In Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), Tokyo, Japan, May 2003, pp. 198-205.

GridFlow: Workflow Management for Grid Computing

Junwei Cao[*], Stephen A. Jarvis[†], Subhash Saini[‡] and Graham R. Nudd[†]
[*]*C&C Research Laboratories, NEC Europe Ltd., Sankt Augustin, Germany*
[†]*Department of Computer Science, University of Warwick, Coventry, UK*
[‡]*NASA Ames Research Centre, Moffett Field, California, USA*
Corresponding email: cao@ccrl-nece.de

Abstract

Grid computing is becoming a mainstream technology for large-scale distributed resource sharing and system integration. Workflow management is emerging as one of the most important grid services. In this work, a workflow management system for grid computing, called GridFlow, is presented, including a user portal and services of both global grid workflow management and local grid sub-workflow scheduling. Simulation, execution and monitoring functionalities are provided at the global grid level, which work on top of an existing agent-based grid resource management system. At each local grid, sub-workflow scheduling and conflict management are processed on top of an existing performance prediction based task scheduling system. A fuzzy timing technique is applied to address new challenges of workflow management in a cross-domain and highly dynamic grid environment. A case study is given and corresponding results indicate that local and global grid workflow management can coordinate with each other to optimise workflow execution time and solve conflicts of interest.

1. Introduction

Grid computing originated from a new computing infrastructure for scientific research and cooperation [11] and is becoming a mainstream technology for large-scale resource sharing and distributed system integration [12]. Essential grid services include information services, resource management, data transfer, security, and so on.

In this work, another important grid service - workflow management - is proposed; it includes an initial development of a framework and supporting algorithms, a so-called *GridFlow* system. An initial implementation of a GridFlow user portal is introduced. A two-tier service framework is presented with both global grid workflow management and local grid sub-workflow scheduling. The main functionalities of grid workflow management include workflow construction, simulation, scheduling, execution, monitoring, conflict solving, and so on.

This work is based on our previous work on grid resource management. An agent-based methodology is developed for global grid resource management using resource advertisement and discovery capabilities [5, 7]. A system implementation, ARMS [8], is also integrated with a local grid resource scheduling system, Titan [23]. Titan utilises an iterative heuristic algorithm to dynamically minimise the makespan and idle time of a particular grid resource without destroying user contracts. The functionalities of both ARMS and Titan are based on application performance prediction capabilities provided by the PACE system [20]. While our previous work assumes that users submit tasks individually to the grid, this work aims to enable grid users to construct, simulate, execute and monitor new grid applications that consist of flows of multiple tasks.

Workflow techniques have been developed for over ten years. A great deal of work has been carried out with regard to defining and implementing standards for workflow management systems [10]. While these are established research areas in other contexts, a grid environment presents a number of new challenges:

- *Cross-domain*: The process of a grid workflow encompasses multiple administrative domains (organisations). The lack of central ownership and control results in incomplete information and many other uncertain factors.
- *Dynamism*: Since grid resources are not entirely dedicated to the environment, computational and networking capabilities can vary significantly over time. Application performance prediction becomes difficult and real-time resource information update within a large-scale global grid becomes impossible.

In this work, a fuzzy timing technique [19] is applied to address the challenges when workflow scheduling and conflict management is processed. Workflow or task execution times are represented using fuzzy timestamps

and calculated via fuzzy enabling times and combined possibility distributions when conflicts occur. This method is illustrated using a case study and the results indicate that the use of fuzzy concepts is feasible especially when multi-site scheduling is involved [22].

There is a limited amount of work related to grid workflow issues in the grid computing community. Pioneering work includes WebFlow [2], a visual programming paradigm for the development of high performance distributed computing applications; this is however no longer an active project. A complementary concept to *workflow* is a *component*. The CCA (Common Component Architecture) and its XML implementation [14] have been developed for grid programming. Symphony [18] is a framework for combining existing codes to meta-programs without changes to the code, which is simpler and focuses more on security issues. In the work described in [13], CXML (Component XML) is used for component specification and further issues such as performance optimisation and implementation selection are addressed for component-based grid applications. Another XML based grid workflow specification is documented in [3] and used in the ASCI (Accelerated Strategic Computing Initiative) grid infrastructure. As mentioned in this work, the WfMC (Workflow Management Coalition) standard, WPDL (Workflow Process Definition Language) [10], is sophisticated and perhaps too generalised for grid computing. With the integration of grid technologies with Web Services protocols, WSFL (Web Services Flow Language) [17] alternately has potential as a grid workflow language. Other grid projects such Condor [1] and UNICORE [21] provide similar functionalities but require specific infrastructures.

