that in the context of whole workflows some level of uncertainty is likely and in certain application environments inevitable. Failure to manage uncertainty can adversely affect the provider's long-term income as the provider will be forced to resort to a number of coping strategies. The tendency for the provider will be to over provision; in terms of the specifications of machines and their number allocated to running workflow instances. Alternatively, the provider will be forced to either compose weak SLAs, place tight restrictions on the applications they support or leave themselves exposed to the penalties associated with unwanted SLA violations. A preferable strategy is to recognize that uncertainties are inherent in such a system and hence need to be managed by allocating the resources to the workflow in real-time.

3 Problem Definition

The following paragraphs outline the workflow execution scenario and the QoS metric for real time SLA monitoring, considered in this paper. The section also introduces key notations and definitions needed for subsequent discussions.

3.1 Deadline driven workflows

The most common QoS parameter in scientific workflows is completion time. It is this QoS requirement that we have chosen to illustrate the advantages of our real-time policy based management scheme for resource allocation. There are some domains which are particularly applicable to deadline driven workflows and could benefit from such a management strategy. One such example is in

computational finance where Monte Carlo simulation strategies are widely used. Typical uses for Monte Carlo are in option pricing and evaluating the risk exposure of investment portfolios. This involves running many computationally intensive tasks on different data sets; which makes it very well suited to being executed as a workflow and demands a high confidence level that the result is made available to traders within strict deadlines. Similarly, weather analysis is another scenario where weather simulations are undertaken on a daily basis and need to be completed before the forecast. The need for workflow management systems to model the weather has been addressed by Christopher W Harrop [4], where the need for time-based QoS requirements in production environments has been recognized. Realizing this need, the problem therefore is to architect a workflow execution, monitoring and management system that is both scalable and responsive. By *responsive* we mean that impending QoS violations should be detected as soon as they occur. By *scalable* we mean that the overheads associated with recognizing the 'unusual' events should be minimal. The challenge then is to design a WFMS that is responsive, while keeping management and monitoring costs to a minimum.

3.2 Notation and Definition

Workflow applications can be modelled as a directed acyclic graph consisting of nodes ($t_i \in T$, $1 \leq i \leq N$) and edges ($e \in E$) of the form (t_i, t_j), $1 \leq i,j \leq N$ and $i \neq j$ where t_i is the parent of t_j . The edges represent the data/control dependencies between two adjacent components in the workflow.

Definition-1: Tasks – A task is represented as a node $t_i \in T$, $1 \leq i \leq N$ in the graph.
Definition-2: Data-parallel tasks – Task belonging to the task set $T' \subset T$ for which it is possible to split the input data into multiple partitions and process them in parallel on multiple machines. Each such partition of the input data for the task is called a *chunk*.

Definition-3: Chunk – The input for a data parallel task, t_i $1 \leq i,j \leq N$, can be split into K_{t_i} partitions. Each such partition is called a chunk. The k^{th} chunk $k \in K_{t_i}$ is denoted by t_i^k .

Definition-4: Workflow completion time – The actual time take to complete the workflow execution. It is different from the latest finish time of the workflow which is the deadline as specified in the SLA.

Definition-5: Task completion time – The actual finish time of a task $t_i \in T$, $1 \leq i \leq N$. It is calculated as the $\max_{k \in K_{t_i}} (d_{t_i^k})$ where $d_{t_i^k}$ denotes the completion time of the k^{th} chunk of the task $t_i \in T$, $1 \leq i \leq N$.

Definition-6: Latest finish time of a task – The latest finish time of a task $t_i \in T$, $1 \leq i \leq N$. It is calculated as D_{t_i} $t_i \in T$, $1 \leq i \leq N$.

If $D_{t_i} \geq \max_{k \in K_{t_i}} (d_{t_i^k})$ then the task completion is as per schedule. However, if $D_{t_i} < \max_{k \in K_{t_i}} (d_{t_i^k})$ then the task completion is delayed by an amount of [$\max_{k \in K_{t_i}} (d_{t_i^k})$ - D_{t_i}]. Accordingly, more resources need to be allocated to the task $t_j \in T'$ where t_j is the child of t_i.

