



# A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on a Cloud Computing Platform

The International Journal of High Performance Computing Applications  
24(4) 445–456

© The Author(s) 2010

Reprints and permission:

sagepub.co.uk/journalsPermissions.nav

DOI: 10.1177/1094342010369114

hpc.sagepub.com



Ke Liu<sup>1,2</sup>, Hai Jin<sup>1</sup>, Jinjun Chen<sup>2</sup>, Xiao Liu<sup>2</sup>, Dong Yuan<sup>2</sup> and Yun Yang<sup>2</sup>

## Abstract

The concept of cloud computing continues to spread widely, as it has been accepted recently. Cloud computing has many unique advantages which can be utilized to facilitate workflow execution. Instance-intensive cost-constrained cloud workflows are workflows with a large number of workflow instances (i.e. instance intensive) bounded by a certain budget for execution (i.e. cost constrained) on a cloud computing platform (i.e. cloud workflows). However, there are, so far, no dedicated scheduling algorithms for instance-intensive cost-constrained cloud workflows. This paper presents a novel compromised-time-cost scheduling algorithm which considers the characteristics of cloud computing to accommodate instance-intensive cost-constrained workflows by compromising execution time and cost with user input enabled on the fly. The simulation performed demonstrates that the algorithm can cut down the mean execution cost by over 15% whilst meeting the user-designated deadline or shorten the mean execution time by over 20% within the user-designated execution cost.

## Keywords

Cloud Workflows, Workflow Scheduling Algorithms, Instance-Intensive Workflows, Cost-Constrained Workflows, Cloud Computing

## 1. Introduction

On investigation of certain e-business and e-government applications, there emerges a new type of workflow which we call instance-intensive workflows. Unlike computation-intensive workflows in e-science, instance-intensive workflows are workflows characterized by a huge number of concurrent, often relatively simple, workflow instances (hence instance-intensive). Considering instance-intensive workflows, the mean execution time, which indicates how many workflow instances can be completed in a certain time, becomes a more important criterion of scheduling instance-intensive workflows than execution time of individual instances. Typical examples of instance-intensive workflows include bank check processing, insurance claim processing and many other e-business and e-government scenarios. Here we take the bank check processing as an example to illustrate the characteristics of instance-intensive workflows. Typically there are millions of checks which need to be processed concurrently. However, as

shown in Figure 1, the processing procedure can be modeled as a rather simple workflow with only several steps.

With the promotion of the world's leading companies, cloud computing is attracting more and more attention from researchers. Cloud computing refers to *a variety of services available over the Internet that deliver computing functionality on the service provider's infrastructure* (Boss et al., 2008). Clouds are *a large pool of easily usable and accessible*

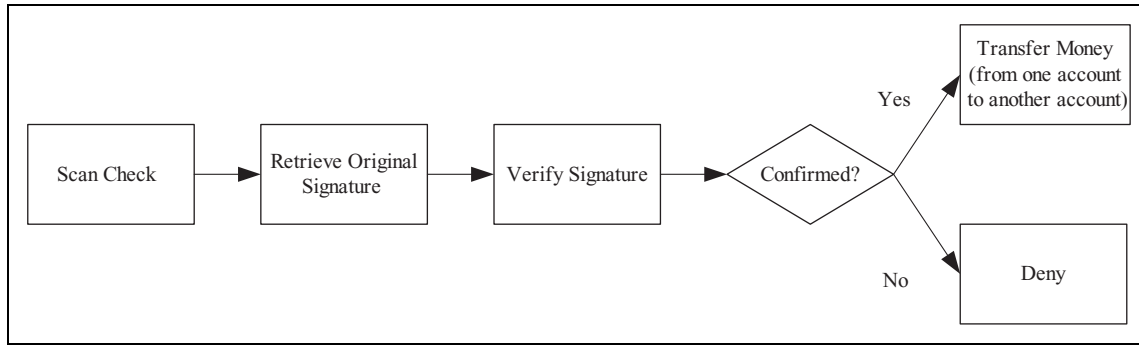
<sup>1</sup>School of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan, Hubei, China

<sup>2</sup>Faculty of Information and Communication Technologies, Swinburne University of Technology, Hawthorn, Melbourne, Australia

## Corresponding author:

Ke Liu, University of Science and Technology, Wuhan, Hubei, China 430074 and Faculty of Information and Communication Technologies, Swinburne University of Technology Hawthorn, Melbourne Australia 3122.

Email: keliuwuhan@gmail.com



**Fig. 1.** Simplified bank check workflow processing.

virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized service level agreements (Vaquero et al., 2008). It has many potential advantages which include lower cost, device and location independence, user friendliness and so on (Vaquero et al., 2008). These advantages can also benefit workflow systems built on top of a cloud. The advantages of moving workflows to a cloud include the following:

- Moving workflows to a cloud computing environment enables the utilization of various cloud services to facilitate workflow execution.
- In contrast to dedicated resources, the resources in clouds are shared and provided to users “on-demand”, meaning the expenditure on hardware for workflow execution can be eliminated.
- The “user-centric” model in a cloud computing environment makes workflow execution more user-friendly thus increasing user satisfaction.
- The “pay as you go” business model in a cloud can reduce the execution cost of workflows.

For these reasons there is a need to migrate workflow executions to cloud computing platforms. However, on a commercial cloud platform, users need to pay for what they use. Assuming these instance-intensive workflows are bounded by user budgets (i.e. cost-constrained) and executed on cloud computing platforms (i.e. cloud workflows), we call them instance-intensive cost-constrained cloud workflows. In this paper we focus on designing algorithms for scheduling these workflows effectively. We start by analyzing the requirements of these algorithms.

On one hand, users are always concerned about the execution time of workflows. On the other hand, although in reality some cloud services provided, say, by Google for searching, are free, cloud services for execution of business workflow applications are very unlikely to be free. This means users may need to pay for what they use. As users

are normally sensitive to execution cost, the cost becomes another major concern for cloud workflow applications. Given that the attributes of both time and cost are involved, it is necessary to enable users to compromise for better user satisfaction. For example, during workflow execution, in some circumstances, users may decide on the fly to pay slightly more to reduce execution time, or save execution cost by allowing longer execution time as long as the final deadline can be met.

