J Supercomput (2013) 63:256–293
DOI 10.1007/s11227-011-0578-4

A market-oriented hierarchical scheduling strategy in cloud workflow systems

Zhangjun Wu · Xiao Liu · Zhiwei Ni · Dong Yuan · Yun Yang

Published online: 15 March 2011

Abstract A cloud workflow system is a type of platform service which facilitates the automation of distributed applications based on the novel cloud infrastructure. One of the most important aspects which differentiate a cloud workflow system from its other counterparts is the market-oriented business model. This is a significant innovation which brings many challenges to conventional workflow scheduling strategies. To investigate such an issue, this paper proposes a market-oriented hierarchical scheduling strategy in cloud workflow systems. Specifically, the service-level scheduling deals with the Task-to-Service assignment where tasks of individual workflow instances are mapped to cloud services in the global cloud markets based on their functional and non-functional QoS requirements; the task-level scheduling deals with the optimisation of the Task-to-VM (virtual machine) assignment in local cloud data centres where the overall running cost of cloud workflow systems will be minimised given the satisfaction of QoS constraints for individual tasks. Based on our hierarchical scheduling strategy, a package based random scheduling algorithm is presented as the

Z. Wu · Z. Ni
Institute of Intelligent Management, School of Management, Hefei University of Technology, Hefei, Anhui 230009, China

Z. Wu
e-mail: wuzhangjun@hfut.edu.cn

Z. Ni
e-mail: nzwgd@hfut.edu.cn

Z. Wu · X. Liu (✉) · D. Yuan · Y. Yang
Faculty of Information and Communication Technologies, Swinburne University of Technology, Hawthorn, Melbourne 3122, Australia
e-mail: xliu@swin.edu.au

D. Yuan
e-mail: dyuan@swin.edu.au

Y. Yang
e-mail: yyang@swin.edu.au

candidate service-level scheduling algorithm and three representative metaheuristic based scheduling algorithms including genetic algorithm (GA), ant colony optimisation (ACO), and particle swarm optimisation (PSO) are adapted, implemented and analysed as the candidate task-level scheduling algorithms. The hierarchical scheduling strategy is being implemented in our SwinDeW-C cloud workflow system and demonstrating satisfactory performance. Meanwhile, the experimental results show that the overall performance of ACO based scheduling algorithm is better than others on three basic measurements: the optimisation rate on makespan, the optimisation rate on cost and the CPU time.

Keywords Cloud workflow system · Cloud computing · Workflow scheduling · Hierarchical scheduling · Metaheuristics

1 Introduction

Cloud computing is emerging as the latest distributed computing paradigm and attracts increasing interests of researchers in the area of Distributed and Parallel Computing [33], Service Oriented Computing [1], and Software Engineering [37]. Though there is yet no consensus on what is Cloud, but some of its distinctive aspects as proposed by Ian Foster in [17] can be borrowed for an insight: "Cloud computing is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualised, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet." Compared with the definitions of conventional computing paradigms such as cluster [29], grid [16], and peer-to-peer (p2p) [45], "*economies*" is a noticeable keyword in cloud computing which has been neglected by others. "*Economies*" denotes that cloud computing adopts a market-oriented business model where users are charged for consuming cloud services such as computing, storage, and network services like conventional utilities in everyday life (e.g. water, electricity, gas, and telephony) [2]. Meanwhile, cloud service providers are obligated to provider satisfactory QoS (quality of service) based on business service contracts. It is evident that cloud computing is becoming the latest driving force to deliver computing as the fifth utility besides the previous efforts on such as utility based grid computing [34].

Cloud can generally provide three levels of services: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service). As described in [17], a cloud workflow system can be regarded as a type of platform service which facilitates the automation of distributed large-scale e-business and e-science applications in the cloud [24, 42]. As a cloud service itself, a cloud workflow system belongs to a specific service provider and under the management of its cloud resource managers. However, as a platform service, a cloud workflow system possesses the ability to get access to other cloud services and facilitate the service level agreements (SLA) [2, 26, 36]; and the ability to control the underlying cloud resources (e.g. computing and storage resources) in its own data centre.