4 WFMS Architecture and Implementation

4.1 Architecture overview

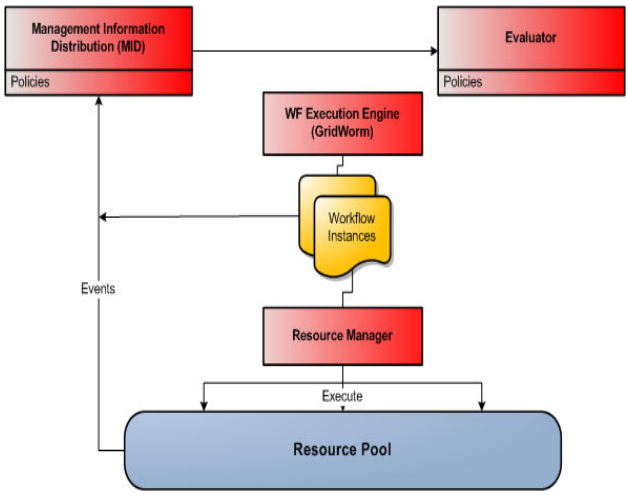

Figure 1. Architecture Diagram

The entry point into the system shown above is the workflow execution engine. Here, the user can create workflows from new components or existing components (including composite components). Events are then annotated into the workflow (described in section 3.3) by the provider for monitoring execution progress. The workflow is then stored in GWLang (an XML based Grid Workflow Language) [5]. GWLang describes the workflow in a platform neutral form that has been

specifically designed for capturing grid workflow descriptions, including QoS requirements. Plug-ins can be added to the workflow engine for each resource manager the system supports, thus acknowledging the need to support resource managers of different types. Each plug-in is responsible for submitting workflow jobs to the resource manager in the appropriate form. The resource manager then schedules each job in accordance to internal policies and the jobs requirements. The job is then executed on the machine in the resource pool that matched those requirements. Workflow specific events are emitted from executed jobs in the workflow itself. Events can also be generated from services monitoring the underlying infrastructure, for example if a machine is not responding. Each event is sent to a URL specified, where a MID is listening. The purpose of the MID is to act as an information filter and route 'interesting' events to the most suitable management entity. In our example implementation this is the Evaluator. The Evaluator contains policies which when triggered invoke a management action (see 4.3). The loosely coupled relationship between the MID, Evaluator and management services facilitates a distributed and scalable architecture. When the system grows in size more instances of the MID and Evaluator can be deployed to manage a subset of workflows. SLA management is simplified because it is undertaken at one location, in the evaluator policies.

4.2 Workflow monitoring event categories

Unforeseen and unpredictable events may disrupt the normal execution of a workflow. This could be due to multiple reasons such as unreliable network, machine failures, patch updates happening concurrently etc. Such incidents tend to slow down the overall execution rate of the workflow. It is pertinent to detect the slowdown in workflow execution soon after such incidents happen. Once a delay has been detected, it is important to allocate more resources either to the currently executing task or to the subsequent task in the workflow in case they belong to the set of data-parallel tasks, $T' \subset T$. Time lost due to execution delays incurred while processing earlier tasks can then be recovered. For this purpose the current work proposes two different event categories:

