Performance-aware Workflow Management for Grid Computing

D.P. Spooner[1], J. Cao[2], S.A. Jarvis[1], L. He[1] and G.R. Nudd[1]

[1]*Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK*
[2]*Center for Space Research, Massachusetts Institute of Technology, Cambridge, USA*
Email: dps@dcs.warwick.ac.uk

Grid middleware development has advanced rapidly over the past few years to support component-based programming models and service-orientated architectures. This is most evident with the forthcoming release of the Globus toolkit (GT4) which represents a convergence of concepts (and standards) from the web services community. Grid applications are increasingly modular, composed of workflow descriptions that can feature both resource and application dynamism. To manage workflow effectively, it is advantageous to understand the performance implications of executing individual tasks in a given configuration to ensure that specific qualities of service are achieved across the entire flow. In this work, a multi-tiered workflow management system is described. The system couples a performance modelling tool with local and global scheduling algorithms that aim to meet user-specified deadlines while resolving user and resource conflicts.

Keywords: Workflow Management, Grid Middleware, Performance Modelling

1. INTRODUCTION

Grid computing has undergone a number of significant changes in a relatively brief time-frame [1]. With tremendous community effort, supporting grid middleware has expanded significantly from rudimentary batch processing front-ends to fully distributed components with complex scheduling, reservation and information sharing facilities. Central to this evolutionary process have been the releases of the Globus toolkit [2], which represents the de-facto platform for grid applications. With the imminent release of GT4, grid computing moves closer towards web services frameworks [3], where applications are increasingly dynamic, modular and service-orientated.

Component-based systems typically require workflow descriptions that reflect both organisational and technical boundaries. Applications may span multiple administrative domains in order to obtain specific data or to utilise specific processing capabilities. Equally, applications may wish to select components from a particular domain to increase throughput or reduce execution costs. In the grid context for example, an application may have limited choice in data acquisition (possibly requiring a particular type of instrumentation), but more scope for data post-processing (which requires a cluster of commodity processors). A further level of decomposition may exist within the organisation or domain, where individual task atoms are composed to provide the overall service.

Managing workflow is complex and currently the subject of many research projects. The reasons for this are manifold: component architectures encourage code re-use, the ability to build dynamic applications is attractive to active middleware developers and it is a natural way to utilise services in a manner that promotes inter-organisation collaboration. Much of the existing work focuses on workflow languages and how they describes component features and facilities. These languages typically support a variety of operators to express the workflow, with methods to compile (or translate) these descriptions into a working application. In addition, they often describe various input and output conventions used to inter-connect components.

The migration to component-based systems, orchestrated by a workflow management system, has interesting consequences for the performance community. First, the performance implications will become increasingly difficult to evaluate as applications are constructed from components that are deeply de-coupled and widely distributed. It will require an understanding of the impact of executing complex applications on different architectures under dynamic load conditions, as well as the communication and data-transfer effects. There are a number of performance evaluation techniques that are useful and can be applied to this type of activity including simulation-based approaches [8] and analytical modelling techniques [5, 6]. In certain conditions, historical data is applicable, although usually a hybrid approach is required to allow some form of performance parameterisation [4].

A second performance criterion relates to a strength-

ening of the quality-of-service (QoS) characteristics and the implications to the anticipated 'grid user'. Grid users were traditionally perceived as being members of academic scientific collaborations with relatively straightforward requirements in terms of service quality. As the commercial features of web services have influenced Grid middleware design, QoS support has been extended to offer a range of additional facilities. Timeliness, response and experimental accuracy may be differentiating factors for grid resource providers and future grid users. These factors are closely associated with grid performability [7].

Grid workload management systems can be considered as a supplementary layer to existing grid schedulers, an area of research that is well established in the grid community. One such system, Pegasus [9], generates scripts that Condor's [10] DAGman [11] can use in order to execute local tasks. Layering systems in this manner is often seen in grid work and a customary approach while standards are agreed. In the interim, a number of systems (such as ICENI [12]) use proprietary mechanisms to handle workflow, and include interactive workflow designers, solvers and execution modules to build, plan and execute applications end-to-end. As standard languages develop, it is important to consider the performance effects at all stages in a component's lifetime, including the workflow actuation phase. As evident in work on grid scheduling, significant gains to service quality can be made when application and user behaviour is predicted.

The system described in this paper is an extension to an existing grid scheduling system [13] that maps tasks to appropriate resources based on their expected performance behaviour. The system does not prescribe any particular workflow language, neither is it tied to any particular implementation. The purpose of the scheduler is to predict how an application workflow will behave *a priori*, permitting stronger assertions regarding quality of service. The research described in this paper uses an analytical modelling system [6] to anticipate component behaviour given a particular resource configuration. This assumes that the models are sufficiently parameterised and produce performance data of reasonable fidelity. While these factors can be problematic when users submit custom code (as in batch processing systems) the use of components is helpful in this context as high quality performance models can be written by a performance specialist when the component is developed.

The paper is organised as follows: section 2 introduces the grid workflow management problem and describes the Warwick architecture and the supporting performance service; section 3 includes a detailed account of the local and global scheduling algorithms. Section 4 evaluates the system and provides supporting experimental results. Related work and conclusions are documented in sections 5 and 6.

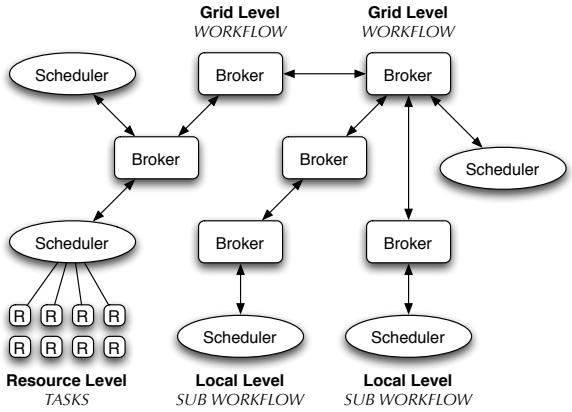


FIGURE 1. The multi-tiered workflow management architecture. Grid level is used to describe inter-domain resource balancing. Local level is used to describe the mapping of tasks to resources.

2. GRID WORKFLOW MANAGEMENT

Workflow management systems have been the subject of research for a number of years (see [9, 11, 14, 15]), but have grown in popularity since the advent of web services and more recently grid computing. Pioneering work in the grid community has allowed applications to be distributed across different domains to form virtualised applications. This promotes collaboration, a key goal in grid computing, as it improves the ability to co-ordinate resources and share data. The challenges with grid workflow involve the traditional problems related to wide-scale virtualisation: cross-domain management, where processes in the grid traverse different domains of administration; and dynamic state, where the availability and performance of grid resources may vary significantly over time.