A good summary of the above work can be found in [16], which refers to grid programming environments and models. The key issue that differentiates our work from these is that we focus more on service-level support, workflow management and scheduling, as opposed to workflow and component specifications, standards, or communication protocols at the programming level. The fuzzy timing method introduced here is suitable and straightforward when applied to the scheduling scenarios described in this work. The goal is not to necessarily provide the best scheduling solution. Another advantage of this approach is that the fuzzy time functions can be computed very fast and are suitable for scheduling of time-critical grid applications.

The rest of the paper is organised as follows: Section 2 provides an overview of our previous work on grid resource management; a workflow management framework and the supporting scheduling algorithms are described in detail in Section 3; a simple case study is included in Section 4 to illustrate the fuzzy timing method; and the paper concludes in Section 5 with proposed future work.

2. Grid Resource Management

Our previous work on grid resource management is based on two grid services: information and performance services. The Globus MDS [9] is adopted to provide grid resource information and indexing services and the PACE toolkit [20] is utilised to provide performance modelling and prediction capabilities for parallel programs. The implementation of grid resource management is carried out at multiple layers:

- *Grid Resource*: A particular grid resource is a high-end computing or storage resource that can be accessed remotely. These can be multiprocessors, or clusters of workstations or PCs with large disk storage space. Titan [23] is designed as a grid resource manager that schedules the execution of multiple parallel tasks with maximum resource utilisation.
- *Local Grid*: A local grid consists of multiple grid resources that belong to one organisation. These resources are usually connected with high speed networks. In our previous work, each local grid is managed using an agent [8].
- *Global Grid*: The global grid includes all grid resources that belong to different organisations within a virtual organisation. ARMS is developed as an agent-based resource management system for grid computing, in which multiple agents are organised in a hierarchical way [8].

2.1. PACE

Prediction-based grid resource management is provided using a system of application performance modelling and evaluation. The PACE toolkit [20] is used to provide this capability for both the local schedulers [23] and the grid agents [6]. Figure 1 illustrates the main components of the PACE toolkit.

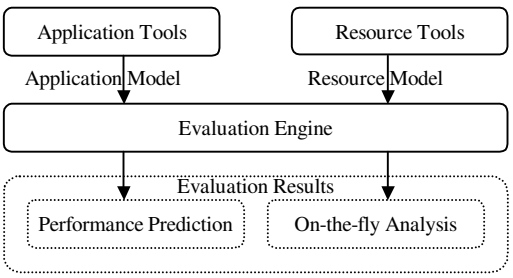

Figure 1. The PACE Toolkit

The PACE evaluation engine is able to combine application and resource models at run time to produce performance data (such as total execution time). PACE has been validated using ASCI high performance computing applications [4, 15]. The validation results show that a high level of accuracy can be obtained, cross-platform comparisons can be easily undertaken, and the process benefits from a rapid evaluation time. These features allow PACE predictive data to be used *on the fly* for grid resource management and scheduling.

The prediction capabilities of PACE have been developed for scientific computing tasks (e.g. parallel programs in MPI or PVM) that are computationally intensive (rather than data intensive); this work is therefore based in this domain. It is also the case that grid resources are only considered to be providers of high performance computing power as opposed to large-scale data storage facilities.

2.2. Titan

The Titan system [23] has been developed as a grid resource manager. By coupling application performance data with iterative heuristic algorithms, the system is able to dynamically balance the processes of minimising makespan of multiple tasks and idle time of multiprocessors, whilst adhering to deadlines requirements. Figure 2 illustrates the main components of the Titan system.

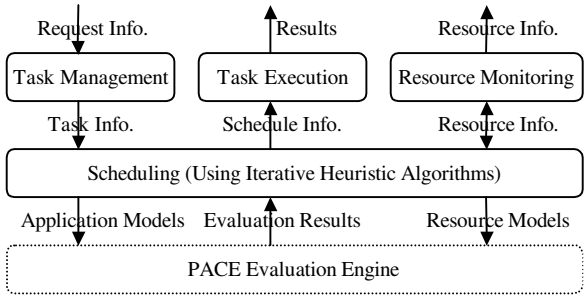

Figure 2. The Titan System

Requests are passed to the task management module where they queue for scheduling and execution. Resource monitoring is responsible for gathering statistics concerning the processors of a grid resource on which tasks may execute. The scheduling process uses heuristic algorithms to search for near-optimal schedules for the current task queue. This allows makespan and processor idle time to be minimised. When there are free resources available, tasks are submitted for execution. This is supported by the PACE performance predictive data. A Titan system also acts as a grid resource information provider in the Globus MDS implementation.

