Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G

Yun Yang[1], Ke Liu[2, 1], Jinjun Chen[1], Joël Lignier[1], Hai Jin[2]

1- Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Melbourne, Australia 3122
{yyang, kliu, jchen, jlignier}@ict.swin.edu.au

2- School of Computer Science & Technology
Huazhong University of Science and Technology
Wuhan, Hubei, China 430022
hjin@hust.edu.cn

Abstract

Nowadays, grid and peer-to-peer (p2p) technologies have become popular solutions for large-scale resource sharing and system integration. For e-science workflow systems, grid is a convenient way of constructing new services by composing existing services, while p2p is an effective approach to eliminate the performance bottlenecks and enhance the scalability of the systems. However, existing workflow systems focus either on p2p or grid environments and therefore cannot take advantage of both technologies. It is desirable to incorporate the two technologies in workflow systems. SwinDeW-G (Swinburne Decentralised Workflow for Grid) is a novel hybrid decentralised workflow management system facilitating both grid and p2p technologies. It is derived from the former p2p based SwinDeW system but redeveloped as grid services with communications between peers conducted in a p2p fashion. This paper describes the system design and functions of the runtime environment of SwinDeW-G.

Index Terms: Grid Workflows, Peer-to-Peer Workflows, E-Science, Coordination, Decentralisation

1. Introduction

Recently, grid computing [11] and peer-to-peer (p2p) technology [1] are two popular solutions for resource sharing and system integration which are desirable for sophisticated e-science workflows. Accordingly, there is a demand to investigate grid and/or p2p workflow systems. A grid workflow can be defined as the composition of grid application services which execute on heterogeneous and distributed resources in a well-defined order to accomplish a specific goal [21]. A p2p workflow, on the other hand, facilitates p2p technologies to workflow for direct communication and cooperation among relevant peers [19]. Both systems have their respective advantages.

Compared with traditional workflow systems, the grid workflow systems have some advantages [15]: (1) ability to build dynamic applications which orchestrate distributed resources; (2) utilisation of resources that are located in a particular domain to increase throughput or reduce execution costs; (3) execution spanning multiple administrative domains to obtain specific processing capabilities; and (4) integration of multiple teams involved in managing of different parts of the experiment workflow thus promoting inter-organisational collaborations.

The p2p workflow systems also have some merits [19]. They abandon the centralised data repository and control engine and fulfil the whole workflow functions by distributing both data and control. Thus the performance bottlenecks are likely eliminated and the system scalability can be greatly enhanced.

Inherited from our exiting work on the p2p based workflow system SwinDeW (Swinburne Decentralised Workflow) [19], SwinDeW-G (SwinDeW for Grid) also uses XPDL (XML Process Definition Language[1]) for workflow definition. However, this paper focuses on the SwinDeW-G runtime environment. The specific requirements of the SwinDeW-G runtime environment are as follows. First, in order to take advantages of both grid computing and p2p technology, it is desirable that SwinDeW-G is developed as a p2p based grid workflow. Second, given the existence of SwinDeW, it is desirable that SwinDeW-G is realised in a cost effective manner, i.e., not developed from scratch, rather, it should be based on p2p based SwinDeW but ported to the grid environment.

The rest of the paper is organised as follows. In the next section, some typical grid workflow systems are

[1] http://www.wfmc.org/standards/xpdl.htm

discussed. Section 3 illustrates the system design which combines grid and p2p technologies. Section 4 then demonstrates the system functions of SwinDeW-G. After that, a case study is used to illuminate how the system works in Section 5. Finally, Section 6 concludes the paper and outlines future work.

2. Related Work

Our former SwinDeW described in [19] is a typical p2p based workflow system where the detailed related work of p2p workflow systems can be found there. Therefore, in this section, we concentrate on related work for grid workflow systems as it is the primary focus of this paper.

With the increasing interest in grid workflow, many grid workflow systems emerge in recent years [20]. Here we choose some grid workflow systems, namely, Gridbus [4], Pegasus [9], Taverna [13], GrADS [3], ASKALON [10], GridAnt [17], Karajan [18], Triana [6], GridFlow [7] and Kepler [2], to demonstrate the main characteristics of current research outcomes and compare them with SwinDeW-G [16].