(*i*) *Checkpoint events*: Assuming that the service provider and consumer agree on the execution deadline for a certain workflow, it is important to determine the completion times of individual tasks $t_i \in T$, and the overall workflow completion time so that execution progress can be monitored. Systems and methods needed to estimate the overall workflow completion times and the individual task execution times are orthogonal to the current work. Since the focus of the present study is to architect a scalable, distributed and responsive monitoring and management system, we assume that the execution time estimates can be reliably obtained. Given the execution time estimates of the individual tasks that make up the workflow and the execution deadline of the workflow (specified as part of the SLA), it is possible to compute the latest start times and the latest finish times (D_{t_i}) of all the tasks in the workflow such that the workflow execution can be completed by its deadline. This analysis is performed before the workflow submission to the grid middleware. On completion of a particular task, we are interested in knowing if the actual completion time ($\max_{k \in K_{t_i}} (d_{t_i^k})$) of the task is greater than the latest finish times computed at the time of workflow submission. The difference, if any, is a measure of the quantum of delay incurred in workflow execution by the time the processing of the task is complete. When the workflow is running behind schedule (i.e. $D_{t_i} < \max_{k \in K_{t_i}} (d_{t_i^k})$), it is important to proportionately allocate more resources to subsequent long running data-intensive tasks (t_j, $j \neq i$) in the workflow so that the workflow execution can be completed within the deadline. This requires instrumenting the task-graph and inserting special constructs – called *check-point jobs*, after each data-intensive operation. Instrumenting and insertion of check-point jobs is done prior to submitting the workflow to the grid middleware. The check-point jobs capture details related to the execution progress made by the workflow up-to the task preceding the check-point job.

```
<workflow xmlns:="http://BT_Infosys.org/Management_Events/2008">
    <checkpoint>
        <checkpointID>C_start_blast</checkpointID>
        <start-time-of-workflow>1241528239062</start-time-of-workflow>
        <ideal-elapsed-time>0</ideal-elapsed-time>
    </checkpoint>
    <currenttime>1241528239062</currenttime>
    <workflowName>BLAST</workflowName>
    <workflowInstance>instance_1</workflowInstance>
  </workflow>
```

Figure 2. Sample XML structure for checkpoint events.

Figure 2 shows the XML structure of a typical checkpoint event. The information captured by the checkpoint jobs include latest finish time of the task preceding the checkpoint job ('ideal-elapsed-time'), the actual finish time for the task preceding the checkpoint job ('current time') and the time when the workflow execution was initiated ('start-time-of-workflow'). This information is forwarded as a *checkpoint event* to the policy based system for taking appropriate corrective action. Instrumenting the workflow in this fashion enables synchronisation between the monitoring and the corrective action. This can be shown by looking at the scenario where the workflow execution is running slowly. If the next component is parallelisable the policy will dictate that more machines are added to the pool. The new

machines have to be made available before the next job is submitted or the scheduler will not consider them for execution. The checkpoint job provides a logical separation that ensures the corrective action will be invoked before the next job is scheduled.

(*ii*) *Task progress events*: Checkpoint jobs by themselves are inadequate as a monitoring mechanism for detecting delays in a timely fashion that might ensure workflow completion by the deadline. Consider the situation where one task in the workflow requires overwhelmingly large amount of processing time compared to the other tasks. Further assume that the execution time for this task as a percentage of the overall completion time is 80%. Having a checkpoint job subsequent to this task may not necessarily help in expediting the workflow execution in case of delays. A more fine-grained monitoring mechanism is required to capture delays arising out of such instances. For detecting delays due to such scenarios, we devise jobs that provide more fine grained information related to the execution progress made by the currently executing task. These are daemon jobs running in the background at pre-specified time instances. If the currently executing task happens to be a data-parallel operation, the job determines the number of data-chunks yet to be processed by observing the grid queue. Accordingly, the job calculates the time needed to process these chunks on machines allocated to the task. In case the calculated finish time exceeds the latest finish time of the task, the job generates an event called *time progress events* (as they are time dependent as opposed to structure dependent in case of checkpoint jobs) for initiating necessary corrective action. Based on this information the resources are scaled up to meet the specified SLA. The *time progress events* have the XML structure as shown in figure 3 and captures details such as latest finish time of the task ('targetfinishtime'), the start time of the task ('actualstarttime'), the number of chunks waiting in the grid queue for processing ('queuedchunks'), the total number of chunks into which the input data for the task was split ('numofchunks').

```
<workflow currenttime="1231240538456" workflowName="BLAST"
workflowInstance="instance_1"
xmlns: ="http://BT_Infosys.org/Management_Events/2008">
    <task>
        <actualstarttime>1231153260896</actualstarttime>
        <targetfinishtime>1231191060896</targetfinishtime>
        <queuedchunks>0</queuedchunks>
        <numofchunks>8</numofchunks>
    </task>
</workflow>
```