2.3. ARMS

Agents comprise the main components of ARMS [8]. Each agent is viewed as a representative of a local grid at a global level of grid resource management. Agents are organised into a hierarchy, which provides a high level view of grid resources. An illustration of ARMS, integrated with a number of Titan resource managers, is given in Figure 3.

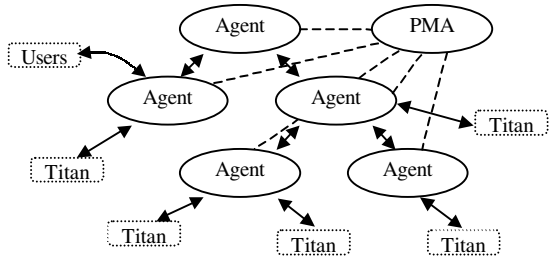

Figure 3. The ARMS Architecture

An agent utilises the Globus MDS for storing local grid resource information and those advertised from other agents. Agents also cooperate with each other to discover available resources for task execution requests submitted by grid users. The discovery processes utilise the Globus MDS protocols to lookup available grid resources. The PACE performance service is also accessed to provide an estimation of the task execution time so that appropriate resources can be allocated. Different strategies are used to optimise agent performance, which is controlled using a simulation-based performance monitor and advisor (PMA).

3. Grid Workflow Management

While our previous work assumes that grid users submit tasks individually to the ARMS agents, this work aims to provide additional services to enable management of flows of tasks submitted by grid users. The context of grid workflow management is illustrated in Figure 4.

While this work focuses more on service-level support such as grid workflow management and scheduling, a GridFlow user portal is also developed that provides a graphical user interface (GUI) to facilitate the composition of grid workflow elements and the access to additional grid services. The system is designed so that workflow management follows the same layered framework as that of resource management, including global grid workflow management and local grid sub-workflow scheduling. The implementation of grid workflow management is carried out at multiple layers:

- *Task*: Tasks are the smallest elements in a grid workflow. In general, grid workflow tasks are MPI & PVM jobs running on multiple processors, data transfers to visualisation servers, or archiving of large data sets to mass storage. In this work, only MPI & PVM jobs are considered. Task scheduling is implemented using Titan, and as stated, this work focuses more on the sub-workflow and workflow levels of management and scheduling.
- *Sub-workflow*: A sub-workflow is a flow of closely related tasks that is to be executed in a predefined sequence on grid resources of a local grid (within one organisation). Conflicts occur when tasks from different sub-workflows require the same resource simultaneously.
- *Workflow*: A grid application can be represented as a flow of several different activities, each activity represented by a sub-workflow. These activities are loosely coupled and may require multi-sited grid resources. Simulation, execution and monitoring services can be provided.

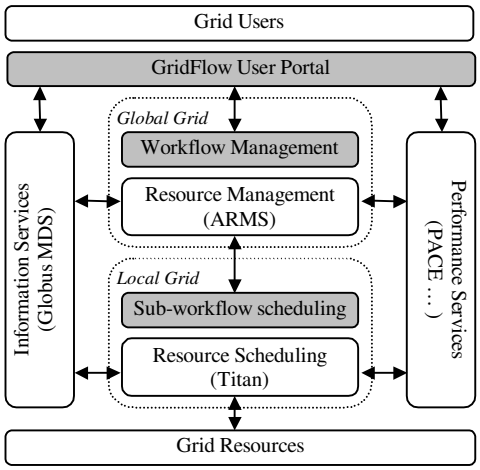

Figure 4. GridFlow in Context

The grey parts of Figure 4 are introduced in detail in the following sections. Corresponding scheduling algorithms are included and a supporting case study is provided in Section 4.

3.1. GridFlow User Portal

The GridFlow portal is an integrated environment that enables users to construct a grid workflow and access grid services. An initial Java implementation is illustrated in Figure 5.