Moreover, workflow scheduling designers should consider the characteristics of cloud computing when designing cloud workflow scheduling algorithms. Let us have a look at the essence of cloud computing first. Generally speaking, cloud computing has the following major characteristics (Boss et al., 2008): 1) it hosts a variety of different loads, including batch-style back-end jobs and interactive, user-facing applications, which means the load on the servers in a cloud is highly dynamic; 2) it allows loads to be deployed and distributed quickly through the rapid provisioning of virtual machines or physical machines, which means it is highly scalable; 3) it supports redundant, self-recovering, highly scalable programming models that allow loads to recover from many unavoidable hardware/software failures; and 4) it rebalances the allocations when needed in real time. All these characteristics should be considered in the cloud workflow scheduling algorithm design.

From the analysis above, we can summarize the primary requirements for designing scheduling algorithms for instance-intensive cost-constrained cloud workflows. First, the scheduling algorithms should take execution cost (“pay as you go”) as one key factor. Second, as an important criterion of instance-intensive workflows, mean execution time will be taken as another key factor. Third, the algorithms should facilitate multiple strategies for compromising execution time and cost with user intervention enabled on the fly. Finally, they should conform to the nature of cloud computing, particularly, the scheduling algorithms should consider the background load of the servers when calculating the task execution time. After tasks are distributed to a server, actions should be taken when scheduled tasks are not completed successfully due to hardware or software failures.

The remainder of the paper is organized as follows. In Section 2, we will describe the related work, followed by the fundamental system design for our cloud workflows in Section 3. Section 4 will discuss the details of the scheduling algorithm and Section 5 will show the benefits of our work via simulation and comparison. Finally in Section 6, we will conclude our contributions and point out future work.

## 2. Related Work

In this section, we begin by briefly overviewing cloud computing. We then introduce the related workflow scheduling work and demonstrate why existing workflow scheduling algorithms are not suitable, and why it is necessary to design a dedicated scheduling algorithm for instance-intensive cost-constrained cloud workflows.

Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the infrastructure provider by means of customized service level agreements (Vaquero et al., 2008). Currently, there are many cloud computing platforms. Companies like Google, Yahoo, Amazon, eBay and Microsoft have all built Internet consumer services like search, social networking, web e-mail and online commerce based on cloud computing. Some specific examples of cloud computing are Amazon Elastic Compute Cloud, Enomalism and MobileMe. In academia, cloud computing is becoming an area of increased focus at a rapid pace. For infrastructure design, a cloud-based infrastructure using wide area high performance networks has been proposed (Grossman et al., 2008) and a concept of market-oriented cloud computing has been described (Buyya et al., 2008). For applications, high performance data clouds have been used for data mining (Grossman and Gu, 2008) and “Storage Cloud” resources have been explored for content creators (Broberg et al., 2008).

As for cloud workflow systems, similar to many other workflow systems, scheduling is a very important component. It directly determines the performance of the whole system. According to Yu and Buyya (2007) there are two major types of workflow scheduling: best-effort based and Quality of Service (QoS) constraint based. Best-effort based scheduling attempts to minimize the execution time without considering other factors such as the monetary cost of accessing resources and various users’ QoS satisfaction levels. Some examples are the Heterogeneous Earliest-Finish-Time algorithm (Tannenbaum et al., 2002) used by ASKALON (Fahringer et al., 2005), the Min-Min algorithm (Maheswaran et al., 1999) used by GrADS (Berman et al., 2001) and a throughput maximization strategy (Liu et al., 2008) used by SwinDeW-G (Yang et al., 2007) and SwinDeW-C (Yang et al., 2008). In contrast,

QoS constraint-based scheduling attempts to maximize the performance under QoS constraints of which cost is one major constraint that we focus on in this paper. Examples include time minimization under budget constraints or cost minimization under deadline constraints, such as the back-tracking proposed by Menasc and Casalicchio (2004), the LOSS and GAIN approach implemented in CoreGrid (Sakellariou et al., 2005), a genetic algorithm approach (Yu and Buyya, 2006) and the deadline distribution algorithm (Yu et al., 2005) implemented in GridBus (Buyya and Venugopal, 2004).

In a QoS constraint-based scenario of cloud computing, the scheduling should conform to the constraints of the workflow model, typically, time-constraints for a timed workflow process model (Chen and Yang, 2008), cost-constraints for a utility workflow process model (Yu et al., 2005), or both. Generally speaking, users are often sensitive to the cost and more tolerant to the execution performance in most circumstances as long as the final deadline can be met. Thus performance-driven workflow scheduling algorithms are not very suitable for cloud workflows. In the remainder of this section we only focus on the literature related to market-driven, especially cost-constrained, workflow scheduling algorithms.

Back-tracking (Menasc and Casalicchio, 2004) assigns available tasks to the least expensive computing resources. An available task is an unmapped/unscheduled task whose parent tasks have been scheduled. If there is more than one available task, the algorithm assigns the task with the largest computational demand to the fastest resource in its available resource list. It can be seen that this algorithm does not consider the competition with other activities such as those in instance-intensive workflows.

Yu and Buyya (2006) proposed a genetic-algorithm-based approach to solve a deadline- and budget-constrained scheduling problem. For budget-constrained scheduling, the cost-fitness component encourages the formation of the solutions that satisfy the budget constraint. For deadline-constrained scheduling, it uses the genetic algorithm to choose solutions with less cost.

The LOSS and GAIN approach (Sakellariou et al., 2005) is a strategy that iteratively adjusts a schedule which is generated by a time-optimized heuristic or a cost-optimized heuristic to meet the budget constraints. A time-optimized heuristic attempts to minimize the execution time while a cost-optimized heuristic attempts to minimize the execution cost. The above two strategies usually spend a lot of time on iterations, so they are not suitable for scheduling instance-intensive workflows which have a lot of concurrent instances.