Figure 3. Sample XML structure for timer-events

4.3 Management policies

The policies specified in the MID and the Evaluator are loosely based on the policy schema derived by Lionel Sacks et al. [6]. New policies can be loaded dynamically avoiding any down time for these management components. Below is a snippet taken from a policy in the Evaluator triggered by a checkpoint event.

```
<ng:actions>
   <ng:condition>
   (:true() :)
   declare namespace BT_I='http://BT_Infosys.org/Management_Events/2008_12';
   let $start := xs:decimal(data(//BT_I:workflow/BT_I:checkpoint/BT_I:start-time-of-workflow))
   let $now := xs:decimal(data(//BT_I:workflow/BT_I:currenttime))
   let $target := xs:decimal(data(//BT_I:workflow/BT_I:checkpoint/BT_I:ideal-finish-time))
   let $ideal_elapsed := $target - $start
   let $thresholdFactor := 1.1
   return ($target - $now) gt ($ideal_elapsed * $thresholdFactor)
   </ng:condition>
   <ng:action>
      <ng:target>
         http://localhost:8080/org.nextgrid.omf.evaluator
      </ng:target>
      <ng:data>
         <ng-omf-ea:ActionData
            xmlns:ng-omf-ea="http://www.nextgrid.org/2006/11/omf/evaluator/action">
            <Action>
               http://BT_Infosys.org/Management_Actions/2008_12:CondorIncreaseCapacity
            </Action>
            <Parameters>
               <resinfo>http://dg03.grid.bt.co.uk:8080/.../ResInfoServiceSOAP</resinfo>
               <condormanager>dg02.grid.bt.co.uk:6543</condormanager>
               <MID>dg03.grid.bt.co.uk:9034</MID>
            </Parameters>
         </ng-omf-ea:ActionData>
      </ng:data>
   </ng:action>
```

Figure 4. Actions tag content

The condition is written in Xquery and can reference values from within the policy and the triggering event. In this case the start time of the workflow, current time and target time (for QoS) are extracted from the event. The condition is evaluated to true when the actual elapsed time is greater than the ideal elapsed time multiplied by a threshold. So in this example if the workflow is 10% behind schedule the management action is invoked. The 'Action' tag points to by a URL that references an executable piece of code. A number of parameters can also be passed to the executable within the 'Parameters' tag. In this case the action is to increase the capacity of the condor pool.

4.4 Workflow Manager

Workflow manager facilitates the execution of workflows over the grid. It has a pluggable and service oriented architecture which allows integration with various grid middleware products like Condor, PBS, LSF. The workflow manager leverages the vast computational capacity offered by grid to execute workflows. It is capable of executing hundreds of workflows simultaneously and provides control to the user for

managing the workflow. It is best suited for executing applications which operate on huge amount of data. The data-splitting feature of the workflow manager can be leveraged for reducing the execution time of data intensive applications. The reduction in the execution time is achieved by splitting the data into multiple chunks and distributing these chunks amongst various nodes available in the grid for execution [5]. The main components that constitute the workflow manager, as shown in figure 5, are:

WorMServer: WorMServer orchestrates the execution of the workflows. WorMServer comprises of the following components:

Workflow Management Service: This is a webservice providing the user controls for managing and monitoring the workflow.

Workflow Scheduler service: This service is responsible for scheduling the workflows for execution based on their recurrence pattern.

Preparser: This component provides the workflow with the details of the subflows and customized processes referred in it. In addition to this, it also does some pre-execution tasks like database updates, workflow validations etc.

State Engine: Workflows' are executed in the state engine. It identifies the dependencies amongst the elements of a workflow and processes them. It also handles the failover and exceptions that might occur during a workflow's execution.