Generally speaking, the function of a cloud workflow system and its role in a cloud computing environment, is to facilitate the automation of user submitted workflow applications where the tasks have precedence relationships defined by graph-based modelling tools such as DAG (directed acyclic graph) and Petri Nets [22, 40], or language-based modelling tools such as XPDL (XML Process Definition Language) [48]. For conventional applications with independent tasks, resource managers usually employ scheduling strategies with relatively simple heuristics such as FCFS (first come, first served), Min-Min and Max-Min [39, 50] to satisfy the QoS constraints such as on time and/or cost. On the contrary, cloud workflow applications with dependent tasks are managed by cloud workflow systems which require more sophisticated scheduling strategies to satisfy QoS constraints as well as the precedence relationships between workflow tasks. Moreover, with the increasing demand for process automation in the cloud, especially for large-scale collaborative and distributed e-business and e-science applications [13, 24, 38], the investigation on cloud workflow scheduling strategies is becoming a significant issue not only in the area of cloud workflow systems but also general cloud computing.

Among many others, one of the most important aspects which differentiate a cloud workflow system from its other counterparts is the *market-oriented* business model [2, 17]. Such a seemed small change actually brings significant innovation to conventional computing paradigms since they are usually based on non-business community models where resources are shared and free to be accessed by community members [48]. Specifically, given such a *market-oriented* business model, in this paper, we identify the following two changes in comparison to conventional computing paradigms which may bring major challenges to cloud workflow scheduling strategies: (1) from best-effort based scheduling to QoS-constraint based scheduling; (2) from specific application oriented scheduling to general service oriented scheduling. Details will be presented in Sect. 3.

In this paper, to adapt to the above changes and address the consequent challenges brought by them, we propose a market-oriented hierarchical scheduling strategy in cloud workflow systems. Specifically, our strategy includes a service-level scheduling stage and a task-level scheduling stage. Based on workflow specifications (including task definitions, process structures, and QoS constraints), in the service-level scheduling stage, the global scheduler assigns workflow tasks to candidate services in the global cloud markets and negotiate service level agreements (SLA). For service-level scheduling, we present a package based random scheduling algorithm which can generate service-level scheduling plans in an efficient fashion. In the task-level scheduling stage, given the QoS constraints for both workflow tasks and non-workflow tasks, the local scheduler optimises the Task-to-VM (virtual machine) assignment in its own data centre with specific metaheuristic scheduling algorithms. In this paper, three representative metaheuristic scheduling algorithms including genetic algorithm (GA), ant colony optimisation (ACO), and particle swarm optimisation (PSO) are investigated as the candidate task-level scheduling strategies and their performance is compared in our SwinDeW-C cloud workflow system. The work presented in this paper aims to provide a tentative investigation on workflow scheduling strategies in market-oriented cloud workflow systems where satisfactory workflow QoS can be guaranteed for user submitted workflow applications while in the meantime, the system running cost of cloud workflow systems can be decreased through dynamic optimisation.

The remainder of the paper is organised as follows. Section 2 introduces a general architecture of the cloud workflow system and presents the problem analysis for cloud workflow scheduling. Section 3 proposes the market-oriented hierarchical scheduling strategy. Section 4 presents the package based random generation algorithm for service-level scheduling and Sect. 5 presents the three representative metaheuristic scheduling algorithms for task-level scheduling. Section 6 demonstrates simulation experiments to verify the effectiveness of our hierarchical scheduling strategy and presents the comparison results for the performance of the three metaheuristic scheduling algorithms. Section 7 presents the related work. Finally, Sect. 8 addresses the conclusions and the future work.

2 Workflow scheduling in cloud workflow systems

In this section, we first present a general architecture of the cloud workflow system to demonstrate the role of workflow schedulers in the lifecycle of a workflow instance. Afterward, we present the problems faced by workflow scheduling strategies in cloud workflow systems.

2.1 Cloud workflow system architecture

The architecture of SwinDeW-C is depicted in Fig. 1. As discussed earlier, the general cloud architecture includes four basic layers from top to bottom: application layer (user applications), platform layer (middleware cloud services to facilitate the development/deployment of user applications), unified resource layer (abstracted/encapsulated resources by virtualisation), and fabric layer (physical hardware resources). Accordingly, the architecture of SwinDeW-C can also be mapped to the four basic layers. Here, we present the lifecycle of an abstract workflow application to illustrate the system architecture. Note that here we focus on the system architecture, the introduction on the cloud management services (e.g. brokering, pricing, accounting, and virtual machine management) and other functional components are omitted here and will be introduced in the subsequent sections.