As for system installation, Gridbus and ASKALON only need Globus Toolkit[2], while Taverna, Karajan and Kepler only need Java. Pegasus, GrADS and GridAnt need other toolkits such as Condor's DAGman [12], autopilot [14] and Apache Ant[3], as well as Globus Toolkit. Tirana and GridFlow need their own platforms to run. Using popular toolkit brings better adaptability while using own toolkits brings more flexibility. Considering its complex runtime environment, SwinDeW-G chooses the more general Globus Toolkit and Java to develop on.

As far as QoS (quality of service) constraints are concerned, most grid workflow systems mentioned above do not support this feature. Gridbus supports QoS constraints including task deadline and cost minimisation, GrADS and GridFlow mainly use estimated application execution time, and ASKALON supports constrains and properties specified by users or predefined. Right now, SwinDeW-G supports QoS constraints based on task deadline.

When it comes to fault tolerance, at the task level, Gridbus, Taverna, ASKALON, Karajan, GridFlow and Kepler use alternate resource; Taverna, ASKALON and Karajan use retry; GrADS uses rescheduling. At the workflow level, rescue workflow is used by ASKALON and Kepler; user-defined exception handling is used by Karajan and Kepler. Pegasus, GridAnt and Triana use their particular strategies respectively. As a comparison, SwinDeW-G uses effective task-level temporal constraint verification for fault tolerance.

As for the architecture of the workflow scheduling infrastructure, Pegasus, Taverna, GrADS, GridAnt, Karajan and Kepler use a centralised architecture; Gridbus and GridFlow use a hierarchical architecture; ASKALON and Triana use a decentralised architecture. It is believed that centralised schemes produce more efficient schedules and decentralised schemes have better scalabilities, while hierarchical schemes are their compromises. Derived from former SwinDeW, SwinDeW-G uses a decentralised scheme for workflow scheduling.

As for scheduling strategies, Pegasus, Taverna, GrADS, Triana and GridFlow support performance-driven strategies; Gridbus supports market-driven strategy; only ASKALON supports both performance-driven and market-driven strategies. A performance-driven strategy can achieve optimal execution performance by mapping workflow tasks onto resources according to specific strategies and the market-driven strategy tries to allocate resources for workflow tasks according to market models. SwinDeW-G aims at using a performance-driven strategy to achieve an overall load balance of the whole system via distributing tasks to least loaded neighbours.

For intermediate data movement, Gridbus, Taverna and ASKALON use a centralised approach; Pegasus uses mediated approach; GridAnt and Karajan use user-directed approach; GrADS, Triana and GridFlow use p2p approach. Kepler supports all approaches mentioned above. The centralised approaches are easier to implement and mediated approaches are more scalable and suitable for applications which need to keep intermediate data for later use, while p2p approaches are more suitable for those applications which involve with large-scale data flow. Designed to support large-scale workflows, SwinDeW-G chooses the p2p approaches not only at the data level for such as intermediate data movement but also at the control level for such as workflow execution.

In overall terms, although the most existing grid workflow systems mentioned above can support the execution of grid workflows and have their respective characteristics, they do not fully facilitate the p2p technology to their runtime tools. While we port SwinDeW to grid environment as SwinDeW-G in a cost effective fashion, we retain its p2p feature to increase the system efficiency and enhance the system scalability and at the same time inherit the advantages of the grid technology.