JobManager: JobManager manages the lifecycle of jobs submitted to MAGI for execution. It performs operations such as submitting the jobs to MAGI, getting status reports and terminating the jobs when necessary. The State Engine interacts with this component for carrying out the execution of jobs.

GWlang Generator: Workflows are represented based on GWLang (an XML based Grid Workflow Language) specifications. GWLang Generator provides an API for modelling workflows. This component is used for creating workflows and also by the WorMServer to interpret workflows.

GUI: GridScape- Workflow manager provides a rich user interface for workflow creation, submission, execution and monitoring. The GUI interacts with the GWLang Generator to convert the graphical workflows into GWLang format. The GUI invokes the Workflow Management Service to perform all the workflow management operations.

MAGI: MAGI (Management of Adaptive Grid Infrastructure) is a meta-scheduler for the grid. It aids GridWorM in executing jobs over the grid and shields the workflow manager from the heterogeneity of the underlying Grid Middleware.

5 Related work

Much of the previous work on SLA aware workflows has focused on QoS aware scheduling [7]. These consider the tasks functional and non-functional requirements (QoS), however a static mapping is created prior to execution, of tasks onto resources. Recently more emphasis has been placed upon managing workflows in real-time. For example, [8] proposes fault tolerant workflow systems in case of service failures. Work on workflow composition and optimisation regarding time and cost constraints has also been expanded upon to include dynamic optimisation algorithms [9]. Little work has been based on workflow monitoring and how the necessary information is distributed to the appropriate parties responsible for real-time management of QoS requirements in the context of a whole WFMS. Brandic et. al. [11] discussed the real time monitoring of deadline constrained workflows. One notable piece of work was part of the GRIDCC project [10], where they built a comprehensive WFMS that considered real-time management and monitoring. A monitoring web service was included as part of the workflow. However, it relies on clients calling the web service which would have to be a separately managed process. We argue that all monitoring, as it utilizes resources and is workflow dependent should be included in the workflow but in a generic fashion not dependent on the workflow engine. We feel our solution also offers increased scalability as the workflow engine is not used to query the workflows status or execute management functions, thus avoiding performance issues.

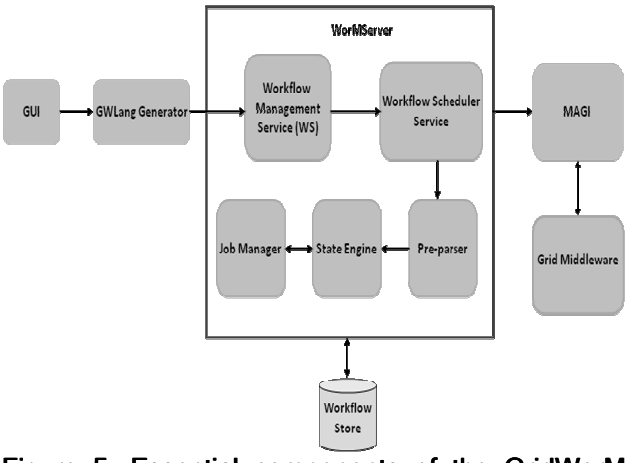

Figure 5. Essential components of the GridWorM architecture

6 Experimentation

The following paragraphs describe the sample workflow application that was designed using the workflow manager for execution on grid environments. This is followed by the testing scenario and results. For our testing environment the workflow manager and the Condor head node was installed on an Intel Pentium 4 CPU 3.2GHz. Each worker node was installed on Dell PowerEdge Servers running the Intel Xeon Processor 2.8 GHz. Each had condor execute capability and was connected to the head node via Gigabit Ethernet.