Users can easily get access to SwinDeW-C Web portal (as demonstrated in Sect. 6) via any electronic devices such as PC, laptop, PDA, and mobile phone as long as they are connected to the Internet. Compared with SwinDeW-G which can only be accessed through a SwinDeW-G peer with pre-installed programs, the SwinDeW-C Web portal has greatly improved its usability. At workflow build-time stage, given the cloud workflow modelling tool provided by the Web portal on the application layer, workflow applications are modelled by users as cloud workflow specifications (consist of such as task definitions, process structures, and QoS constraints). After workflow specifications are created (static verification tools for such as structure errors and QoS constraints may also be provided), they will be submitted to any one of the coordinator peers on the platform layer. Here, an ordinary SwinDeW-C peer is a cloud service node which has been equipped with specific software services similar to a SwinDeW-G peer. However, while a SwinDeW-G peer is deployed on a standalone physical machine with fixed computing units and memory space, a SwinDeW-C peer is deployed on a virtual machine of which the computing power can scale dynamically according to task request. As for the SwinDeW-C coordinator peers, they

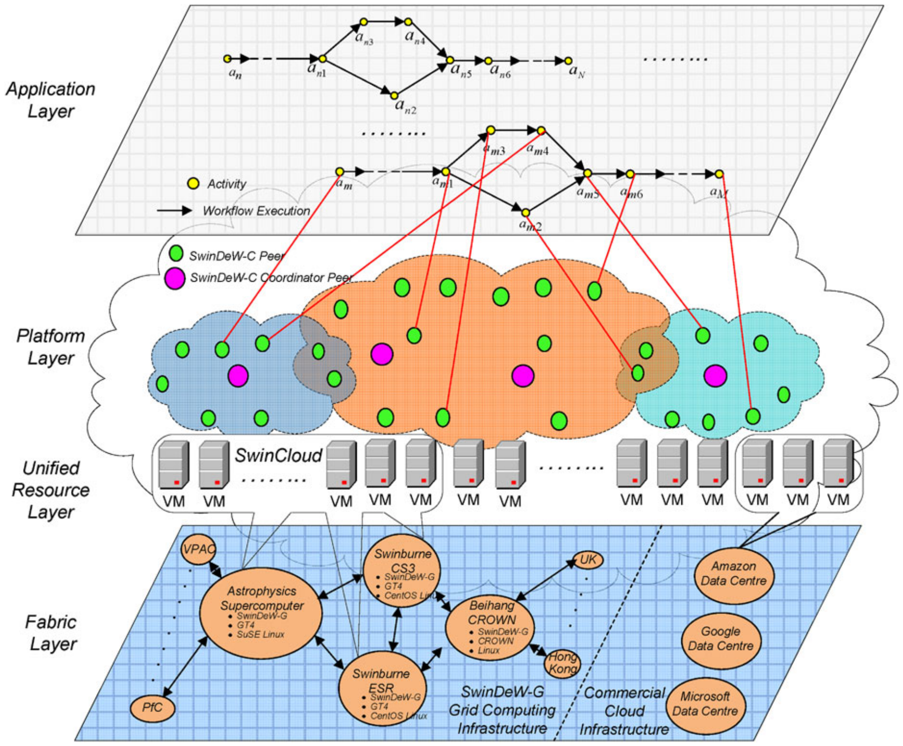


Fig. 1 Cloud workflow system architecture

are super nodes equipped with additional workflow management services and the knowledge across different clouds compared with ordinary SwinDeW-C peers. For example, a SwinDeW-C coordinator peer can act as a global scheduler which deals with service-level scheduling at the platform layer (will be discussed in Sect. 3) for mapping workflow tasks with suitable cloud services.

At the run-time instantiation stage, the cloud workflow specification can be submitted to any of the SwinDeW-C coordinator peers. Afterward, the workflow tasks will be assigned to suitable peers through peer to peer based communication between SwinDeW-C peers. Since the peer management such as peer join, peer leave, and peer search, as well as the p2p based workflow execution mechanism, is the same as in SwinDeW-G system environment. Therefore, the detailed introduction is omitted here but can be found in [45]. Before workflow execution, a coordinator peer will conduct an evaluation process on the submitted cloud workflow instances to determine whether they can be accepted or not given the specified non-functional QoS requirements under the current pricing model. It is generally assumed that functional requirements can always be satisfied given the theoretically unlimited scalability of cloud. In the case where users need to run their own special programs, they can upload them through the Web portal and these programs will be automatically deployed in the data centre by the resource manager. Here, a negotiation process between the user and the cloud workflow system may be conducted if the user submitted workflow instance is not acceptable to the workflow system due to the unacceptable offer on budgets or deadlines. The final negotiation result will be either the compromised QoS requirements or a failed submission of the cloud workflow instance. If all the task instances have been successfully allocated (i.e. acceptance messages are sent back to the coordinator peer from all the allocated peers), a cloud workflow instance may be completed with satisfaction of both functional and non-functional QoS requirements (if without exceptions). Hence, a cloud workflow instance is successfully instantiated.