The deadline distribution algorithm called Deadline-MDP (Yu et al., 2005) tries to minimize the cost of execution while meeting the deadline. It describes task partitioning and overall deadline assignment for optimized execution planning and efficient run-time rescheduling. Their simulation indicates that it outperforms Deadline-Level and Greedy-Cost algorithms that are derived from

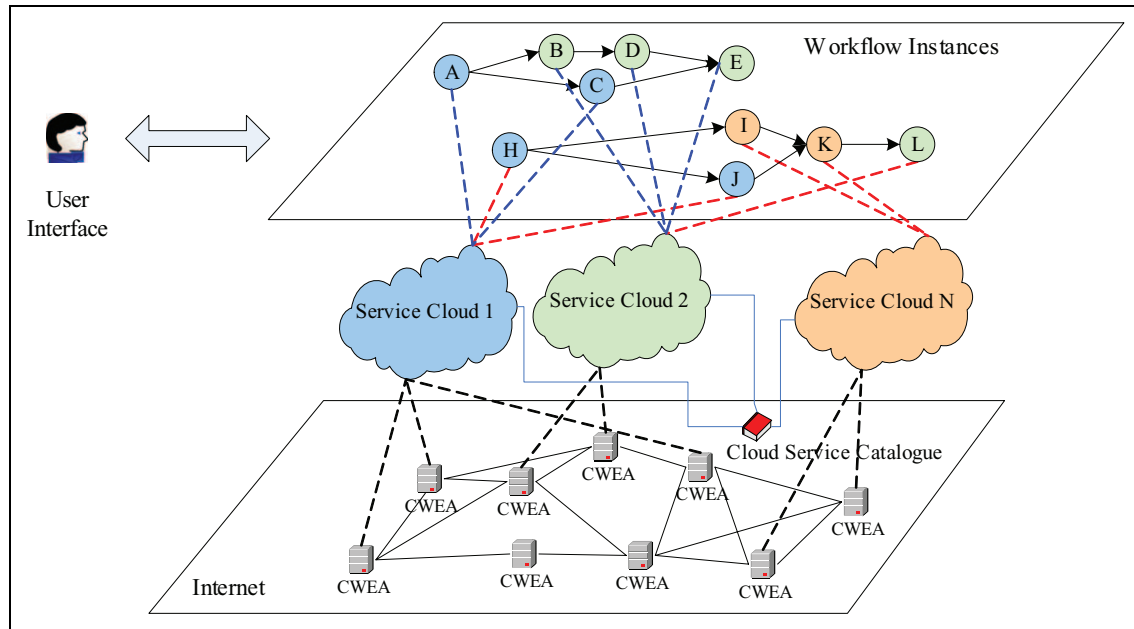


Fig. 2. Architecture of SwinDeW-C.

the cost optimization algorithm in Nimrod-G (Buyya et al., 2000). This algorithm is designed for a single instance on a utility grid rather than an instance-intensive scenario on a cloud computing platform. However, the philosophy of this algorithm can be borrowed by our algorithm for scheduling instance-intensive cost-constrained cloud workflows.

Generally speaking, the algorithms overviewed above are designed for scheduling a single workflow instance. However, for instance-intensive workflows on a cloud computing platform, fierce competition on servers/services may occur and failures may happen from time to time. Thus, our scheduling strategy needs to incorporate these situations accordingly. In addition, our scheduling algorithm needs to consider the characteristics of cloud computing in order to accommodate instance-intensive cost-constrained workflows by compromising execution time and cost with user intervention enabled on the fly which is not considered in other strategies.

### 3. SwinDeW-C System Design

As depicted in Figure 2, our SwinDeW-C (**S**winburne **D**ecentralised **W**orkflow for **C**loud) cloud workflow architecture is divided into the following major parts at a high level: dynamic service clouds, cloud workflow execution agents (CWEAs), a cloud service catalog, and a user interface. The other general parts of a cloud computing platform such as system management, provisioning tools, monitoring and metering are not shown explicitly in the figure because we focus on the discussion of cloud workflow systems.

#### 3.1. Service Clouds

Servers with the same service are organized dynamically as a service cloud. It should be noted that all service clouds are

independent of each other and every server will automatically join the service clouds according to the services it can provide. In a service cloud, every server tracks the information of its neighbors so that scheduling is always performed cooperatively with its neighbors. In the case when a cloud is too big, it can be divided into several sub-clouds according to servers' geographical positions in order to reduce communication delays.

#### 3.2. Cloud Workflow Execution Agents (CWEAs)

To efficiently manage services, each service in a cloud service is administered by a cloud workflow execution agent (CWEA). The CWEA manages the service during workflow scheduling and execution, including service monitoring, cooperative scheduling and execution with other agents which manage the services of the same type.

The task allocation among service clouds can be described as follows. After a task requesting a specific service for execution has been scheduled to a service cloud, a suitable node in the service cloud will be selected for execution according to the applied strategy, such as achieving load-balance, minimum cost or shortest distance to the task host, etc. Our previous work on SwinDeW peer-to-peer workflows (Yan et al., 2006) and SwinDeW-G Grid workflows (Yang et al., 2007) formed the foundation for designing CWEA.

#### 3.3. Cloud Service Catalog

Cloud services are the foundation of cloud computing. There are a variety of services available over the Internet that deliver compute functionality on the service provider's infrastructure. To access the services, there is a cloud service catalog which is a list of services that a user can

**Table 1.** Overview of the CTC algorithm

Algorithm: CTC Scheduling	
	<b>Input:</b> An array of workflow instances
	<b>Output:</b> A schedule (lowest cost by default without losing generality)
Pre-Step	0.1 Check uncompleted tasks and schedule them first in this round.
Step 1	1.1 Calculate the sub-deadlines for tasks of the last instance. 1.2 Calculate the sub-deadlines for tasks of other instances based on the last instance assuming that the schedule follows the stream-pipe mode to avoid competitions for cheaper services.
Step 2	2.1 Calculate the estimated execution time and cost for each service
Step 3	3.1 Allocate each task to the service which gives the execution time that does not exceed the sub-deadline and the lowest total cost.
Step 4	4.1 Provide a just-in-time time-cost relationship graph for user to optionally choose an updated compromised deadline for scheduling.
Step 5	5.1 Repeat the above steps for next round scheduling

request. In workflow systems, a task usually needs a cloud service to execute. This cloud service is registered along with other cloud services by service providers on the cloud service catalog server. Usually, the items in the catalog are maintained by their corresponding service clouds via the heartbeat mechanism.

### 3.4. User Interface

SwinDeW-C provides a user interface for users to monitor the workflow execution status and, more importantly, to provide input on the setting of compromised time and cost on the fly if needed. The input from the user on time and cost will be taken into account for scheduling the next round of tasks in order to achieve better user satisfaction.

## 4. CTC Scheduling Algorithm

Considering the nature of cloud computing and the application scenario of instance-intensive cost-constrained workflows, we propose a compromised-time-cost (CTC) scheduling algorithm which focuses on the following perspectives against the requirements.

First, considering the pay-per-use feature of cloud workflows, the CTC algorithm takes execution cost and execution time as the two key considerations. For simplicity without losing generality, the CTC algorithm described in this paper is to minimize the cost under certain user-designated deadlines because the corresponding algorithm for minimizing the execution time under certain user-designated cost is similar.