6.1 Workflow Description

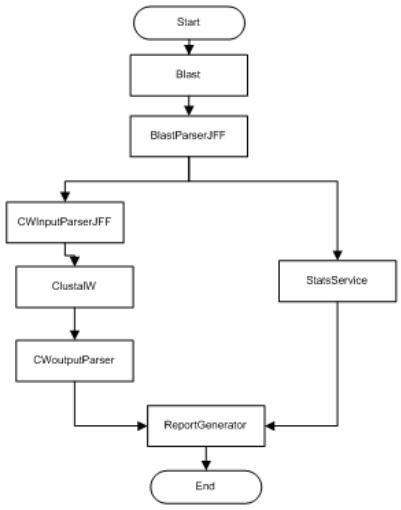

Figure 6. Example workflow

A drug discovery pipeline from life sciences shown in figure 6, is taken as an example to demonstrate the efficacy of our work. The workflow begins with executing the BLAST application which queries the NCBI (National centre for Biotechnology Information) database to filter out the nucleotide sequences with close resemblance to the input sequence. An analysis is then carried out, as part of BlastParser on this result to evaluate the percentage of similarity based on the score and the e-value of each matched sequence. CWInputparser makes use of the same input sequence to determine its functional role and the organism from which it originates. StatsService generates a report which lists the sequences that are similar to the input sequence and the statistical information related to similarity to their source of origin. ClustalW is then executed to align all the sequences with the input nucleotide. CWOutputparser performs a statistical analysis on the aligned results to validate the output. Finally, ReportGenerator generates a report detailing the gene family of the input nucleotide sequence and relevant metrics to depict its biological significance. Events are processed after each task, effectively making each step a composite task.

All the Jobs in the workflow other than StatsService require Linux environment for their execution. StatsService does not have any operational constraints associated with it and can be executed on either Windows or Linux environments

6.2 Test Results

Results were captured for three different scenarios – (1) the resource pool has only one worker node, which is responsible for executing all the tasks of the workflow. (2) The resource pool has four worker nodes, which share the activity of executing the workflow. (3) Using real time allocation of worker nodes. In this case the number of worker nodes that executes the workflow is dependent on the time and progress made by the workflow. The SLA specified the completion time had to be within 10% of the target time. The target completion time was 24 minutes, therefore any workflow running more than 00:26:24 (hr:min:sec) will break the SLA.
Table 1 shows the average completion time and standard deviation over five runs. During testing the maximum number of execution streams for the workflow was four (including parallelized tasks). Using one worker node violated the SLA on each occasion and executed the workflow in the longest times. Using four worker nodes consistently recorded the shortest completion times. The real time allocation instances all completed the workflow just inside the time required for SLA adherence. The standard deviation was calculated. It is noticeable that the real time allocation scenario has a much narrower distribution of completion times.

Table 1.General test results

Execution instance	1 worker nodes	4 worker nodes	Real time allocation
1	00:29:34	00:19:40	00:25:14
2	00:33:16	00:23:21	00:24:43
3	00:27:00	00:20:52	00:24:31
4	00:29:33	00:20:55	00:25:34
5	00:26:29	00:21:28	00:24:51
Average	00:29:10	00:21:15	00:24:59
Standard Deviation	00:02:25	00:01:12	00:00:23

The tables 2 and 3 show in detail the completion times of each component for the first execution instances. In the real time management scenario during the BlastparserJFF step the current time fell more than 10% behind the target time and 2 machines were added. This enabled the workflow to catch up in the parallelisable JFF steps. The real time allocation implementation showed a nice

balance between executing within the deadline and minimising resources used, with an average of just 2.25 worker nodes allocated at one time.

Table 2. Detailed execution instance results for 1 worker and 2 worker scenarios

	1 worker		4 worker nodes	
Component	Completion time	Duration	Completion time	Duration
BLAST	00:05:55	00:05:55	00:04:34	00:04:34
BlastParserJFF	00:10:42	00:03:52	00:07:05	00:02:19
StatsService	00:13:32	00:02:25	00:08:45	00:01:28
CWInputParserJFF	00:23:33	00:12:25	00:15:05	00:07:46
ClustalW	00:25:34	00:01:37	00:16:37	00:01:20
CWOutputParser	00:27:40	00:01:45	00:18:05	00:01:17
ReportGenerator	00:29:34	00:01:35	00:19:40	00:01:22