To construct a grid workflow, a user needs to define properties of each sub-workflow and task and their execution sequences. In general, a sub-workflow or a task can have several pre- and post- activities. These are represented using an XML specification. If the user knows where a sub-workflow or a task will be executed, he can define this within the portal, which will contact the local grid agent or Titan directly. The portal also provides direct user interfaces to the information and performance services. However, if the user does not know anything about the available grid services and resources, he can submit the workflow to the global workflow management system, which will provide the services automatically. This work focuses on this situation.

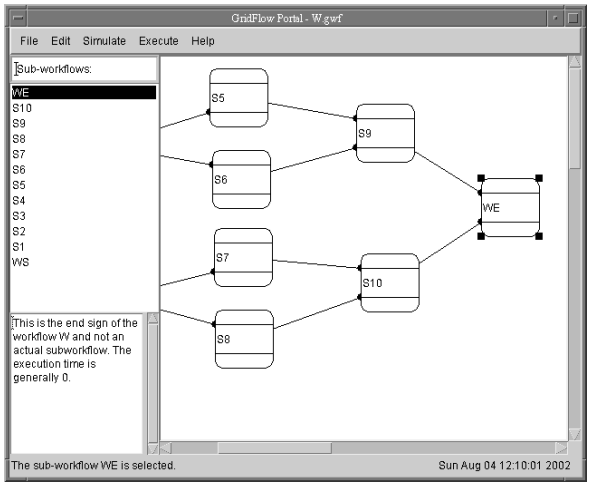

Figure 5. The GridFlow Portal

3.2. Global Grid Workflow Management

The global grid workflow management system receives requests from the GridFlow portal with XML specifications of grid workflows, and provides three main functionalities:

- *Simulation*: Simulation takes place before a grid workflow is actually executed, during which time a workflow schedule is achieved. The simulation results can be returned to the GridFlow portal for user agreement or passed directly to the execution engine.
- *Execution*: A grid workflow is executed according to the simulated schedule. Due to the dynamic nature of the grid environment, the schedule may not be executed accordingly. When large delays of some sub-workflows occur, the rest or whole of the workflow may be sent back to the simulation engine and rescheduled.
- *Monitoring*: Global grid workflow management also provides interfaces that provide access to real-time status reports of task or sub-workflow execution.

A workflow W can be defined as a set of sub-workflows S_i (i=1,……,n), including two checkpoints, S_1

and S_n, that indicate the starting and ending points respectively. Let p_i be the number of the pre- sub-workflows of S_i, and q_i be the number of the post- sub-workflows of S_i. Suppose that the global grid G is a set of local grids L_j (j=1,……,m). The main purpose of workflow management is to find a near optimal (in terms of the execution time) schedule Λ, which is a set of tri-tuples, $(\pi^s, \pi^e, \zeta)_i$, where $\pi^s{}_i$, $\pi^e{}_i$, and ζ_i are defined as the start time, the end time, and the allocated local grid of the sub-workflow S_i, respectively. The simulation is processed one sub-workflow at a time according to the algorithm GGWM described in Figure 6.

```
GGWM:
  // Initialisation
  FOR i=1 TO i=n DO
    π^s_i=NULL; π^e_i=NULL; ζ_i=NULL; κ_i=FALSE;
  ENDDO
  π^s_1=π^e_1=CurrentTime(); κ_1=TRUE;
  // Scheduling
  FOR lp=2 TO lp=n DO
    // Searching an schedulable S_i
    FOR i=1 TO i=n DO
      IF κ_i=FALSE AND ALL κ_ip=TRUE (p=1,……,p_i)
        BREAK;
      ENDIF
    ENDDO
    // Scheduling via ARMS
    π^s_i=latest{π^e_ip | p=1,……,p_i};
    IF i=n π^e_i=π^s_i;
    ELSE (π^e_i,ζ_i)=earliest{LGSS_j(S_i,π^s_i) | j=1,……,m};
    ENDIF
    κ_i=TRUE;
  ENDDO
  // Adjustment
  FOR i=2 TO i=n-1 DO
    π^e_i=earliest{π^s_iq | q=1,……,q_i};
  ENDDO
END
```

Figure 6. The GGWM Algorithm

The process is started with all the properties of each sub-workflow initialised (as null). An additional parameter κ 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 an ARMS agent. ARMS agents work together to discover an available local grid that can finish the sub-workflow execution at the earliest time. These are illustrated in Figure 6 as calls to local grid sub-workflow scheduling functions $LGSS_j$, which are introduced in the next section. In an actual situation, not all of local grids have to be tried. Firstly, agents can filter the local grid resource information (from the information service) according to other properties and judge its applicability before a local grid is actually contacted; Secondly, if there are a large number of local grids in the environment, a discovery scope can be defined to optimise the agent 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 Figure 6, 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 whole workflow. Another process can also be considered for rescheduling the less critical sub-workflows via ARMS. This is required when the cost and the execution time of the workflow have both to be considered. In this situation, less critical sub-workflows can be allocated to less powerful resources whose compute cost is less. This is not documented in Figure 6, as in this work we focus on the single metrics of workflow execution time.