[2] http://www.globus.org/toolkit/

[3] http://ant.apatche.org

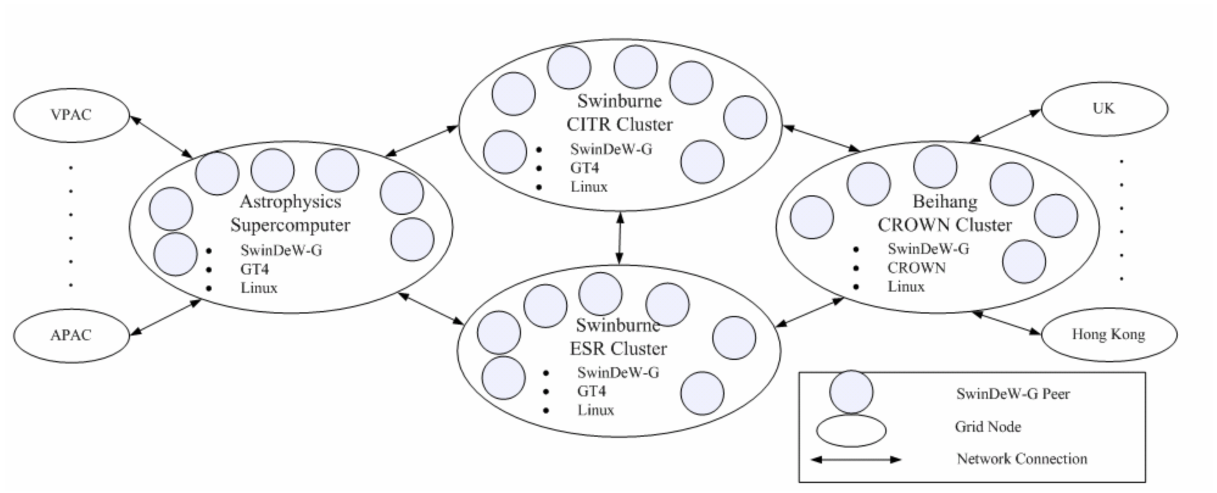

Figure 1: Physical Network Layout of SwinGrid Environment

3. System Design

SwinDeW-G is running on a grid environment called SwinGrid. An overall picture of SwinGrid is depicted in Figure 1 which contains many grid nodes distributed in different places. Each grid node contains many computers including high performance PCs and/or supercomputers composed of significant number of computing units. The primary hosting nodes include the Swinburne CITR (Centre for Information Technology Research) Node, Swinburne ESR (Enterprise Systems Research laboratory) Node, Swinburne Astrophysics Supercomputer Node, and Beihang CROWN (China R&D environment Over Wide-area Network) [8] Node in China. They are running Linux, GT (Globus Toolkit) 4.04 or CROWN grid toolkit 2.5 where CROWN is an extension of GT 4.04 with more middleware, hence compatible with GT 4.04. Besides, the CROWN Node is also connected to some other nodes such as those in Hong Kong University of Science and Technology and University of Leeds in UK. The Swinburne Astrophysics Supercomputer Node is cooperating with such as APAC (Australian Partnership for Advanced Computing), VPAC (Victorian Partnership for Advanced Computing) and so on. Currently, SwinDeW-G is deployed at all primary hosting nodes. In SwinDeW-G, a grid workflow is executed by different peers that may be distributed at different grid nodes. As shown in Figure 1, each grid node can have a number of peers, and each peer can be simply viewed as a grid service.