Second, different from other scheduling algorithms, our CTC algorithm always allows compromising between execution cost and time. The algorithm provides a just-in-time graph of the time-cost relationship during workflow execution in the user interface for users to choose an acceptable compromise before the next round of scheduling begins if they wish. If no user input is detected at the time of the next round of scheduling, the default scheduling strategy will be automatically applied so no delay will be caused. More details are in Step 4 of the algorithm described later.

Third, the algorithm considers sharing, conflicting and competition of services caused by multiple concurrent instances running on the highly dynamic cloud computing platform. For example, the lower cost services that are normally used when only considering a single workflow instance, may be heavily used and, hence, unavailable at the time whilst waiting for their availability could cause a significant and unnecessary delay.

Therefore, our algorithm needs to consider the following: 1) background load: in order to estimate the execution time more accurately, background load is considered when calculating the task execution time on each specific server; 2) dynamic adjustment: in order to adapt to a dynamic change of load, after tasks are distributed to a server, the server may reschedule the tasks when it encounters a heavy load; 3) checking and rescheduling: in order to deal with execution failures, uncompleted tasks will be rescheduled with a high priority for the next round.

Accordingly, our scheduling algorithm has a pre-step to discover and reschedule the tasks which failed to complete due to various reasons on the cloud computing platform or distribute tasks to other light-load servers when the host server encounters a heavy load.

Before we give the details of the steps, in order to have an overview, we summarize the CTC algorithm in Table 1 and present the algorithm details next.

*Pre-Step: Check uncompleted tasks and schedule them first in this round.* The main purpose of this step is to discover the unsuccessful tasks or redistribute tasks to other light-load servers when the host server encounters a heavy load. For the former case, each task has been allocated a timer once it is scheduled to other servers. If a timer of a certain task expires and the task has not yet been completed, the execution will be considered unsuccessful. For the latter case, all tasks allocated to such host servers have to be rescheduled. In order to give a higher priority to these tasks, they will be scheduled first before new available tasks.

*Step 1: Allocate sub-deadlines to tasks for each instance.* Similar to the overall deadline of the entire workflow instance,

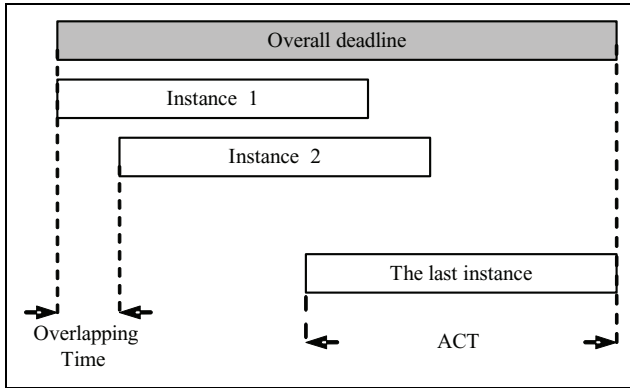


Fig. 3. Stream-pipe mode sub-deadline distribution.

a sub-deadline is the latest completion time allocated to a single task, which the completion time of the task should not exceed. In Deadline-MDP (Yu et al., 2005), the overall deadline is divided over task partitions in proportion to their minimum processing time. This partition works well for a single instance. As for multiple concurrent instances of instance-intensive cost-constrained workflows, it is not ideal to calculate all sub-deadlines for every instance because it is time consuming and not accurate enough due to the service competition involved.

The CTC algorithm uses a more practical method to estimate sub-deadlines than that used by Deadline-MDP. It only calculates sub-deadlines of tasks of the last instance of the same type once and deducts sub-deadlines of others from that result. The procedure is described as follows: let the average completion time on all available services of the instance be the overall deadline to be distributed, for sequential tasks, sub-deadlines are distributed in proportion to their average processing time on all available services; and for branches with several sequential tasks, they are resolved by modeling the branch as a Markov Decision Process. The details can be found in Sulistio and Buyya (2004). Although the Markov Decision Process may be relatively time-consuming, considering it is only needed to be calculated once for all instances generated from the same workflow, the scheduling time cost is quite acceptable.

With the sub-deadlines of the last instance calculated, we can deduct the sub-deadlines for tasks of other instances. In this algorithm, we assume the schedule follows the stream-pipe mode. The reason for this assumption is to stagger the deadlines of the large number of concurrent instances of the same nature to avoid fierce competition for cheaper services. We will see later in this step that if we set the sub-deadlines of these instances to be the same, the service utilization of cheaper services will drop due to competition within a certain period of time. However, if we stagger the deadlines, the tasks with later sub-deadlines will have a chance to use cheaper services after they are released by tasks with earlier sub-deadlines. Of course, these workflow instances can still be executed in parallel as long as possible and the completion time of each task

does not exceed its sub-deadline allocated. With this assumption, from Figure 3, we have:

$$ACT + (n - 1) \times OT = OD \quad (1)$$

where ACT is the average completion time, OT is the overlapping time and OD is the overall deadline.

After the overlapping time is calculated, we can calculate sub-deadlines for tasks of other instances based on the sub-deadline distribution of the last task. The sub-deadline of each task of the  $i$ th instance is equal to the sub-deadline of the corresponding task of the last instance minus  $(n - i) \times$  overlapping time where  $n$  is the total number of workflow instances, except the last task whose sub-deadline equals the overall deadline.

The comparison of scheduling results based on the different deadline distribution methods of Deadline-MDP and our CTC algorithm is shown in Figure 4. For convenience, we use  $I_i T_j$  to represent the  $j$ th task of the  $i$ th instance. The details of the example are listed below:

There are four workflow instances from the same workflow model that need to be scheduled, and the user gives a total deadline of 14 seconds. The directed acyclic graph (DAG) of the workflow model is shown on the top of Figure 4.

The estimated execution times of tasks  $T_1$ ,  $T_2$  and  $T_3$  are 2, 3 and 2 seconds, respectively, and the minimum execution times follow this ratio. The estimated execution costs of tasks  $T_1$ ,  $T_2$  and  $T_3$  are 20, 30 and 40 dollars respectively, and the minimum execution costs also follow this ratio.

Task 1 can be executed on Service 1 with an execution speed of half the estimated execution speed and an execution cost of half the estimated execution cost, or executed on Service 2 with an execution speed of double the average execution speed and an execution cost of double the estimated execution cost.