Maintaining QoS targets for a flow of tasks across different domains can create resource and user conflicts that are potentially more severe than in single task scheduling. User conflicts can occur when a user in one domain have a priority that does not equate to a priority in a foreign domain, or in which there is not the same level of privilege. In this case it is necessary to map user classes from one domain to another based on a set of definable characteristics. Inevitably, this sort of mapping is the responsibility of the domain administrator and depends on the required behaviour. Similarly, resource conflicts can occur when tasks from one workflow require systems in use by another workflow.

Previous work at Warwick addressed similar issues with the development of multi-tiered workload management system that consists of a *local* scheduler [13] for managing applications on physical resources and a *grid* scheduler [16, 17] that uses a hierarchy of brokers to load balance across distributed domains. Performance models were developed for both scheduled applications and the underlying resources to assist in workload place-

ment decisions. This system, known as TITAN, provides a solid basis for managing more complex applications with associated topologies (workflow). The evaluation of these performance models provides a level of performance awareness that can be used to enhance service quality and the two-tiered architecture assists in handling workflows across domains.

The distinction between handling tasks at the local and grid level is crucial: local resources must be managed in order to obtain good resource utilisation and throughput. It is the responsibility of the local scheduler to map individual tasks to resources. A grid level scheduler operates across domains to move task sequences to suitable resources that can meet service quality constraints. This is used to provide a general load balancing effect where faster architectures can accommodate a greater proportion of the workload. Figure 1 illustrates the system architecture with brokers at the grid level, schedulers at the local level and resources as the underlying processing systems.

Local resource owners are interested in their own systems and how these resources are utilised. However, the emphasis (or balance) of system parameters may be different depending on the particular domain. One environment might offer a system that aims to move jobs through the system rapidly (high throughput), with users treated on an equal basis. Individual task deadlines may be considered less important than the overall system throughput. A second environment may have different domain criteria: deadlines should always be met (or at least a good attempt undertaken) with a reasonably good throughput, otherwise the domain is limited on how many customers can be accommodated. The local system developed in this work is based on a genetic algorithm (GA) that uses a fitness function that can be adapted to suit various requirements for scheduling. A tuple of deadline, idle-time and end-to-end time is weighted to provide a single penalty measure that can select 'good quality' schedules from a population of candidate solutions. The GA uses a coding scheme that facilities interdependent relationships and workflow, while the deadlines are based on penalty cost functions that are shaped on 'user class'.

Management at the grid level faces a slightly different problem. Brokers are used to exchange tasks to create a load balancing effect and advertise the capabilities of their local domains to other brokers. A compromise exists between executing tasks locally (possibly for economic or contractual reasons) as well as locating more appropriate remote resources that might better meet deadlines. There are a number of higher level issues involved with multi-domain management that are not considered in this work including trust, allegiances and economic cost models [18]. Performance models can still assist at this level as they allow brokers to speculate on the potential performance of a remote resource, given appropriate parameterisation. In a workflow scenario this is important as the consequences of misplacement can have far-reaching effects across different domains.

2.1. Workflow Definition

In this work, workflows are considered both abstractly and hierarchically. At the atomic level are *tasks* that are composed into intra-domain applications referred to as *sub-workflows*. Sub-workflows can then be connected into *workflows* that represent the entire application. Relationships between tasks and sub-workflows include continuation operators, forks and joins. In practice an XML script is used to describe workflows and this is based on the ANT Java build tool [19], which provides a mechanism for representing runnable applications with dependencies. The choice of workflow language is independent of this work; here we concentrate on the performance model and scheduling characteristics.

Tasks: Tasks are the basic building blocks of the application in the grid workflow. They are the individual components or jobs that are executed on a local grid resource. They are typically MPI tasks that execute on multiple processors and can be *shaped* by the GA to change the level of parallelism. By selecting an appropriate mapping for the task, good quality schedules can be constructed that meet the various QoS requirements.

Sub-workflows: A sub-workflow is a sequence of closely related tasks that are executed in a predefined order on local grid resources. This typically occurs when there is significant communication between two components and where it is unlikely that decomposition across domains will improve application response. Again, the GA has some freedom to reorganise the task order as long as it does not change the application's task precedence. Conflicts occur when tasks from different sub-workflows require the same resource simultaneously, and it is the responsibility of the GA to manage this interleaving.

Workflows: A grid application can be represented as a sequence of several different activities represented by sub-workflows. These activities are loosely coupled and may require multi-sited grid resources.

Figure 2 illustrates the structure of the TITAN workflow management system. Workflow descriptions, which define each of the tasks with clearly defined pre- and post-activities, enter the system at the portal level and are passed to a broker. The broker runs an initial execution simulation of the workflow using the local performance service and neighbouring brokers to develop a preliminary schedule. The simulation results can be returned to the portal for user agreement or executed directly.

The actual execution of the workflow may diverge from the simulation due to the dynamic nature of the grid environment. Where significant changes have

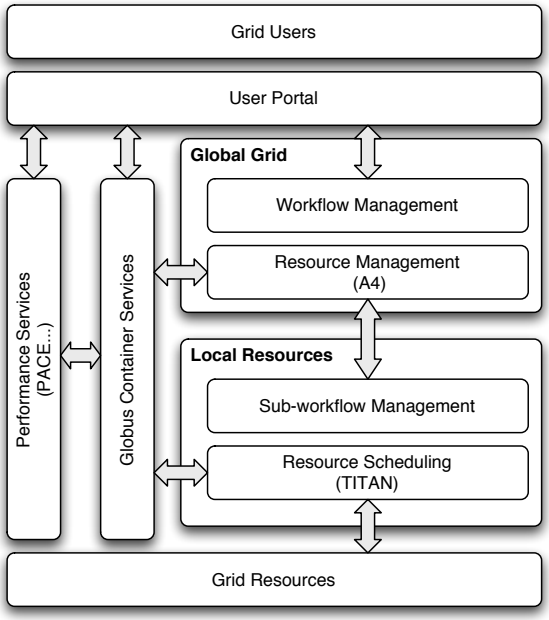


FIGURE 2. Key components in the TITAN architecture. Workflow descriptions are submitted to a broker, which locates a suitable resource set and passes the sub-workflow scripts to a local scheduler.