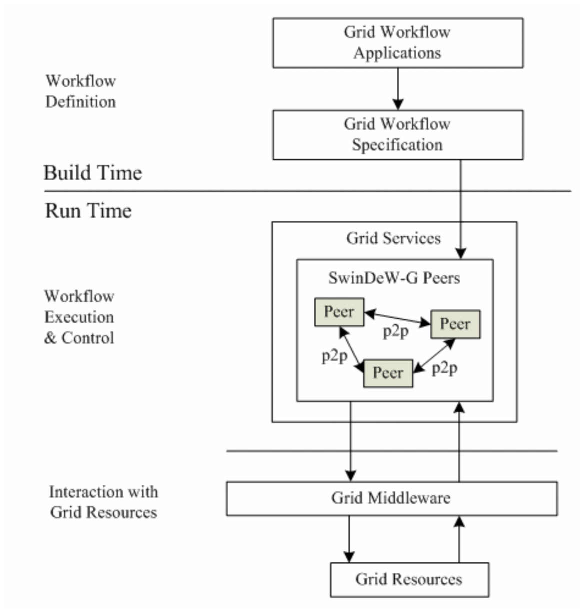

Figure 2: Architecture of SwinDeW-G

As we mentioned before, SwinDeW-G is a p2p based grid workflow system that enables workflows to be executed over a grid environment using direct p2p communications among peers. This is achieved by wrapping SwinDeW-G peers inside grid services and deploying them as grid middleware applications. This relationship can be seen in Figure 2. Once deployed, SwinDeW-G peers will search for and connect with other SwinDeW-G peers. After that, the peers use p2p to exchange various information required to execute a workflow.

Unlike SwinDeW, a SwinDeW-G peer runs as a grid service along with other grid services. However, it communicates with other peers via JXTA[4], a platform for p2p communication. As Figure 3 shows, a SwinDeW-G peer consists of the following components:

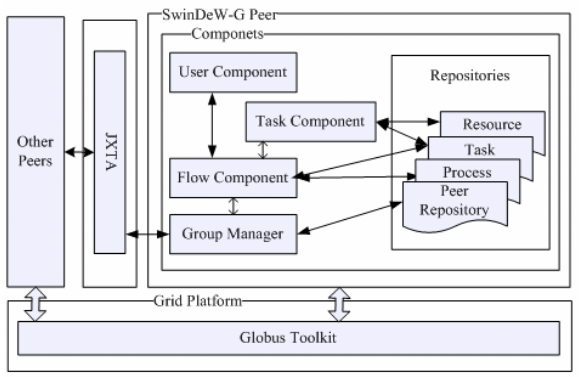

Figure 3: Architecture of SwinDeW-G Peer

- The Task Component manages the workflow tasks. It has three main functions. First, it provides necessary information to the Flow Component for scheduling and stores received tasks to Task Repository. Second, it determines the appropriate time to start, execute and terminate a particular task according to the capability. A capability in SwinDeW-G is an object encapsulating rules with a role in workflow processes, which include the responsibility of this role, usage scenarios of this role, application-related constraints of each scenario (input, allowable operations, output, etc.), and so on. The resources that a workflow task instance may require are stored in the Resource Repository.
- The Flow Component interacts with all other modules. First, it receives the workflows definition and then creates the instance definition. Second, it receives tasks from other peers or redistributes them. Third, it decides whether to pass a task to the Task Component to execute locally or distribute it to other peers. The decision is made according to the capabilities and load of itself and other neighbours. And finally, it makes sure that all executions conform to the data dependency and control dependency of the process definitions which are stored in the Process Repository and the Task Repository.
- The Group Manager is the interface between the peer and JXTA. In JXTA, all communications are conducted in terms of peer group, and the Group Manager maintains the peer groups the peer has joined. The information of the peer groups and the peers in them is stored in the Peer Repository. While a SwinDeW-G peer is implemented as a grid service, all direct communications between peers are conducted via p2p. Peers communicate to distribute information of their current state and messages for process control such as heartbeat, process distribution, process enactment etc.
- The User component is the interface between the corresponding workflow users and the workflow environment. In SwinDeW-G, its primary function is to allow users to interfere with the workflow instances when exceptions occur.

Globus Toolkit serves as the grid service container of SwinDeW-G. Not only a SwinDeW-G peer itself is a grid service located inside Globus Toolkit, the capabilities which are needed to execute certain tasks are also in forms of grid services that the system can access. That means when a task is assigned to a peer, Globus Toolkit will be used to provide the required capability as grid service for that task.

4. System Functions

This section describes the system functions of workflow execution in SwinDeW-G. First, we illustrate how a workflow process is defined; second, we demonstrate how the peers are managed; third, we address how tasks of a workflow instance are organised; and finally, we describe how the instance is executed.

4.1. Process Definition

In SwinDeW-G, the process definition is specified in the XPDL workflow language. In general, a SwinDeW-G process can be represented by a two-tuple process notation P (Process-ID; Task-Set). Further, a task can be described as a four-tuple task notation T (Process-ID; Task-ID; Transition-Restriction-Set; Extended-Attribute-Set).