Task 2 can be executed on Service 3 with an execution speed of exactly the estimated execution speed and an execution cost of exactly the estimated execution cost, or on Service 4 with an execution speed of triple the estimated execution speed and an execution cost of triple the estimated execution cost.

Task 3 can be executed on Service 5 with an execution speed of half the estimated execution speed and an execution cost of half the estimated execution cost, or on Service 6 with an execution speed of exactly the estimated execution speed and an execution cost of exactly the estimated execution cost.

The communication time of each of the two tasks executed on each of the two services is exactly 1 second.

According to Deadline-MDP, the overall deadline is divided over task partitions in proportion to their minimum processing time. Giving the overall deadline of 14, after applying this policy to Instance 1, we will get  $sub\_deadline(I_1 T_1) = 4$ ,  $sub\_deadline(I_1 T_2) = 10$  and  $sub\_deadline(I_1 T_3) = 14$ . As Deadline-MDP does not stagger the deadlines of

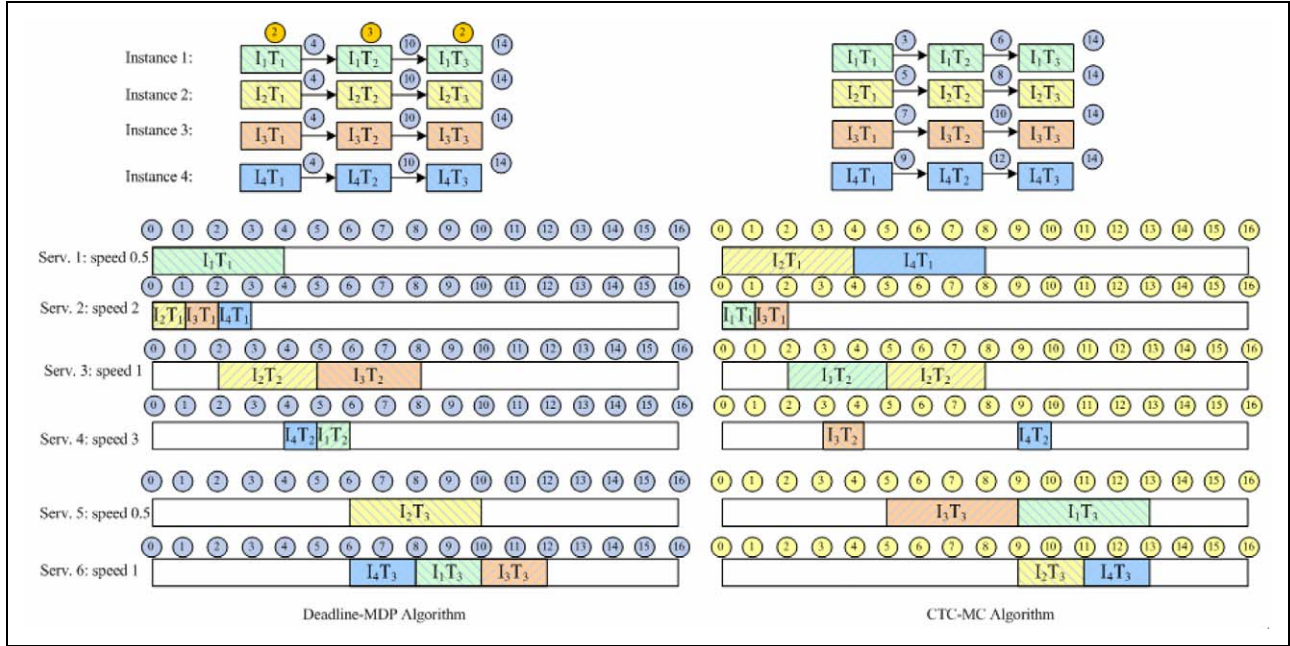


Fig. 4. Comparison of deadline distributions.

Table 2. Service Processing Speeds and Corresponding Prices for Task Execution

Service ID	Processing speed	Cost (\$/Task)
1	0.5	3
2	1	6
3	2	12
4	4	24

Table 3. Service Bandwidths and Corresponding Prices for Data Transmission

Service ID	Bandwidth (Mbps)	Cost (\$/MB)
1	100	1
2	1,000	10
3	10,000	100
4	100,000	1,000

instances, the sub-deadlines of the corresponding tasks of other instances are equal to the calculated sub-deadlines.

As for CTC, according to Equation 1,  $ACT + (n - 1) \times OT = OD$ , here  $ACT = 2 + 3 + 2 = 7$ ,  $n = 4$  and  $OD = 14$ , so  $OT = 2$ . For the last instance  $I_4$ , we have

$sub\_deadline(I_4T_3) = OD = 14$ ,  $sub\_deadline(I_4T_2) = sub\_deadline(I_4T_3) - estimated\_execution\_time(I_4T_3) = 14 - 2 = 12$ ,  $sub\_deadline(I_4T_1) = sub\_deadline(I_4T_2) - estimated\_execution\_time(I_4T_2) = 12 - 3 = 9$ .

Knowing the sub-deadlines of the last instance, we now deduce the sub-deadlines of other instances as shown in Figure 3:

$sub\_deadline(I_3T_3) = OD = 14$ ,  $sub\_deadline(I_3T_2) = sub\_deadline(I_4T_2) - (4 - 3) \times 2 = 12 - 2 = 10$ ,  $sub\_deadline(I_3T_1) = sub\_deadline(I_4T_1) - (4 - 3) \times 2 = 9 - 2 = 7$ .

$sub\_deadline(I_2T_3) = OD = 14$ ,  $sub\_deadline(I_2T_2) = sub\_deadline(I_4T_2) - (4 - 2) \times 2 = 12 - 4 = 8$ ,  $sub\_deadline(I_2T_1) = sub\_deadline(I_4T_1) - (4 - 2) \times 2 = 9 - 4 = 5$ .

$sub\_deadline(I_1T_3) = OD = 14$ ,  $sub\_deadline(I_1T_2) = sub\_deadline(I_4T_1) - (4 - 1) \times 2 = 12 - 6 = 6$ ,  $sub\_deadline(I_1T_1) = sub\_deadline(I_4T_1) - (4 - 1) \times 2 = 9 - 6 = 3$ .