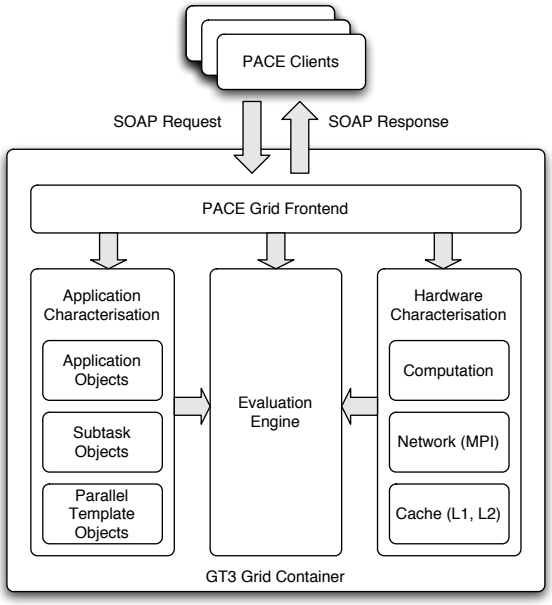


FIGURE 3. Key PACE components. Clients connect to a PACE *performance service* which runs inside an OGSA/OGSI container. This in turn compiles, caches and evaluates model objects using the PACE back-end tools.

occurred, the workflow may be sent back to the simulation engine and rescheduled. Scheduling the sub-workflow at the local resource level is similar to the global process except that the local grid sub-workflow scheduling has to deal with multiple tasks that may belong to different sub-workflows. The execution times have to be estimated with the extra consideration of conflicts, which may occur when multiple tasks require the same grid resource at the same time.

2.2. The PACE Performance Service

Central to this work is ability to anticipate how components will behave on a particular architecture in a given configuration. The Performance Analysis and Characterisation Environment (PACE [6]) is used to develop high-fidelity models that are used to drive the decision making processes at all levels in the system. While the accuracy of the models has a direct influence on how effective the scheduling system is, it is possible to develop good quality models due to the reusable nature of component-based systems.

The PACE toolkit is based around an evaluation engine that takes separate resource and application models as inputs and is able to predict the execution time of a task prior to run time. A task's scalability (execution time vs. level of parallelism) can be determined using PACE. In the majority of cases, the product of a tasks execution time and the number of processors it occupies are not constant and so PACE can be used to explore different run-time scenarios. This is useful for strong-scaling applications to prevent them over-occupying a local resource by using more processors than is necessary and is especially apparent where the execution time may not be inversely proportional to the allocated size due to the presence of communication among components.

Due to the separation of resource and application models, it is possible for brokers to execute 'what-if' scenarios to identify how a sub-workflow will behave on a foreign architecture prior to runtime. This can be used to construct a preliminary workflow schedule. Each broker makes a resource model available as an 'advert' of its local scheduler. It can then share this model with neighbouring brokers.

To integrate with existing grid middleware, PACE is made available as a grid service through a front-end that lies between the existing toolkit and grid clients (including TITAN). Clients are able to invoke models and query the PACE evaluation engine using this interface, which currently runs inside a GT3 container. The performance service itself consists of a grid service factory [20], which can create application model instances that encapsulate the computation and communication behaviour inside a component using a combination of static code analysis and instrumentation. Resource descriptions are also installed which model the processing capability, cache behaviour and network performance. When combined, the models form an evaluation engine instance with which clients interact to obtain the relevant performance information.

The instance can be used by passing model

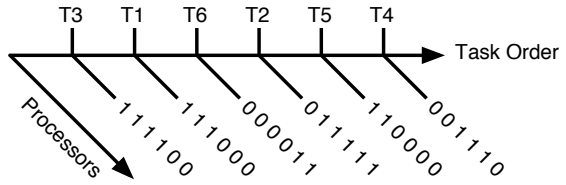


FIGURE 4. 2D coding scheme which forms the basis of the GA evaluation and evolutionary phases. Task orders are represented as an m-ary array of task identifiers, with a corresponding binary representation in the processor dimension.

parameters to a `queryModel` function that returns the expected execution time. A fundamental property is the expected number of processors. The GA adjusts this parameter in order to improve throughput, utilisation and deadline response by improving the interleaving of each sub-workflow. Where appropriate, other variables can be modified by the GA, this includes application parameters that affect runtime, such as the resolution of a grid in a weak-scaled code for example.

Figure 3 illustrates the structure of the PACE toolkit and its architecture as a grid service. Multiple clients can connect through the container to install application and resource models as well as communicate with the evaluation engine.

3. THE GRID WORKFLOW SYSTEM

This section introduces the workflow management system that is responsible for analysing work flow descriptions and apportioning them to particular resources. Initially a local level algorithm is introduced, this shapes the tasks in a sub-workflow so that it uses appropriate resources within a cluster as guided by the analytical performance models. At this level it is necessary to ensure that tasks are run in an appropriate order (depending on the sub-workflow) and that deadlines and inter-dependant relationships are maintained. Following the local level, the grid level load balancing algorithm is described. The grid-level works across domains and makes use of a different algorithm which aims to find appropriate resources (not necessary the best) using a system of relaxing deadlines.

3.1. The Genetic Algorithm

A genetic algorithm is used by TITAN which forms the basis of the local workflow system. A two dimensional coding scheme is used to represent a schedule of parallel jobs in a cluster. Each column in the coding scheme specifies the allocation of processing nodes to a parallel job, while the order of these columns in the coding is the *preferred* sequence in which the corresponding jobs are to be executed. An example is given in Figure 4 and illustrates the coding for a small (6 task) schedule with associated processor mappings.

The task order sequence specified in the coding is termed the *preferential* task order as it may not represent the actual task sequence in time. Earlier task allocations that utilise resources that a current task requires may prevent the task from running, allowing a later sequenced task to operate instead. Such artefacts are discovered when the coding is transformed into the time domain as part of the schedule composition process. The coding scheme in Figure 4, for example, contains a blocked processor so that task 6 executes earlier than task 1, despite being later in the coding sequence. This transformation is also used to handle task dependencies and inter-task relationships.