For processes, Process-ID is the unique identifier of the process in the workflow system, and Task-Set is the set of tasks which constitutes the process. For tasks, Process-ID is the identifier of the process in the workflow system to which the task belongs, and Task-ID is the unique identifier of the task in the context of the process.

Transition-Restriction-Set is a set of workflow constraints. Each constraint represents an edge of the directed graph of the workflow process. Each edge can be described as a three-tuple notation R (Mode, Condition, Other-Task-ID). When Mode is 'join', it represents that this task is the end point on the flow

[4] http://www.sun.com/software/jxta/

edge. In this circumstance, Condition can be such as 'and' or 'or'. If it is 'and', the task will not be initialised until all 'join' conditions become true, and if it is 'or', the task will be instanced as long as any 'join' condition becomes true. When Mode is 'split', it represents that this task is the beginning point on the flow edge. Again, Condition can be such as 'and' or 'or'. If it is 'and', the subsequent tasks can be executed in parallel, and if it is 'or', a follow up task will be selected from the subsequent task list according to the condition in order. Finally, Other-Task-ID is the identifier of the task on the other end of the flow edge.

Extended-Attribute-Set is the collection of optional attributes depending on the application. Each attribute can be described as (Name, Value). The most important attribute in a task is the attribute named capability, the value of which indicates one of the required capabilities needed to execute the task.

4.2. Peer Management

Unlike normal grid services, SwinDeW-G is always considered dynamic due to the joining and leaving of peers. Peer management derives mostly from former SwinDeW. Its main function is maintaining a list of current neighbours which is essential for distribution, scheduling and execution of workflows. In detail, peer management has to handle the following:

- Peer join

In SwinDeW-G, Peer Groups are organised by capabilities defined in a workflow process. When a new peer joins SwinDeW-G, it joins a base group which contains all the peers in the network, regardless of their capabilities. In this base group, each peer will advertise in the group, so when a peer is searching for another peer, it will search through the advertisements to find the peer it wants. Then the new peer will try to join some groups according to each of its capabilities. If the group already exists, the peer simply joins it; otherwise, it creates a new group and joins it as creator.

When a peer joins an existing group successfully, it will send an advertisement message to the group. Other peers which are already in this group will respond the advertisement and pass the related process definition data to the new peer. Also, the newly joined peer is added to these peers' list of neighbours automatically. Thus the new peer can merge into the workflow system and be able to execute certain tasks immediately.

- Peer search

In SwinDeW-G, each peer has respective capabilities. When a task is distributed to a peer, it checks if it has the capability to execute it first. If the peer can execute the task, then the task will be passed to the Task Component for execution; else it will check if one of its neighbours has the requested capability. If there is one, it will redirect this task to that neighbour; otherwise it will invoke a global search to find if any peer has the required capability, which is rare in general.

The process of global search is described as follows. First, in every peer group it joined, including the base group of SwinDeW-G which contains all SwinDeW-G peers, the peer sends a search message to all other peers. For every peer who received the message, if it knows that some peers have the required capability, it will return the information of those peers to the sender. The process will stop when either the peers with required capability are found or no responses are received for a certain period of time, which usually means exception that no peer in the system has the required capability.

- Peer leave

A SwinDeW-G peer may leave the system at any time either explicitly or implicitly. The system has to respond to these events and keep the system running properly. In the former situation, the peer who is going to leave will inform the neighbours in its neighbour list about its leave. When its neighbours receive the message, they will remove the peer from their neighbour list accordingly. In the later circumstance, the discovery mechanism depends on the heartbeat messages which are gossiped periodically in the system to indicate that the peer who sent them is still alive. So, if a peer left the system unexpectedly, its heartbeat messages will not be heard by its neighbours anymore. When a peer has not heard one of its neighbours for a period of time, it will consider this neighbour to be inactive and will remove it from its neighbour list. In both circumstances, the tasks which are relevant to this peer have to be rescheduled accordingly by finding a replacement peer with the same procedure of task instantiation described in Section 4.3 next.