As the completion time of each task cannot exceed its sub-deadline, as shown in Figure 4, in the Deadline-MDP algorithm, only  $I_1T_1$  can be scheduled on a potentially (much) cheaper service, for example, Service 1, while other tasks have to be scheduled on a more expensive service, for example Service 2. Thus, the mean execution cost would increase. By contrast, the CTC algorithm uses a different deadline distribution method,  $I_2T_1$  and  $I_4T_1$  can both be scheduled on the cheaper service, i.e. Service 1, while not exceeding the calculated deadlines. Therefore, the mean execution cost would be reduced accordingly. The detailed scheduling result of these two algorithms are shown in Figure 4, and we can see that CTC has a lower total cost than Deadline-MDP, for there are more tasks that are scheduled on cheaper resources.

**Step 2: Calculate the estimated execution time and cost for each service.** First, let us list the price tables. In Table 2 it can be seen that a higher processing speed, which reduces the execution time, usually ends up with a higher cost. Table 3 shows that a higher bandwidth, which reduces the transmission time, normally results in a higher cost. The round trip

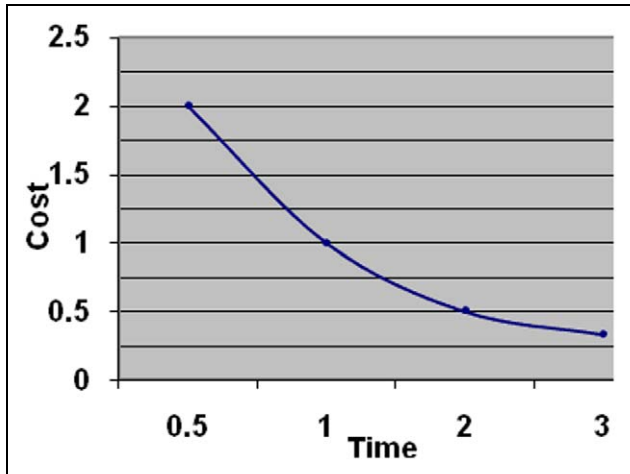


Fig. 5. Example of time-cost relationship graph.

time is not listed here explicitly, but is considered when calculating transmission time.

- 1) For each service ID, suppose the processing speed is PS and the current load is L, we can calculate the execution time and cost for executing a task on this service as follows

$$ET = OET/PS/L \quad (2)$$

where ET is the execution time, OET<sup>1</sup> is the original execution time, PS is the processing speed and L is the load.

$$EC = \text{cost}(PS) \quad (3)$$

where EC is the execution cost, cost(PS) is a function whose value can be looked up in Table 2.

- 2) Suppose the round trip time from the service on which the previous task is executed is RTT, for each service ID we calculate the transmission time and cost as follows

$$TT = RTT + DS/B \quad (4)$$

where TT is the transmission time, DS is the data size and B is the bandwidth.

$$TC = \text{cost}(B) \quad (5)$$

where TC is the transmission cost and cost(B) is a function whose value can be looked up in Table 3.

- 3) Then the total execution time and cost can be calculated as follows in formulas (6) and (7) respectively:

$$FCT = \max(SAT, DAT) + ET \quad (6)$$

where FCT is the final completion time, SAT is the service available time and equals to previous SAT + ET and DAT is the data available time and equals to previous DAT + TT.

$$FTC = EC + TC \quad (7)$$

where FTC is the final total cost.

It should be noted that FCT and FTC are calculated one by one, following the rule of FCFS (First Come First Served).

**Step 3: Allocate tasks to services.** Following the FCFS rule, the CTC algorithm allocates each task to the corresponding service which gives the execution time that does not exceed the sub-deadline at the lowest total cost.

**Step 4: Provide just-in-time time-cost relationship graph for the user to optionally choose an updated compromised deadline for scheduling.** For the user's convenience, the CTC algorithm provides a time-cost relationship graph on the fly for the user, optionally, to decide the scheduling objective at the time of each scheduling round. During this on-demand interaction, a user can decide whether to reduce the execution time by paying (slightly) more or (slightly) postpone the deadline to reduce the execution cost. An example of such a graph is shown in Figure 5. The user has the opportunity to redefine the deadline or cost in each scheduling round on the fly to achieve more satisfactory compromised-time-cost outcomes.

It should be noted that the algorithm currently described is the strategy to achieve low execution cost when the user sets the deadline on the fly. For simplicity, without loss of generality, we do not address the other strategy of the algorithm that is similar to the procedure above with cost and time swapped and can achieve a faster execution time when the user sets the execution cost on the fly. The provision of two strategies gives the users maximum flexibility and, hence, more satisfaction.

**Step 5: Next round scheduling.** After the user has chosen an updated compromised deadline, the CTC algorithm will repeat all the steps to find a schedule conforming to the objective based on the new deadline for the next round. If no user input is provided, the default scheduling strategy will be applied.

## 5. Simulation and Comparison

In this section, we will simulate the instance-intensive cost-constrained cloud workflow scenario on our SwinDeW-C platform, and compare the mean execution cost of concurrent workflow instances using our CTC algorithm in comparison to the other representative scheduling algorithm, namely, the Deadline-MDP algorithm (Yu et al., 2005) which is more effective than Deadline-Level and Greedy-Cost algorithms as overviewed in Section 2.

<sup>1</sup>OET is the execution time of the task on a standard reference service.

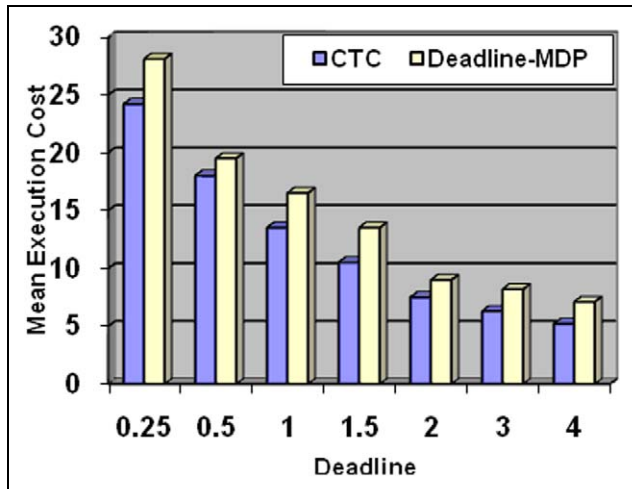


Fig. 6. Comparison of mean execution cost.