In order to transform the coding into time, each task identifier (with its allocated processor maps) is passed to the PACE performance service to obtain an estimation of the application runtime. The expected time and host allocations can then be used to build a schedule for subsequent evaluation. A schedule is evaluated using three metrics: cumulative over-deadline, schedule end-to-end time and cumulative idle-time. These metrics are combined with domain-controlled weights to form a single comprehensive penalty metric (denoted by CP). CP is defined in Equation 1 where Γ, ω and θ are makespan, idle-time and over-deadline, respectively; and W^{i}, W^{m} and W^{c} are their associated weights. For a given weight combination, the lower the value of CP, the better the comprehensive performance of the schedule.

$$CP_k = \frac{(W^{\mathrm{i}} * \Gamma_k) + (W^{\mathrm{m}} * \omega_k) + (W^{\mathrm{c}} * \theta_k)}{W^{\mathrm{i}} + W^{\mathrm{m}} + W^{\mathrm{c}}} \qquad (1)$$

TITAN uses the GA to find a schedule with a low CP. The algorithm first generates a set of random schedules with what is felt are reasonable properties (early deadline first etc.). The performance of a schedule, evaluated by the CP, is normalised to a fitness function using a dynamic scaling technique which is shown in Equation 2: where CP_{min} and CP_{max} represent the best and the worst comprehensive performance in the schedule solution set; and CP_k is the penalty of the k-th schedule.

$$f_k = \frac{CP_{\mathrm{max}} - CP_{\mathrm{k}}}{CP_{\mathrm{max}} - CP_{\mathrm{min}}} \qquad (2)$$

The GA uses f_k to determine the probability that the schedule is selected to create the next generation. Schedules with good comprehensive performance have a far greater chance of being selected than poorer schedules. Crossover and mutation operations are performed to generate the next generation of the current schedule set, this continues until the variation in CP stabilises (diversity $\rightarrow$ 0) or a pre-set number of iterations have occurred.

The crossover operation selects two schedules from the current schedule set, cuts the schedules at a random location, merges the head portion of one schedule

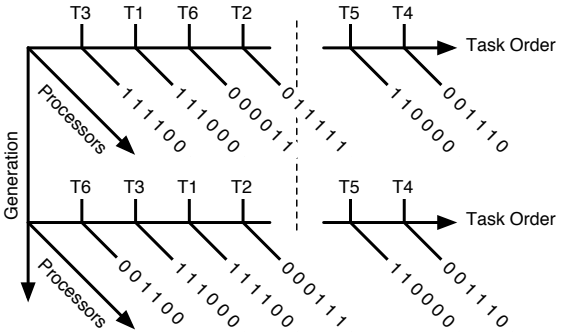


FIGURE 5. The crossover operator which merges the tail of one schedule with the head another. Where tasks are duplicated, the crossover process is reversed.

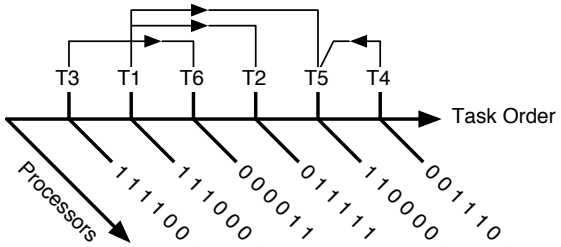


FIGURE 6. 3 sub-workflows coded in TITAN.

with the tail portion of the other and then reorders the schedules to produce two legitimate children; this process is illustrated in Figure 5. The mutation operation consists of two parts; one exchanges the execution order of two randomly selected jobs, while the other involves randomly adjusting jobs sizes as well as the allocation to host computers. The probability that these adjustments are inherited depends on whether they lead to performance improvements as well as the extent of the improvement. The probabilities of performing crossover and mutation are predetermined.

Using a GA with the concept of a fitness function to reward good quality schedules has been applied to a number of machine task processing problems [21]. However, its application in this case is useful because it has the ability to integrate well with the performance tool and be guided by multiple metrics. This has implications for both user class designation (how important a user is) and how the domain controller prioritises scheduled work (resource utilisation versus deadline).

3.2. Local Workflow

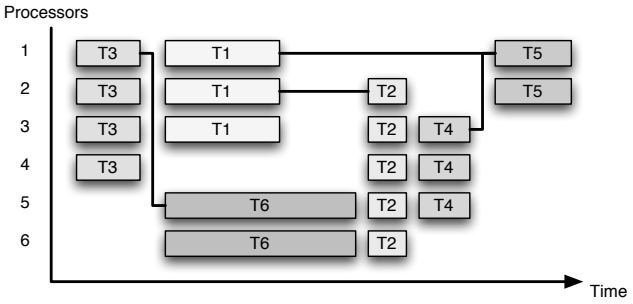
At the local level, it is necessary to resolve conflicts that occur when allocated sub-workflows compete for the same resource. Each scheduled task will have associated deadlines together with dependencies specified in the workflow description. In most cases, the domain administrator will want to minimise the degree of deadline failure while increasing throughput and utilisation. The GA is able to deal with this problem effectively as the multi-metric fitness function aims to identify schedule solutions that offer a reasonable compromise between each of the constituent factors.

In addition to the single task representation given in Figure 4, it is possible to specify task dependencies using the ordering property of the chromosome codings. Figure 6 illustrates the method of mapping sub-workflow descriptions onto the two dimensional coding scheme used by the GA. Tasks are moved into the TITAN system by the broker in temporal order and dependencies can only be specified on tasks that have already been accepted by the system. In this case, T_6 is dependent on T_3 (representing the simplest form of sub-workflow). The preferred task order representation places T_6 after T_3, although they would run simultaneously if the dependencies did not exist (the processor maps do not clash).

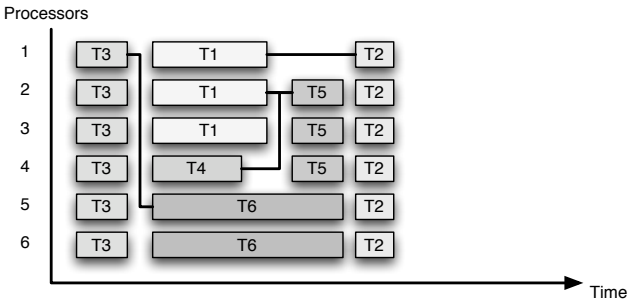
The composition algorithm, which queries the relevant performance models to build the schedule, transforms each dependency into a *task reservation* that behaves in the same manner as a blocked processor, except that this reservation is only visible to the task with the particular dependency. Creating these reservations will inevitably result in significant blocks of idle time in the schedule as tasks are kept back to wait for the completion of dependant tasks. However, assuming that the fitness function considers makespan and idle time (achieved by giving W^{i} and W^{m} non-zero weights), offspring schedules will invariably fill these gaps with other tasks as a consequence of the GA packing behaviour. This allows different sub-workflows to interleave according to their deadline, whilst respecting makespan and idle-time metrics.