The global grid workflow management introduced in this section relies heavily on the simulation results of local grid sub-workflow scheduling.

3.3. Local Grid Sub-workflow Scheduling

Scheduling a flow of tasks onto grid resources within a local grid is very similar to the process that schedules a workflow onto different local grids introduced above. One important difference is that the local grid sub-workflow scheduling has to deal with multiple tasks that may belong to different sub-workflows. The execution time has to be estimated with the extra consideration of conflicts, which may occur when multiple tasks require the same grid resource at the same time.

A sub-workflow can be defined as a set of tasks T_k (k=1,……,l). Each task requires a specific grid resource R_k. Again, let $\pi^s{}_k$ and $\pi^e{}_k$ be the start time and end time of task T_k. When there are resource conflicts, a task enabling time $\pi^a{}_k$ is also defined that is different from the actual start time $\pi^s{}_k$ of the task T_k. Two possible end times $\pi^{e1}{}_k$ and $\pi^{e2}{}_k$ are also defined that can be used to calculate the final end time of the task T_k. In the case where no conflicts exist, a task enabling time is equivalent to the start time, and the two possible end times are not used. The Titan system located on each grid resource is responsible for allocating processors to the task, and providing predictive task execution time, $\pi^d{}_k$, using the PACE functions. Suppose that T_c is a task from a different sub-workflow that has resource conflict with the task T_k. Assuming that T_0 is the start point of the task flow, the *LGSS* algorithm aims to provide an estimation of the end time of the last task, $\pi^e{}_l$, to the GGWM function, given a sub-workflow S and the start time π^s. This is described in Figure 7.

```
LGSS:
  // Initialisation
  FOR k=1 TO k=l DO
    π^s_k=NULL; π^e_k=NULL; κ_k=FALSE;
  ENDDO
  π^s_0=π^e_0=π^s; κ_0=TRUE;
  // Scheduling
  FOR lp=1 TO lp=l DO
    // Searching an schedulable T_k
    FOR k=1 TO k=l DO
      IF κ_k=FALSE AND ALL κ_kp=TRUE (p=1,……,p_k)
        BREAK;
      ENDIF
    ENDDO
    // Scheduling via Titan
    IF R_k=R_c
      // Conflict occurs
      // Calculating enabling times
      π^a_k=latest{π^e_kp | p=1,……,p_k};
      π^a_c=latest{π^e_cp | p=1,……,p_c};
      // Calculating start times
      π^s_k=min{π^a_k,earliest{π^a_k,π^a_c}};
      π^s_c=min{π^a_c,earliest{π^a_k,π^a_c}};
      // Calculating end times
      // If T_k occurs first
      π^e1_k=sum{π^s_k,π^d_k};
      π^e1_c=sum{latest{π^s_c,π^e1_k},π^d_c};
      // If T_c occurs first
      π^e2_c=sum{π^s_c,π^d_c};
      π^e2_k=sum{latest{π^s_k,π^e2_c},π^d_k};
      // Combining two possibilities
      π^e_c=max{π^e1_c,π^e2_c};
      π^e_k=max{π^e1_k,π^e2_k};
    ELSE
      // No conflict
      π^s_k=latest{π^e_kp | p=1,……,p_k};
      π^e_k=sum{π^s_k,π^d_k};
    ENDIF
    κ_k=TRUE;
  ENDDO
  // Adjustment
  FOR k=1 TO k=l-1 DO
    π^e_k=earliest{π^s_kq | q=1,……,q_k};
  ENDDO
END
```