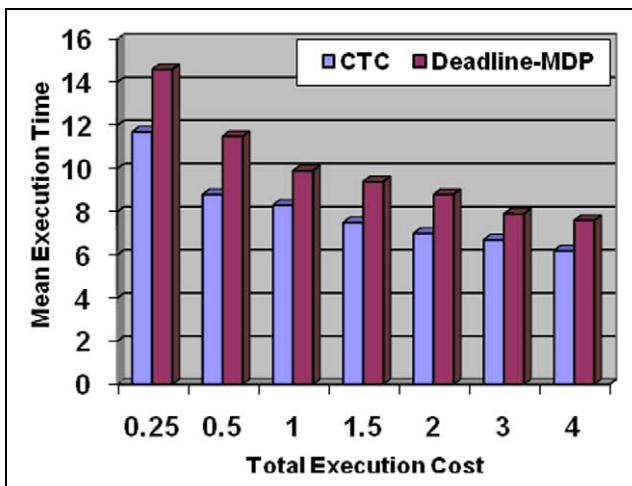


Fig. 7. Comparison of mean execution time.

1) *Criteria for comparison.* Makespan (execution time) is the time spent from the beginning of the first task to the end of the last task for a workflow instance. It is normally the most popular criterion for workflow scheduling algorithms. The shorter the mean execution time, the better the performance.

As cost is also a very significant factor for cloud workflows, users normally need to pay every time they use cloud services. Workflow systems should be designed to reduce costs for users. Usually as long as the completion time does not exceed the deadline, the smaller the cost is, the more satisfied the users are.

In a batch mode scenario using our stream-pipe technique for scheduling instance-intensive cost-constrained cloud workflows, mean execution time and mean execution cost are more suitable for assessing the scheduling outcome. Therefore, we compare our CTC algorithm with Deadline-MDP against the above two criteria.

2) *Simulation method.* To simulate instance-intensive cost-constrained cloud workflows, we construct the simulation environment focusing on the following:

- We have initialized a DAG database which includes all kinds of DAGs that represent instance-intensive workflows. Each of these workflows is a simple workflow with only 5–10 tasks, and the duration of each task is 0.1–1 second. These DAGs are generated in advance from practical instance-intensive workflows, such as bank check workflow processing.
- In order to simulate the huge number of concurrent instances, we use a workflow instance generator which creates 1,000 instances per second and a total number of 1,000,000 instances are used to test the performance of different algorithms.
- In a commercial cloud computing platform, there are different types of services for selection. We provide services with different performances based on Table 2 and services with different transmission speeds based on Table 3 for the algorithms to choose from. For the purpose of comparison with Deadline-MDP, in Table 2 and Table 3 we use service sets of the same style as used by Deadline-MDP. However, using services with different parameters has shown similar results.
- For fault tolerance testing, we set an execution failure ratio for each task, which means that each task has a small possibility that it would fail to complete. If a task failed to complete, it will be processed with higher priority in the next scheduling round as described in Section 4.
- The servers were executing other tasks when we did the simulation, thus they had a dynamic background load.

The simulation includes two processes. The first calculates the actual mean execution cost from 0.25 to 4 for the input deadline, using the CTC-MC and Deadline-MDP algorithms, and the second calculates the actual mean execution time from 0.25 to 4 for the input cost, using the CTC-MT and Deadline-MDP algorithms, respectively, to compare their performance.

3) *Comparison of results.* Figure 6 demonstrates the comparison results of the mean execution cost with different deadlines. Given both algorithms meet the deadlines, it can be seen that the mean execution cost of our CTC algorithm is always lower than that of the Deadline-MDP algorithm in all circumstances. On average, the saving on the mean execution cost is over 15%.

Figure 7 demonstrates the comparison of results of the mean execution time for different execution costs. Given both algorithms do not exceed the execution cost, it can be seen that the mean execution time of our CTC algorithm is always shorter than that of the Deadline-MDP algorithm in all circumstances. On average, the saving on the mean execution time is over 20%.

## 6. Conclusions and Future Work

Although cloud computing is a new paradigm, it has been rapidly accepted by both industry and academia all over the

world. Meanwhile, instance-intensive cost-constraint workflows are very suitable for execution on a cloud computing platform. As scheduling is always a significant component of such workflow systems, it is necessary to design workflow scheduling algorithms suitable for cloud computing. This paper has proposed an innovative instance-intensive cost-constrained cloud workflow scheduling algorithm named CTC (compromised-time-cost) which takes cost and time as the main concerns with user intervention on the fly enabled and incorporates the characteristics of cloud computing. The simulation has demonstrated that our CTC algorithm can achieve a lower cost than others while meeting the user-designated deadline or reduce the mean execution time than others within the user-designated execution cost.

In the future, we will further develop SwinDeW-C (Swinburne Decentralised Workflow for Cloud) for the rapidly evolving cloud computing environment. In addition, we will also try to develop more cloud workflow scheduling algorithms for various scenarios.

### Acknowledgement

The research work reported in this paper is partly supported by the Australian Research Council under grant No. LP0990393, the National Science Foundation of China under grant No. 90412010 and the ChinaGrid project from the Ministry of Education of China. We are grateful to B. Gibson for his English proofreading.