4.3. Task Instantiation

In workflow systems, a process can be started by a request or a coming event. The process flowchart of SwinDeW-G can be described as follows.

The peer will start a process instance when it receives an instantiation message. In this occasion, the peer will get the process ID from the message and searches it in its Process Repository. If the process can be found, the peer will check if there is an instance of the process already running. If there is not, it will create one.

In the second step, the peer will get the ID of the start task and find if there is an instance of the task already running. If there is not, it will create one.

Once the new task instance is instantiated, the peer will check the execution condition. The task can be executed if one of the following conditions is met: (1) the join condition is “and” and all the task’s predecessors have been done where sequential execution is a special case; (2) the join condition is “or” and one of the task’s predecessors has been done; (3) The task is the start task.

If a task can be executed, the peer will try to instantiate its subsequent tasks. In fact, for each subsequent task, it will send a message to one of its neighbours who has the required capability. It should be addressed that this peer may or may not be the peer on which the task will be executed. When all subsequent tasks are distributed, it will notify all predecessors that the task has been instantiated via sending them specific messages.

When a peer receives the initiating message, it will search the least loaded peer among itself and its neighbours who have the necessary capability that needs to execute the task. To find which peer has the least load, Globus Toolkit can be facilitated to obtain the needed information which includes current CPU load, service availabilities etc. Once the least loaded peer has been found, it will then send a message to the peer to start the task.

If a task has no successors, it would be the last task. When such a task has been done, the peer will send a message to the enacting peer to start execution of the process.

4.4. Instance Execution

In SwinDeW-G, whether a task can be executed on a peer depends on two conditions: the data condition and the control condition. Most workflow tasks need some input data to start. The data are normally the output of its predecessor(s). Similarly, the task also generates some results as the input of its successor(s). Only all necessary data are collected can a task be started. This is called data condition.

As described earlier, a task can be executed only after some relevant tasks have been finished. This is called control condition. Unlike traditional centralised workflow system, this control consistency is achieved by collaborations among SwinDeW-G peers. Several control messages which are transferred between these peers are as follows:

- Predecessor message

This message is used by a predecessor task to notify its successor task(s) whether a task is completed or still being executing. When a successor task receives this message, it will modify the status of the corresponding predecessor task and check if it itself can be enacted.

- Successor message

This message is used by a successor task to notify its predecessor task(s) whether the task has been enacted or not. When a predecessor task receives this message, it will update the status of the corresponding successor task and check if it itself can be enacted.

- Successor instance message

This message is used to tell its predecessor(s) that an instance of this task has been created. When a peer receives this message, it sets the sender as the successor neighbour of the task instance.

5. Case Study

In this section, we facilitate a case study to illustrate how SwinDeW-G supports the execution of p2p based grid workflows.

At first, we introduce some background of this case study. In reality, complex scientific processes are normally time constrained, hence temporal verification is needed. The tasks at which we conduct the verification are called checkpoints. In grid workflow systems, a checkpoint selection strategy (CSS) is responsible for selecting checkpoints for conducting temporal verification.

This section discusses a case which is realised on SwinDeW-G. The simulation is the comparison of several checkpoint selection strategies where the details can be found in [5]. In this paper, we only focus on how SwinDeW-G supports these grid workflows.

The SwinGrid grid environment has already been described earlier in Section 3. Figure 4 shows a partial workflow process that was used in the simulations and how it is distributed to the grid environment. The complete process for simulation consists of over 1000 activities but for the sake of simplicity only 7 of them are shown. The workflow process executes tasks in a partial order. There is a branch at activity an1 where some tasks are executed in parallel.

When this workflow process is executed each task is assigned to a peer. This assignment is based on which peer is suitable to execute the task. To be suitable the peer must first be capable of executing that task and not be busy with other tasks. Once all the activities have been assigned to a peer the workflow process is then executed from start to end. Each peer that has a task assigned to it will communicate with the other peers so that the workflow process executes in the expected order.