Using the fitness function to improve the interleaving of the sub-workflow is preferable to other techniques that aim to constrain the search through the use of complex coding mechanisms. Such cases can limit the search properties of the scheduler by over-constraining the GA. By adopting a technique that uses similar coding, crossover and mutation processes to the single task case, allows exploration into diverse areas of the solution space without overburdening the search. As the scheduler is weighted towards reducing makespan and idle-time, in addition to resolving deadlines, it allows the system to perform its normal 'gap-filling' activities. This approach only effects the comprehensive penalty (CP) and so successful schedules that interleave the sub-workflow tend to enjoy a better overall performance and hence a lower CP.

Figure 7 illustrates the initial transform from coding to time. It is apparent in 7(a) that idle-time gaps are prevalent where dependent tasks cause blocking in the queue. However, after a 100 iterations of the scheduler GA the queue has been packed, removing most of the idle-time.



(a) Time composition of a schedule with 3 sub-workflows



(b) Improvement after 100 iterations of the GA

FIGURE 7. Schedule containing workflow: before and after GA packing.

3.3. Grid Workflow

At the grid level, a different algorithm is used that aims to allocate sub-workflows to a local scheduler that can run the tasks effectively. The algorithm aims to make effective use of the deadlines by allocating more time to sub-workflows where appropriate.

The approach at the grid level is to consider a workflow W as a set of sub-workflows S_i $(i = 1, \ldots, n)$. Let p_i be the number of pre- sub-workflows of S_i and q_i be the number of post- sub-workflows of S_i. Suppose that the group of potential local schedulers L_j $(j = 1, \ldots, m)$ constitutes an overall grid G. The scheduler aims to identify triples of start time (π_i^s), end time (π_i^e) and the allocated local grid (ζ_I). The scheduler processes each sub-workflow sequentially using the algorithm described in Algorithm 1.

The process is started with all the properties of each sub-workflow initialised to NULL. An additional parameter K is used to signify whether a sub-workflow has been scheduled. The scheduling process starts by looking for a schedulable sub-workflow, the pre-sub-workflows of which have all been scheduled. The start time of the chosen sub-workflow is configured with the latest end time of its pre- sub-workflows. The details of the sub-workflow as well as the start time are then submitted to a grid level broker. The brokers work together to discover an available local grid that can finish the sub-workflow at the earliest time. These are illustrated in Algorithm 1 as calls to local grid sub-workflow scheduling functions managed by the local TITAN system. Brokers filter the local grid resource information (from the information service) according to other properties (system architecture etc) and judge its applicability before a local grid is actually contacted. If there are a large number of local grids in the environment, a discovery scope can be defined to optimise the broker discovery performance. The scheduling ends when the end checkpoint is reached. In general, there is an additional adjustment or rescheduling procedure after scheduling. As shown in Algorithm 1, the adjustment is processed if the end time of a sub-workflow is earlier than the start times of its post- sub-workflows, so that the required deadlines of the sub-workflows are made less critical without increasing the scheduled execution time of the complete workflow. Another process can also be considered for rescheduling the less critical sub-workflows via the brokers. This is required when the cost and the execution time of the workflow are both considered. In this situation, less critical sub-workflows can be allocated to less powerful resources whose compute cost is less.

Algorithm 1 The grid-level scheduling algorithm

```
1: /** Initialisation **/
2: for i = 0 to i = n do
3:    π_i^s = NULL; π_i^e = NULL;
4:    ζ_i = NULL; K_i = FALSE;
5: end for
6: π_1^s = π_1^e = CurrentTime(); K_1 = TRUE;
7: /** Scheduling **/
8: for lp = 2 to lp = n do
9:    for i = 1 to i = n do
10:      if K_i = FALSE,K_ip = TRUE (p = 1,...,p_i)
         then
11:         BREAK;
12:      end if
13:   end for
14:   /** Scheduling via TITAN **/
15:   π_i^s = latest {π_ip^e | p = 1,...,p_i};
16:   if i ≠ n OR π_i^e ≠ π_i^s then
17:      (π_i^e, ζ_i) = ealiest {S_i, π_i^s};
18:   end if
19:   K_i = TRUE;
20: end for
21: /** Adjustment **/
22: for i = 2 TO i = n − 1 do
23:   n_i^e = earliest {n_iq^s | q = 1,...,q_i};
24: end for
```

This global grid workflow management algorithm relies heavily on information obtained from the local sub-workflow scheduler. The scheduler must be capable of returning the *latest* completion time of the incoming tasks as well as the complementary operator *earliest*

which returns the earliest enabling time. The response to these functions allows the broker to build a workflow schedule and compare them against solutions from other brokers. The local TITAN scheduler cannot give precise results to these functions (the GA search is random), but can produce approximate results by attaching the task to the tail of the current queue and using the performance service to obtain an estimate of these figures.

When tasks are received by a broker, they pass a message to the local sub-workflow scheduler to simulate an execution of the task. In practice, this involves running a pre-set number of iterations of the GA. The local scheduler then returns the earliest and latest time to the broker which then compares these with timings received from other brokers. The sub-workflow scheduler that meets the QoS requirements is subsequently allocated the tasks. If multiple solutions exist, the task is routed to the most local resource.

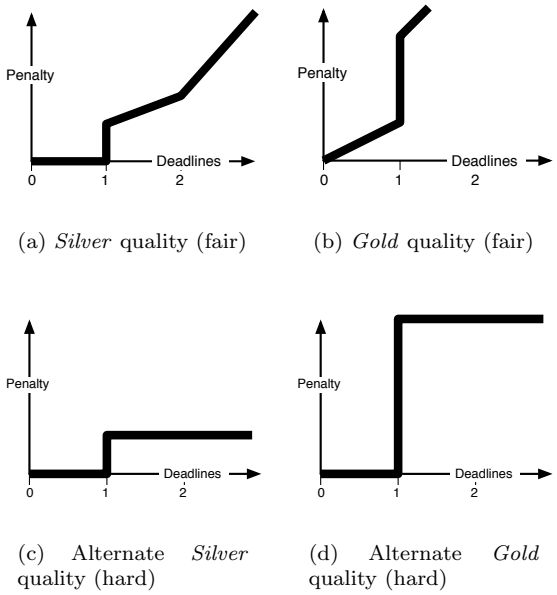
The difficulty with this approach is handling users with different priorities who will place more importance on task deadlines than on system metrics. The brokers and local schedulers are able to address this issue with user classes.