### References

- Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crummey, J., Reed, D., Torczon, L. and Wolski, R. (2001). The GrADS project: software support for high-level grid application development. *International Journal of High Performance Computing Applications* **15**(4): 327–344.
- Boss, G., Malladi, P., Quan, D., Legregni, L. and Hall, H. (2008). Cloud computing (White Paper), IBM, October 2007, [http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud\\_computing\\_wp\\_final\\_8Oct.pdf](http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf), accessed on March 19, 2009.
- Broberg, J., Buyya, R. and Tari, Z. (2008). MetaCDN: Harnessing ‘storage clouds’ for high performance content delivery, *Technical Report*, GRIDS-TR-2008-10, Grid Computing and Distributed Systems Laboratory, the University of Melbourne, Australia, Aug. 15, 2008. <http://www.gridbus.org/reports/meta-cdn2008.pdf>, accessed on March 19, 2009.
- Buyya, R., Giddy, J. and Abramson, D. (2000). An evaluation of economy based resource trading and scheduling on computational power grids for parameter sweep applications. In Proceedings of the 2nd Workshop on Active Middleware Services (AMS 2000), August, pp. 1–10, Pittsburgh, USA.
- Buyya, R. and Venugopal, S. (2004). The Gridbus toolkit for service oriented grid and utility computing: an overview and status report. In Proceedings of the 1st IEEE International Workshop on Grid Economics and Business Models (GECON 2004), April, pp. 19–36, Seoul, Korea.
- Buyya, R., Yeo, C. S. and Venugopal, S. (2008). Market-oriented cloud computing: vision, hype, and reality for delivering its services as computing utilities. In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, September, pp. 5–13, Los Alamitos, CA, USA.
- Chen, J. and Yang, Y. (2008). Activity completion duration based checkpoint selection for dynamic verification of temporal constraints in grid workflow systems. *International Journal of High Performance Computing Applications* **22**(3): 319–329.
- Fahringer, T., Jugravu, A., Pllana, S., Prodan, R., Slovis, Jr. C. and Truong, H. L. (2005). ASKALON: A tool set for cluster and grid computing. *Concurrency and Computation: Practice and Experience* **17**: 143–169.
- Grossman, R. L. and Gu, Y. (2008). Data mining using high performance data clouds: experimental studies using sector and sphere. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August, pp. 920–927, Las Vegas, Nevada, USA.
- Grossman R. L., Gu Y., Sabala M. and Zhang W. (2008). Compute and storage clouds using wide area high performance networks. *Future Generation Computer Systems* **25**: 179–183.
- Liu, K., Chen, J., Yang, Y. and Jin, H. (2008). A throughput maximization strategy for scheduling instance intensive workflows on SwinDeW-G. *Concurrency and Computation: Practice and Experience* **20**(15): 1807–1820.
- Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D. and Freund, R. F. (1999). Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, *J. Parallel and Distributed Computing*, **59**(2): 107–131, 1999.
- Menasc, D. A. and Casalicchio, E. (2004). A framework for resource allocation in grid computing. In Proceedings of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS’04), pp. 259–267.
- Sakellariou, R., Zhao, H., Tsiakkouri, E. and Dikaiakos, M. D. (2005). Scheduling workflows with budget constraints, CoreGRID. In Proceedings of the Workshop on Integrated research in Grid Computing, November, pp. 347–357, Pisa, Italy.
- Sulistio, A. and Buyya, R. (2004). A grid simulation infrastructure supporting advance reservation. In Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), November, pp. 1–7, Boston, USA.
- Tannenbaum, T., Wright, D., Miller, K. and Livny, M. (2002). Condor – a distributed job scheduler. In Computing with Linux. MIT Press.
- Vaquero, L. M., Rodero-Merino, L., Caceres, J. and Lindner, M. (2008). A Break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review* **39**(1): 50–55.
- Yan, J., Yang, Y. and Raikundalia, G. K. (2006). SwinDeW - A peer-to-peer based decentralized workflow management system. *IEEE Transactions on Systems, Man and Cybernetics, Part A* **36**(5): 922–935.

- Yang, Y., Liu, K., Chen, J., Lignier, J. and Jin, H. (2007). Peer-to-peer based grid workflow runtime environment of SwinDeW-G. In Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science07), December, pp. 51–58, Bangalore, India.
- Yang, Y., Liu, K., Chen, J., Liu, X., Yuan, D. and Jin, H. (2008). An algorithm in SwinDeW-C for scheduling transaction-intensive cost-constrained cloud workflows. In Proceedings of the 4th IEEE International Conference on e-Science and Grid Computing (e-Science08), December, pp. 374–375, Indianapolis, USA.
- Yu, J., Buyya, R. and Tham, C. K. (2005). A cost-based scheduling of scientific workflow applications on utility grids. In Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing, December, pp. 140–147, Melbourne, Australia.
- Yu, J. and Buyya, R. (2006). Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming Journal* **14**(3–4): 217–230.
- Yu, J. and Buyya, R. (2007). Workflow scheduling algorithms for grid computing. *Technical Report*, GRIDS-TR-2007-10, Grid Computing and Distributed Systems Laboratory, the University of Melbourne, Australia, May 2007, <http://gridbus.csse.unimelb.edu.au/reports/WorkflowSchedulingAlgs2007.pdf>, accessed on March 19, 2009.

### Author's Biographies

*Ke Liu* was born in Hubei, China. He received a bachelor's degree from Nanjing University of Science and Technology, Nanjing, China in 1999 and a Master of Engineering degree from Huazhong University of Science and Technology, Wuhan, China, in 2002, both in computer science. He is currently a research student, as a PhD candidate, at Huazhong University of Science and Technology, Wuhan, China. As a joint student, he is also a PhD candidate in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. His research interests include P2P, grid and cloud computing based workflow systems and P2P live streaming.

*Hai Jin* is a professor of computer science and engineering at the Huazhong University of Science and Technology (HUST) in China. He is now dean of the School of Computer Science and Technology at HUST. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at the University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the

chief scientist of ChinaGrid, the largest grid computing project in China. Jin is a senior member of the IEEE and a member of the ACM. Jin is a member of the Grid Forum Steering Group (GFSG). His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage and network security.

*Jinjun Chen* received his PhD degree in Computer Science and Software Engineering from Swinburne University of Technology, Melbourne, Australia in 2007. He is currently a lecturer in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. His research interests include scientific workflow management and applications, workflow management and applications in web service or SOC environments, workflow management and applications in grid (service)/cloud computing environments, software verification and validation in workflow systems, QoS and resource scheduling in distributed computing systems such as cloud computing, service-oriented computing (SLA, negotiation, engineering, composition), semantics and knowledge management and cloud computing.

*Xiao Liu* received his master's degree in management science and engineering from Hefei University of Technology, Hefei, China, 2007. He is currently a PhD student in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. His research interests include workflow management systems, scientific workflows, business process management and data mining.

*Dong Yuan* was born in Jinan, China. He received his bachelor's degree and master's degree from Shandong University, Jinan, China in 2005 and 2008, respectively, both in computer science. He is currently a research student, as a PhD candidate, in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. His research interests include data management and scheduling algorithms in P2P, grid and cloud computing based workflow systems.

*Yun Yang* was born in Shanghai, China. He received a Master of Engineering degree from the University of Science and Technology of China, Hefei, China, in 1987, and a PhD degree from The University of Queensland, Brisbane, Australia, in 1992, both in computer science. He is currently a full professor at the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. Prior to joining Swinburne as an associate professor in late 1999, he was a lecturer and

then senior lecturer at Deakin University during 1996–1999. Before that he was a research scientist at DSTC – the Cooperative Research Centre for Distributed Systems Technology during 1993–1996. He also worked at Beihang University in China during 1987–1988. He has published

more than 160 papers in journals and refereed conferences. His research interests include software engineering; P2P, grid and cloud computing based workflow systems; service-oriented computing; Internet computing applications; and CSCW.