Figure 7. The LGSS Algorithm

Local grid sub-workflow scheduling is composed of both forward and backward processes. The difference from the GGWM algorithm is that resource conflicts exist. In this case, the start time of the chosen task cannot be configured with the latest end time of its pre-tasks directly, since another task exists that may use the same resource at the same time. A first-come *possibly*-first-serve policy is adopted in the algorithm described in Figure 7 that gives a higher priority to a possibly earlier enabled task. This does not order the conflictive tasks explicitly, but adds some information on degrees of possibilities of task start times. There may be other policies that are justifiable for particular application domains. For example, a sub-workflow can be predefined with a priority value according to its importance levels among those sub-workflows in its workflow, and also within the local grid. When two tasks conflict on resource allocation, the task with the higher priority can be executed first. In Figure 7, two possible start sequences are considered and are combined to provide an estimation of the end time. There may be more than two tasks that are enabled simultaneously, which is not included in the algorithm but can be solved using a similar method. A more detailed introduction to the method can be found in [19].

It is a difficult task to provide an accurate prediction on the workflow start, execution and end times. The time parameters π used in Figures 6 and 7 are actually fuzzy time functions and corresponding operations, *latest*, *earliest*, *min* and *max* and *sum*, are also defined in detail in the following case study.

4. A Case Study

The algorithms introduced in the above section are implemented using fuzzy timing techniques. In this section, the detailed definitions of fuzzy time functions are included and illustrated using an example grid workflow management scenario.

4.1. Fuzzy Time Operations

A fuzzy time function $\pi(\tau)$ gives the numerical estimate of the possibility that an event arrives at time τ, which is often described in the trapezoidal or triangular possibility distribution specified by the 4-tuple (π_1, π_2, π_3, π_4). Two fuzzy time functions, $\pi_1(\tau)$=0.5(0,2,6,7) and $\pi_2(\tau)$=(2,4,4,6), and corresponding operation results are illustrated in Figure 8.

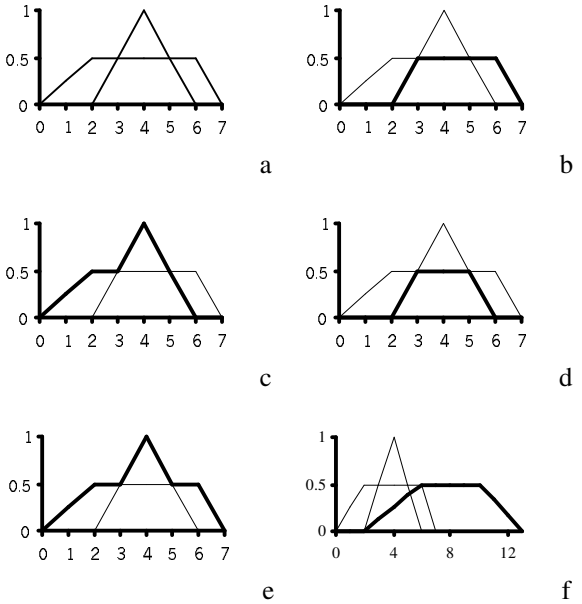

Figure 8. Fuzzy Time Functions and Operations

In Figure 8a, the trapezoidal and triangular possibility distributions of $\pi_1(\tau)$ and $\pi_2(\tau)$ are described. Figure 8b illustrates the operator *latest* that picks the latest arrival distribution of $\pi_1(\tau)$ and $\pi_2(\tau)$. A complementary operator *earliest* is also introduced in Figure 8c that picks up the earliest enabling time. The operator *min* performs the intersection of the two fuzzy time functions (see Figure 8d) and the operator max is opposite (see Figure 8e). The sum of the two fuzzy time functions is processed as follows: min{0.5,1}(0+2,2+4,6+4,7+6)=0.5(2,6,10,13), which is also illustrated in Figure 8f. The applications of these operations are given below.

4.2. An Example Scenario

As an example scenario, we consider a case where two workflows are involved, W_1 and W_2, as shown in Figure 9. In the local grid L_1, the task A_2 of sub-workflow S_3 from W_1 is being executed (grey in Figure 9) and S_3 from W_2 is to be scheduled (shadowed in Figure 9). Suppose that a resource conflict exists between A_3 and A_4. The schedule aims to find the $\pi^e{}_5(\tau)$.