3.4. User Class Desgination

Grid computing grew out academic and research institutions where the user base was, historically, less concerned than industry with QoS issues. As grid services converge with commercially-orientated web services, it is inevitable that service quality will become increasingly important. Such issues have important implications for workflow systems that must consider multi-domain systems where each domain may have significantly different performance requirements.

The local schedulers offer some flexibility in how the fitness function is configured and it is possible to apply cost functions to the deadline weight in order to dynamically alter the behaviour of the local scheduler. Figures 8(a) and 8(b) illustrate two such functions for a standard domain that wishes to offer *gold* and *silver* services to applications. The functions are specified in terms as utility functions that modify the deadline on a per-application basis, returning a penalty measure that is a fraction of the domain's pre-set QoS weight for deadline (W^c). The curves are selected by the domain administrator to obtain a specified behaviour from the GA for the particular domain. In this case, the function *silver* is flat until the first deadline at which it takes a 'penalty hit' - this slopes until twice the deadline time at which it reaches a higher gradient. Where the GA builds schedules that have applications that fail their deadline, the CP is adversely affected and it is likely that the GA will select a schedule where the applications meet their deadlines.

The *gold* service function 8(b) has an increasing deadline penalty from the outset which encourages the GA to select schedules that run the application as soon as possible. Failure to meet the deadline results in a large penalty, worse than the equivalent *silver* failure at the same point in time.



(a) *Silver* quality (fair)

(b) *Gold* quality (fair)

(c) Alternate *Silver* quality (hard)

(d) Alternate *Gold* quality (hard)

FIGURE 8. Deadline utility functions

These functions depend on how QoS failure is measured (and paid for). Figures 8(c) and 8(d) offer alternative utility functions that are based on pass/fail contracts (as found in hard scheduling). In other words, once the deadline has failed the user will be compensated and so it does not matter whether the task is run now or at any time in the future, the penalty is fixed. While this may not feel intuitively correct from a user perspective, it may be reasonable from the broader view of system averages. In this case it may be better to let one deadline slip if it allows many other applications to achieve their deadline.

A related issue at the grid level is how one user designation maps onto another designation in a foreign domain. An application submitted to a grid as *silver* may run differently depending on whether the allocated domain uses cost curves that resemble Figure 8(a) or 8(c). One approach is to adopt an economic model similar to work by Buyya [18] where a credit fund is imposed that costs each deadline function against the incoming application and user profile.

A *gold* user on a local domain may only achieve *silver* status on another domain given the same level of capital. While this is a relatively crude method of mapping users across a domain, it provides some measure of what level of service the user can expect between different domains. Using a similar mechanism to the resource model sharing, brokers can advertise the cost functions which can be used by remote brokers to determine where to place the sub-workflow.

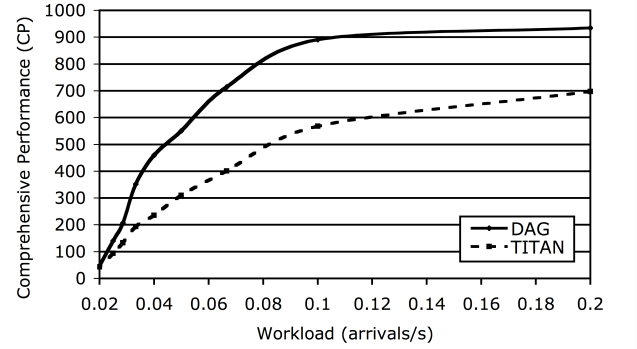
4. EXPERIMENTAL EVALUATION

To demonstrate the TITAN workflow architecture, a series of case studies are performed to evaluate the effectiveness of the system. In each instance a selection of heterogeneous applications are executed that have run-times between 10s and 100s, depending on the type of application and the architecture the task is allocated to. The applications are taken from a group of small scientific kernels that are used as PACE test cases and have sufficiently accurate performance models. The system is configured so that each local scheduler is responsible for 16 underlying grid resources: a cluster of 2.6Ghz Pentium 4 workstations in this case.

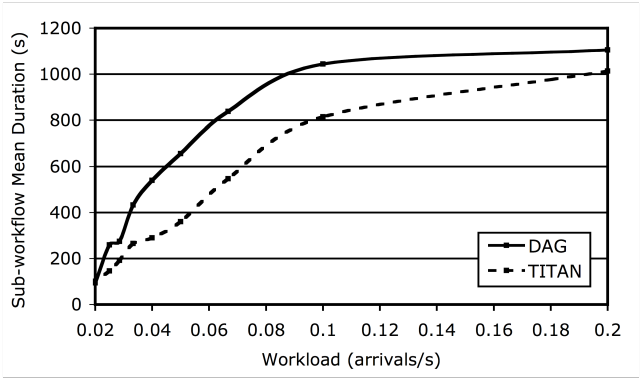
The first experiment is a straightforward demonstration of the packing characteristic exhibited by the GA and is used to increase throughput and reduce idle time. The scheduler is compared with a DAG-based scheduler (which approximates the local scheduler's behaviour), configured with the same parameters as TITAN. 100 sub-workflows, each containing 5 tasks, are are admitted to the DAG scheduler with a user-specified number of processors and allocated to the cluster. The scheduler is able to utilise free processors to minimise idle-time and improve throughput. While TITAN operates in a similar fashion, the GA is able to change the number of processors allocated by exploring the performance model data. It can identify, from the execution curves, appropriate limits (both upward and downward) on the available processors as well as compromising a task's execution time to improve another task's deadline. This gives TITAN an advantage in removing idle time from the schedule. Another advantage is the dynamic element: should a processor (a host) fail, TITAN is able to adapt quickly, removing the processor from the task-order coding and re-evaluating the schedule. The DAG scheduler may stall if there are not sufficient processors available for a particular task to complete.

Figure 9(a) illustrates the difference between the DAG and TITAN approaches when presented with sub-workflows at different arrival rates. The vertical axis is the CP measure and is a straightforward combination of make-span and idle time (W^c is set to 0 in this experiment). It is evident that at low workloads, the schedulers respond in a similar manner. At higher loads however, the GA-based scheduler is able to maintain a lower make-span (higher utilisation and lower idle-time) through task re-mapping, while using the DAG scheduler results in a poorer performance. The related results in Figure 9(b) illustrate the average duration of a scheduled sub-workflow. While TITAN is able to make a small reduction to the makespan of each sub-workflow, it tends to maintain a 'system view' when improving throughput and utilisation. In some cases, this can be to the detriment of an individual user.