Table 3. Detailed execution instance results for real time allocation

	Real time allocation		
Component	completion time	Duration	Total Allocated machines
BLAST	00:06:07	00:06:07	1
BlastParserJFF	00:09:38	00:03:12	3
StatsService	00:13:09	00:06:42	3
CWInputParserJFF	00:18:45	00:08:38	3
ClustalW	00:21:01	00:01:56	3
CWOutputParser	00:23:10	00:01:47	3
ReportGenerator	00:25:14	00:01:38	0
	Average allocation: 2.25		

7 Further Work

At present events are annotated to the end of workflow components and are scheduled separately. Events need to be suitably integrated with components to avoid this extra scheduling overhead. Further, the policies are currently created manually and fed into the system. An interesting area of work would be to dynamically generate policies. This would entail parsing and translating the information contained in the workflow description (GWLang) and the SLA and automatically generating the policies in the MID and Evaluator.

A comprehensive SLA management layer would further enhance the system. It would have a view of the goals of the provider, systems resources, live SLAs and customers. It would operate an acceptance control mechanism that accepts or rejects new SLAs based on their potential profitability and current resource capacity. At times of resource contention this layer would make intelligent decisions that align with the providers business objectives, when allocating resources.

8 Conclusion

The paper described a distributed, scalable architecture that utilizes policy based management techniques to monitor and manage grid based workflows. Monitoring of QoS requirements is incorporated into the workflow and management actions are triggered in real time to ensure SLA adherence. The proposed architecture assigns resources to the tasks in the workflow based on the quantum of delay measured by the WFMS. Two different event monitoring categories have been proposed for facilitating detection of delays. Task-Progress event category has been designed to detect delays during the execution of workflow tasks. Checkpoint events monitor the progress of the each execution stream. Policies associated with these event monitoring categories contain parameters that can be suitably adjusted for designing a highly responsive real time monitoring and management system. The WFMS architecture was tested and demonstrated complete SLA conformance while maintaining low levels of resource utilisation.

9 References

[1]. Philipp Masche, Bryce Mitchell, and Paul Mckee, "The increasing role of SLAs in B2B", In Proceedings of the 2nd international conference on web information systems and technologies, Setubal, Portugal, 2006.

[2]. Jane Hillston, "A compositional approach to performance modelling", Cambridge University Press, 1996

[3]. Yao Wang, Julita Vassileva, "A Review on Trust and Reputation for Web Service Selection", 27th International Conference on Distributed Computing Systems Workshops, 2007

[4]. Christopher W. Harrop et. al., "A Workflow Management System for Automating Weather and Climate Simulations", CIRES' Annual, Institute-wide Symposium

[5]. Keyur Gor et. al., "Scalable enterprise level workflow and infrastructure management in a grid computing environment", In Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid), 2005.

[6]. Lionel Sacks et al. "Active Robust Resource Management in Cluster Computing Using Policies", Journal of Network and Systems Management, Vol. 11, No. 3, September 2003.

[7]. J. Yu, R. Buyya, C. K. Tham, "QoS-based scheduling of workflow applications on service grids", Conference on e-Science and Grid Computing (e-Science), 2005

[8]. S. Stein, T. R. Payne, N. R. Jennings, “Flexible provisioning of web service work flows”, ACM Transactions on Internet Technology (TOIT), Volume 9 (1), 2009.

[9]. H Wanek, E Schikuta, IU Haq, “Grid workflow optimization regarding dynamically changing resources and conditions”, Grid and Cooperative Computing (GCC), 2007.

[10]. A. S. McGough, et. al., "GRIDCC: Real-time workflow system", In Proceedings of the 2nd International Workshop on Workflows in Support of Large-scale Science, 2007.

[11]. I. Brandic, S. Benkner, G. Engelbrecht, R. Schmidt, "QoS support for time-critical grid workflow applications", 1st International Conference on e-Science and Grid Computing, 2005.