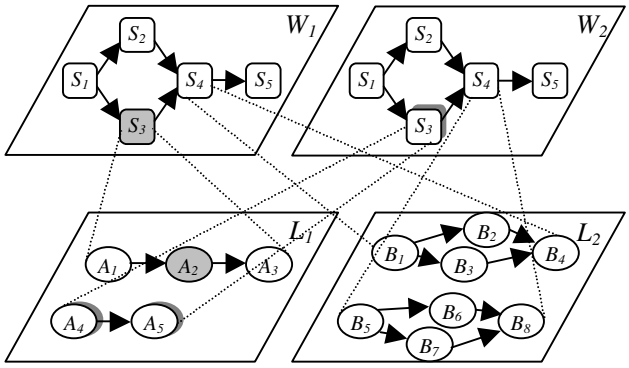

Figure 9. An Example Scenario

The task enabling times can be concluded from pre-task end times and task execution times can be obtained from the Titan system supported by the PACE functions. Suppose that these are all pre-defined as:

$\pi^d{}_3(\tau)$=(3,5,5,7); $\pi^d{}_3(\tau)$=(5,6,7,8);
$\pi^d{}_4(\tau)$=(0,3,3,5); $\pi^d{}_4(\tau)$=(10,12,14,16);
$\pi^d{}_5(\tau)$=(2,5,6,9).

According to the algorithm described in Figure 7, the sub-workflow S_3 from W_2 can be scheduled at the local grid L_1 as follows:

$\pi^s{}_3(\tau)$ = min{(3,5,5,7), earliest{(3,5,5,7), (0,3,3,5)}}
= min{(3,5,5,7), (0,3,3,5)}
= 0.5(3,4,4,5)
$\pi^s{}_4(\tau)$ = min{(0,3,3,5), earliest{(3,5,5,7), (0,3,3,5)}}
= min{(0,3,3,5), (0,3,3,5)}
= (0,3,3,5)
$\pi^{e1}{}_3(\tau)$ = sum{0.5(3,4,4,5), (5,6,7,8)}
= 0.5(8,10,11,13)
$\pi^{e1}{}_4(\tau)$ = sum{latest{0.5(8,10,11,13), (0,3,3,5)}, (10,12,14,16)}
= sum{0.5(8,10,11,13), (10,12,14,16)}
= 0.5(18,22,25,29)
$\pi^{e2}{}_4(\tau)$ = sum{0,3,3,5}, (10,12,14,16)}
= (10,15,17,21)
$\pi^{e2}{}_3(\tau)$ = sum{latest{(10,15,17,21), 0.5(3,4,4,5)}, (5,6,7,8)}
= sum{0.5(10,12.5,19,21), (5,6,7,8)}
= 0.5(15,18.5,26,29)
$\pi^e{}_4(\tau)$ = max{ 0.5(18,22,25,29), (10,15,17,21)}
≈ (10,15,17,29) // See Figure 10
$\pi^e{}_5(\tau)$ = sum{(10,15,17,29), (2,5,6,9)}
= (12,20,23,38)

The calculation concludes that S_3 from W_2 will complete on the local grid L_1 most likely between time 20 and 23. This data can be submitted so that the global grid workflow management system is able to decide whether the local grid L_1 should be allocated the sub-workflow S_3 from W_2. Note that in order to simplify the calculation, the combined possibility distribution of the end time of the task A_4 is represented approximately using a trapezoidal instead of the complicated original result provided by the *max* operation. This is also illustrated in Figure 10.

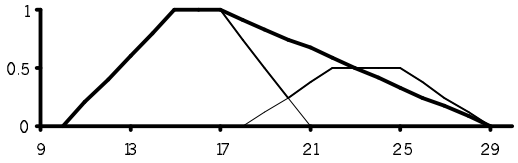

Figure 10. Combined Possibility Distribution and its Approximation

This fuzzy timing technique provides a good solution to the conflict solving problem arising from grid workflow management issues. This method is especially useful in highly dynamic grid environments, where large network latencies exist and application performance is difficult to predict accurately.

5. Conclusions

A grid workflow management system, GridFlow, is introduced in this work. It has been developed at Warwick based on previous work on performance prediction and grid resource management. The GridFlow user portal is described together with the service support of both global grid workflow management and local grid sub-workflow scheduling. Corresponding algorithms are included and a fuzzy timing method is applied and illustrated using a case study.

A grid performance service is under development that comprises the PACE performance prediction capability with a new application response measurement technique [24], which can be used to enable prediction-based

scheduling as well as response-based scheduling. New OGSA [12] standards and protocols are to be applied to the whole system implementation. Grid workflow management also brings new challenges on issues like security, as it requires more flexible cooperation among different grid services and components. These will be addressed when the GridFlow system become mature.