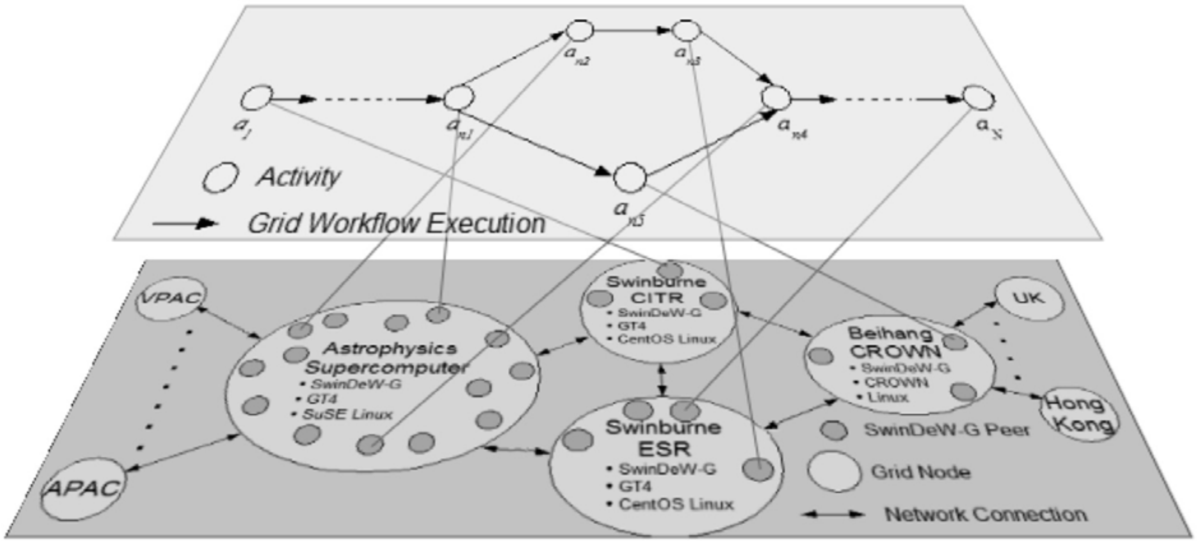

Figure 4: Workflow Process Simulation Environment

The result shows that SwinDeW-G can support complicated workflows which need intensive computation because it uses grid to execute workflow tasks, and its p2p based communication can reduce the overall traffic for increasing the efficiency and improving the scalability.

It has been demonstrated that the test workflows can run on SwinDeW-G properly and many useful outcomes can be drawn. It can be concluded that SwinDeW-G is a suitable p2p based grid workflow environment that can support sophisticated e-science applications effectively. In summary, our primary requirements of SwinDeW-G described in Section 1 are successfully achieved.

6. Conclusions and Future Work

In this paper, we have presented the runtime environment of SwinDeW-G (Swinburne Decentralised Workflow for Grid) which is a novel peer-to-peer (p2p) based grid workflow system incorporating p2p and grid technologies for taking advantages of both. The SwinDeW-G runtime environment is realised based on former SwinDeW p2p based workflow system as grid services to reduce the development cost. The utilisation of the grid technology provides more power to handle sophisticated e-science workflow applications while the facilitation of the p2p technology improves the performance and increases the scalability of the system.

In the future, SwinDeW-G still needs further improvement. On one hand, load balancing would occur when multiple peers have the same capability. However, the task scheduling is now primarily performed at the task instantiation stage and static in the current system which is insufficient. New scheduling algorithms will be developed to balance the load which is of course in a dynamic and distributed manner. On the other hand, monitoring is not implemented in the current version. However, it would be desirable to be able to monitor the status of the workflows. In addition, SwinDeW-G will also be compared with other grid workflow systems.

Acknowledgement

The research work reported in this paper is partly supported by Australian Research Council under Discovery Grant DP0663841, Linkage Grant LP0669660 and by Swinburne Dean's Collaborative Grants Scheme 2007-2008.