A second experiment evaluates the QoS features of TITAN by comparing the execution of sub-workflows under low, medium and high workloads against user-specified deadlines. For each workload rate, sub-workflows arrive at the scheduler with a deadline specified in a given range. The range is uniformally selected and based on the application and architectures performance, representing a realistic restriction for the application. The results are given in Figures 10(a) and 10(b) which show the ratio of failed tasks and the average 'over-time' respectively. TITAN is typically able to meet more deadlines at a given workload.



(a) Comparison of comprehensive performance for DAG and TITAN under different workloads

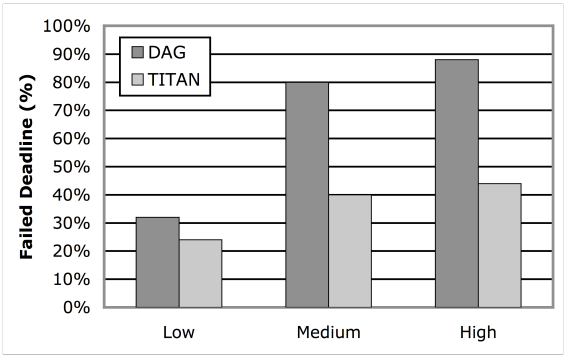


(b) Mean length scheduled sub-workflows

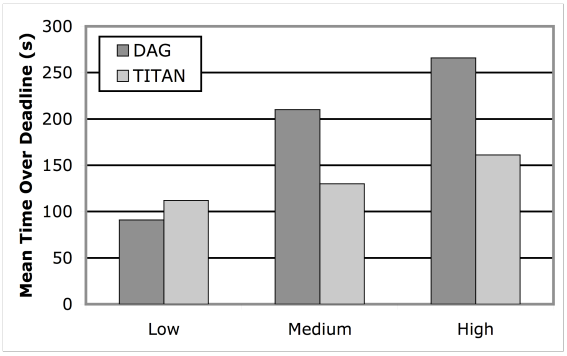
FIGURE 9. Comparison of the TITAN and DAG schedulers

Figures 11(a) and 11(b) illustrate the effect of the utility functions on the fitness and the associated CP. The workload in this experiment is the same as the previous experiment where a number of tasks failed to meet their deadlines. However, a proportion of the tasks have been given *gold* status whilst the remaining are given *silver*.

The 'fair' scheduler fails many deadlines at both low, medium and high load. However, the *degree of failure* (or the 'over-deadline', which contributes directly to the CP) is small. In contract, the 'hard' scheduler typically fails deadlines less often, but the degree of failure is more severe as their is no incentive to run the task after

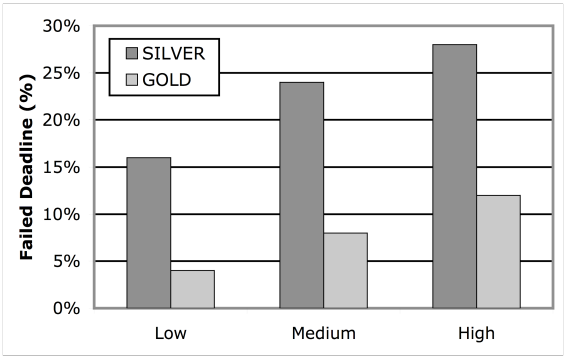


(a) Proportion of failed deadlines

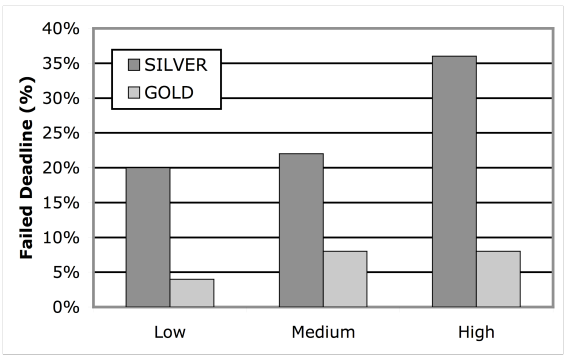


(b) Mean duration of each sub-workflow

FIGURE 10. Impact on QoS metrics with varying workload.



(a) Based on utility functions 8(a) and 8(b)



(b) Based on utility functions 8(c) and 8(d)

FIGURE 11. The effect of the utility functions on the deadlines for the same workload.

the deadline has passed.

5. RELATED WORK

Workflow management has become an essential service in grid environments in a relatively short time frame. Many issues still exist including: workflow specification and adjustment, on-the-fly workflow construction, enactment and orchestration, simulation and scheduling. There are a number of on-going projects that address one or more of these challenges. Among existing projects Pegasus [9] is most related to the work described in this paper. Pegasus aims at planning for workflow execution in grids. Artificial intelligence algorithms are used for workflow scheduling. In addition, it is currently integrated with Condor's [10] DAGman[11] which is similar to the scheduling approach used by TITAN [13]. Pegasus does not provide any QoS support at this time, while the algorithms in TITAN enable deadlines for job executions to be satisfied and provides the basis for QoS support.

There are a number of other research projects which are partly related to this work. GridAnt [22] reuses the Ant framework and provides a client-controllable workflow system for GT3 processes. While GridAnt provides a set of Grid tasks to be used within the Ant framework, workflow scheduling is currently not the key concern in the GridAnt project. IBM BPWS4J is a web services flow execution engine designed for BPEL4WS [23], which also has the potential to become a workflow standard in the grid community. Other current projects that have a workflow focus include USA ASCI grid [14] and GriPhyN [15], UK e-Science project MyGrid [24], EC/IST FP5 projects GEMSS [25], GridLab [26], and GRASP [27], the Swiss project BioOpera [28], Japan's NAREGI [29] and Business Grid projects [30]. Previous research on using workflow management in integrated metacomputing and problem solving environments include Webflow [31], Symphony [32], Triana [33], SPINEware [34], TENT [35] and UNICORE [36]. Most of these projects focus on workflow specification and enactment and target specific applications, they do not primarily address workflow scheduling.

6. CONCLUSIONS

In this paper a workflow management system has been presented that is able to enact workflow descriptions at both the local domain and the grid level. Performance models are used extensively to guide decision making at each level in the system. User class utility functions are applied to local and grid scheduling algorithms to improve workflow allocations across multi-domains. Different domains are able to specify different operating parameters to suit their particular environment, which makes the system attractive to diverse grid applications. The main contribution of this work is to add performance-awareness to workflow services. It does not define a new workflow language, as there are already a large number of contenders, but aims to offer an additional management tier that orchestrates the workflow so that deadlines are met, throughput is increased and resource utilisation is improved.