Acknowledgements

This work is sponsored by grants from the NASA AMES Research Centre (administered by USARDSG, contract No. N68171-01-C-9012), the EPSRC (contract No. GR/R47424/01), the EPSRC e-Science Core Programme (contract No. GR/S03058/01), and the NEC.

References

[1] J. Basney, and M. Livny, "Deploying a High Throughput Computing Cluster", High Performance Cluster Computing, Vol. 1, Chapter 5, Prentice Hall, 1999.

[2] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran, "WebFlow – a Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing", Concurrency: Practice and Experience, Vol. 9, No. 6, pp. 555-577, 1997.

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

[4] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Performance Modelling of Parallel and Distributed Computing Using PACE", in Proc. of 19th IEEE Int. Performance, Computing and Communication Conf., Phoenix, AZ, USA, pp. 485-492, 2000.

[5] J. Cao, D. J. Kerbyson, and G. R. Nudd, "Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing", in Proc. of 1st IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia, pp. 311-318, 2001.

[6] J. Cao, S. A. Jarvis, D. P. Spooner, J. D. Turner, D. J. Kerbyson, and G. R. Nudd, "Performance Prediction Technology for Agent-based Resource Management in Grid Environments", in Proc. of 11th IEEE Heterogeneous Computing Workshop, Fort Lauderdale, FL, USA, 2002.

[7] J. Cao, D. P. Spooner, J. D. Turner, S. A. Jarvis, D. J. Kerbyson, S. Saini, and G. R. Nudd, "Agent-based Resource Management for Grid Computing", in Proc. of 2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Berlin, Germany, pp. 350-351, 2002. (short paper)

[8] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd, "ARMS: an Agent-based Resource Management System for Grid Computing", Scientific Programming, Special Issue on Grid Computing, Vol. 10, No. 2, pp. 135-148, 2002.

[9] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", in Proc. of 10th IEEE Int. Symp. on High Performance Distributed Computing, San Francisco, CA, USA, 2001.

[10] L. Fisher (eds.), Workflow Handbook, Workflow Management Coalition, 2002.

[11] I. Foster, and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.

[12] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid Services for Distributed System Integration", IEEE Computer, Vol. 35, No. 6, pp. 37-46, 2002.

[13] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington, "Optimisation of Component-based Applications within a Grid Environment", in Proc. of Supercomputing 2001.

[14] D. Gannon, R. Bramley, G. Fox, et. al., "Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications", Cluster Computing, Vol. 5, pp. 325-336, 2002.

[15] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive Performance and Scalability Modelling of a Large-Scale Application", in Proc. of Supercomputing 2001.

[16] C. Lee, S. Matsuoka, D. Talia, A. Sussman, M. Mueller, G. Allen, and J. Saltz, "A Grid Programming Primer", Advanced Programming Models Research Group, Global Grid Forum, 2002.

[17] F. Leymann, "Web Services Flow Language (WSFL 1.0)", IBM Software Group, 2001.

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

[19] T. Murata, "Temporal Uncertainty and Fuzzy-Timing High-Level Petri Nets", invited paper in Proc. of Application and Theory of Petri Nets, LNCS 1091, pp. 11-28, 1996.

[20] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, "PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems", Int. J. High Performance Computing Applications, Special Issues on Performance Modelling – Part I, Vol. 14, No. 3, pp. 228-251, 2000.

[21] M. Romberg, "The UNICORE Grid Infrastructure", Scientific Programming, Special Issue on Grid Computing, Vol. 10, No. 2, pp. 149-157, 2002.

[22] J. Sauer, G. Suelmann, and H. Apelrath, "Multi-site Scheduling with Fuzzy Concepts", Int. J. Approximate Reasoning, Vol. 19, pp. 145-160, 1998.

[23] D. P. Spooner, J. Cao, J. D. Turner, H. N. Lin Choi Keung, S. A. Jarvis, and G. R. Nudd, "Localised Workload Management Using Performance Prediction and QoS Contracts", in Proc. of 18th Annual UK Performance Engineering Workshop, Glasgow, UK, pp. 69-80, 2002.

[24] J. D. Turner, D. P. Spooner, J. Cao, S. A. Jarvis, D. N. Dillenberger, and G. R. Nudd, "A Transaction Definition Language for Java Application Response Measurement", J. Computer Resource Management, Vol. 105, pp. 55-65, 2002.