References

[1] K. Aberer and M. Hauswirth, "Peer-to-peer information systems: Concepts and models, state-of-the-art, and future systems", *Proc. of 8th European Software Engineering Conf. (ESEC) and 9th ACM SIGSOFT Symp. Foundations Software Engineering (FSE-9)*, Vienna, Austria, Sep. 2001, pp. 326–327.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher and S. Mock, "Kepler: An Extensible System for Design and Execution of Scientific Workflows", *Proc. of 16th International Conference on Scientific and Statistical*

Database Management (SSDBM 2004), Santorini Island, Greece, June 2004, pp. 423-424.

[3] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development", *International Journal of High Performance Computing Applications (JHPCA)*, 15(4):327-344, Winter 2001.

[4] R. Buyya and S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report", *Proc. of 1st IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, Seoul, Korea, Apr. 2004, pp. 19-36.

[5] J. Chen and Y. Yang, "Adaptive Selection of Necessary and Sufficient Checkpoints for Dynamic Verification of Temporal Constraints in Grid Workflow Systems", *ACM Transactions on Autonomous and Adaptive Systems*, 2(2): Article 6, June 2007 (http://www.acm.org/pubs/taas/)

[6] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor and I. Wang, "Programming Scientific and Distributed Workflow with Triana Services", *Concurrency and Computation: Practice and Experience*, 18(10):1021–1037, Dec. 2005.

[7] J. Coa, S. Jarvis, S. Saini, and G. Nudd, "GridFlow: Workflow Managament for Grod Computing", *Proc. of 3rd International Symposium on Cluster Computing and the Grid, 2003, (CCGrid 2003)*, Tokyo, Japan, May 2003, pp. 198-205.

[8] CROWN Team 2006, CROWN portal, http://www.crown.org.cn/en/, accessed on July 1, 2007.

[9] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta and K. Vahi, "Mapping Abstract Complex Workflows onto Grid Environments", *Journal of Grid Computing*, 1:25-39, 2003.

[10] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr. and H. L. Truong, "ASKALON: A Tool Set for Cluster and Grid Computing", *Concurrency and Computation: Practice and Experience*, 17:143-169, 2005.

[11] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1998.

[12] J. Frey, T. Tannenbaum, I. Foster, M. Livny and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", *Cluster Computing*, 5: 237-246, 2002.

[13] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M.R. Pocock, A. Wipat, and P. Li, "Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows", *Bioinformatics*, 20(17):3045-3054, 2004.

[14] D. Reed and R. L. Ribler, "Performance Analysis and Visualization", In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pp. 367–394, Morgan Kaufmann Publishers, USA, 1998.

[15] D.P. Spooner, J. Cao, S. A. Jarvis, L. He and G. R. Nudd, "Performance-aware Workflow Management for Grid Computing", *The Computer Journal*, 48(3): 347-357, 2004.

[16] SwinDeW-G Team 2006, System Architecture of SwinDeW-G, http://www.ict.swin.edu.au/personal/jchen/SwinDeW-G/System_Architecture.pdf, accessed on Sept. 17, 2007.

[17] G. von Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton and A. Rossi, "GridAnt: A Client-Controllable Grid Workflow System", *Proc. of 37th Hawaii International Conference on System Sciences (HICSS-37)*, Hawaii, USA, Jan. 2004, pp. 210-219.

[18] G. von Laszewski and M. Hategan, "Java CoG Kit Karajan/GridAnt Workflow Guide", *Technical Report*, Argonne National Laboratory, Argonne, IL, USA, 2005.

[19] J. Yan, Y. Yang and G. K. Raikundalia, "SwinDeW - A Peer-to-peer based Decentralized Workflow Management System", *IEEE Transactions on Systems, Man and Cybernetics*, Part A, 36(5):922-935, 2006.

[20] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing", *Journal of Grid Computing*, 3:171-200, Sept. 2005.

[21] J. Yu and R. Buyya, "A Novel Architecture for Realizing Grid Workflow using Tuple Spaces", *Proc. of 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, Pittsburgh, USA, Nov. 2004, pp. 119-128.