Workflow management is currently the subject of a number of research projects as it clearly facilitates distributed, collaborative working practices. Giving users the ability to connect large storage databases to computational and visualisation services co-ordinated by supporting middleware is a strong benefit of grid computing, and one that is being actively pursued by grid researchers. The approaches presented here allow users to submit workflows to grid resources with expected levels of service, performance and reliability. As important is the ability for grid resource providers to carefully tune their environments to meet potentially binding contractual service goals.

ACKNOWLEDGEMENTS

This work is sponsored in part by grants from the EPSRC (contract No. GR/R47424/01) and the EPSRC e-Science Core Programme (contract No. GR/S03058/01).

REFERENCES

[1] F. Berman, R.J. Hey, G. Fox Grid Computing: Making the Global Infrastructure a Reality. Wiley Series in Communications Networking and Distributed Systems, 2003.

[2] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[3] F. Curbera, W. Nagy, S. Weerawarana, Web Services: Why and How. Workshop on Object-Oriented Web Services, ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '01), 14-18 October 2001, Tampa Bay, Florida.

[4] D. Bacigalupo, S. Jarvis, L. He, G. Nudd. An investigation into the application of different performance techniques to e-Commerce applications. Workshop on Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems, 18th IEEE International Parallel and Distributed Processing Symposium 2004.

[5] V. Adve, R. Bagrodia, J. Browne, and E. Deelman et. al. Poems: end-to-end performance design of large parallel adaptive computional systems. *IEEE Transactions on Software Engineering*, 26(11):1027–1048, 2000.

[6] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. PACE : A toolset for the performance prediction of parallel and distributed systems. *Int. Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[7] N. Thomas. Challenges and Opportunities in Grid Performability. CS-TR: 842, School of Computer Science, University of Newcastle, May 2004

[8] P. Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3):225–236, 2002.

[9] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *J. Grid Computing*, 1(1):25-39, 2003.

[10] M. Litzkow, M. Livny, and M. Mutka. Condor – a hunter of idle workstations. *8th International Conference on Distributed Computing Systems (ICDCS'88)*, pp. 104–111, 1988.

[11] Condor Team, University of Wisconsin-Madison, Condor Version 6.4.7 Manual, 2003.

[12] N. Furmento, A. Meyer, S. McGough, S. Newhouse, T. Field, J. Darlington. ICENI: Optimisation of Component Applications within a Grid Environment. *Parallel Computing*, 28(12):1753-1772, 2002.

[13] D. Spooner, S. Jarvis, J. Cao, S. Saini, G. Nudd. Local grid scheduling techniques using performance prediction. *IEE Proc.-Comput. Digit. Tech.*, 150(2):87-96, 2003.

[14] H. P. Bivens. Grid Workflow, Grid Computing Environments Working Group, Global Grid Forum, 2001.

[15] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, S. Koranda, GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists. 11th IEEE Int. Symp. On High Performance Distributed Computing, Edinburgh, Scotland, pp. 225 -234, 2002.

[16] J. Cao, D. Kerbyson, G. Nudd. High performance service discovery in large-scale multi-agent and mobile-agent systems. *Int. Journal of Software Engineering and Knowledge Engineering*, 11(5):621–641, 2001.

[17] J. Cao, S. Jarvis, S. Saini, D. Kerbyson, G. Nudd. ARMS: an agent-based resource management system for grid computing. *Scientific Programming*, 10(2):135–148, 2002.

[18] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience*, 14:1507-1542, 2002.

[19] Ant 1.6.1 from `ant.apache.org`

[20] B. Sotomayor. The Globus Toolkit 3 Programmers Tutorial, Chapter 1, Section 3, May

11, 2004. www.casa-sotomayor.net/gt3-tutorial/multiplehtml/ch01s03.htm

[21] A.Y. Zomaya, Y.H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Trans. Parallel Distrib. Syst.* 12(9):899-911, 2001.

[22] K. Amin, G. von Laszewski, M. Hategan, N.J. Zaluzec, S. Hampton, A. Rossi. GridAnt: a Client-Controllable Grid Workflow System. 37th IEEE Annual Hawaii Int. Conf. on System Sciences, pp. 210-219, 2004.

[23] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0, July 2002. www.ibm.com/developerworks/library/ws-bpel/.

[24] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, A. Wipat, P. Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 2004.

[25] J. Cao, J. Fingberg, G. Berti, J.G. Schmidt. Implementation of Grid-enabled Medical Simulation Applications Using Workflow Techniques. 2nd Int. Workshop on Grid and Cooperative Computing, Shanghai, China, *Lecture Notes in Computer Science* 3032:34-41, 2003.

[26] E. Seidel, G. Allen, A. Merzky, J. Nabrzyski. GridLab - a Grid Application Toolkit and Testbed. *Future Generation Computer Systems* 18(8):1143-1153, 2002.

[27] T. Dimitrakos, D.M. Randal, F. Yuan, M. Gaeta, G. Laria, P. Ritrovato, B. Serhan, S. Wesner, K. Wulf. An Emerging Architecture Enabling Grid Based Application Service Provision. 7th IEEE Int. Enterprise Distributed Object Computing Conf., Brisbane, Australia, pp. 240-251, 2003.

[28] W. Bausch, C. Pautasso, G. Alonso. Programming for Dependability in a Service-based Grid. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, Tokyo, Japan, pp. 164-171, 2003.

[29] K. Miura. NaReGI - The Japanese National Research Grid Initiative. CCGrid 2003.

[30] H. Kishimoto, T. Kojo, F. Maciel. Business Grid Middleware Goals and Status. GlobusWorld 2004. www.globusworld.org/program/slides/3c_1.pdf

[31] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, G. Premchandran. WebFlow: a Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing. *Concurrency: Practice and Experience* 9(6):555-577, 1997.

[32] M. Lorch, D. Kafura. Symphony: A Java-based Composition and Manipulation Framework for Computational Grids. 2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Berlin, Germany, pp. 136-143, 2002.

[33] I. Taylor, M. Shields, I. Wang, O. Rana. Triana Applications within Grid Computing and Peer to Peer Environments. *J. Grid Computing* 1(2):199-217, 2003.

[34] E.H. Baalbergen, H. van der Ven. SPINEware - a Framework for User-oriented and Tailorable Metacomputers. *Future Generation Computer Systems* 15(5-6):549-558, 1999.

[35] A. Schreiber. The Integrated Simulation Environment TENT. *Concurrency and Computation: Practice and Experience* 14(13-15):1553-1568, 2002.

[36] M. Romberg. The UNICORE Grid Infrastructure. *Scientific Programming* 10(2):149-157, 2002.