Finally, at run-time execution stage, each task is executed by a SwinDeW-C peer. In cloud computing, the underlying heterogeneous resources are virtualised as unified resources (virtual machines). Each peer utilises the computing power provided by its virtual machine which can easily scale according to the request of workflow tasks. As can be seen in the unified resource layer of Fig. 1, the SwinCloud is built on the previous SwinGrid infrastructure at the fabric layer. Meanwhile, some of the virtual machines can be created with external commercial IaaS (infrastructure as service) cloud service providers such as Amazon, Google, and Microsoft. The virtual machines within the SwinCloud are managed by its local scheduler which deals with task-level scheduling at the unified resource layer (will be discussed in Sect. 3) for optimising the Task-to-VM assignment to meet QoS constraints of individual workflow tasks. Any SwinDeW-C peer (either coordinator peer or ordinary peer) can act as a local scheduler. During cloud workflow execution, workflow management tasks such as QoS management, data management, and security management are executed by the SwinDeW-C peers. Users can get access to the final results as well as the running information of their submitted workflow instances at any time through the SwinDeW-C Web portal.

2.2 Problem analysis for cloud workflow scheduling

Cloud computing adopts the *market-oriented* business model, and hence service contracts are signed between users and service providers which specify both functional and non-functional QoS constraints before the running of cloud applications. If service providers fail to fulfil the service contracts, they will have to compensate the loss claimed by users. Otherwise, the reputation of service providers will be deteriorated in the global cloud markets. Therefore, cloud service providers need to adopt a series of marketing and technical strategies to ensure the successful fulfilment of service contracts while maintain the profits of their own. Among many others, a critical issue is the cloud workflow scheduling strategy which plays an important role in the resource management of cloud workflow systems. Specifically, given the *market-oriented* business model, we identify the following two changes in cloud computing in comparison to conventional computing paradigms which may bring major challenges to cloud workflow scheduling strategies:

(1) *From best-effort based scheduling to QoS-constraint based scheduling*. Conventional computing paradigms such as cluster and grid are generally based on community models where resources are free to access and shared among community members who are also resource contributors themselves. Since monetary cost is not considered, the resource managers mainly adopts the best-effort based workflow

scheduling strategies which only attempt to minimise the makespan (overall completion time) of workflow applications [50]. On the contrary, with the *market-oriented* business model in cloud computing environments, resource managers should be able to meet the user specified QoS requirements (mainly on makespan and cost) for individual workflow instances as well as minimise the overall makespan and cost for multiple workflow instances running concurrently in cloud workflow systems, namely, it should be able to minimise the overall running cost of cloud workflow systems to gain extra profits for cloud workflow service providers. Therefore, to meet the above requirements in cloud workflow systems, QoS constraint based scheduling strategies [10, 25, 28, 49, 50] instead of best-effort based scheduling strategies need to be investigated.

(2) *From specific application oriented scheduling to general service oriented scheduling*. Conventional workflow systems are often application oriented, i.e. designed to accommodate specific type of applications where resources are managed in a centralised fashion. With the emerging of service oriented architecture (SOA) [3, 12, 14, 15], and especially the advent of three levels of services (IaaS, PaaS, and SaaS) in cloud computing [17], workflow systems have been evolved from application oriented to service oriented. In such a service oriented environment, cloud workflow scheduling faces at least two problems. The first problem is that the cloud workflow system may not be able to provide all the cloud services required for the execution of the entire workflow instance in its own data centre. Hence, as described in Sect. 2, at workflow instantiation stage, the cloud workflow resource manager needs designate a resource broker to acquire available services from other service providers (as sub-contractors) in the global cloud markets and negotiate SLAs to ensure the satisfaction of user requirements. Therefore, cloud workflow schedulers should be able to facilitate the selection of suitable services with the correct assignment of fine-grained QoS constraints and control of precedence relationships. The second problem is that cloud workflow systems need to handle intensive requests for workflow applications, in another word, cloud workflow systems are of instance intensive. As platform services, workflow systems in cloud are no longer application oriented. Hence, they should possess good scalability to handle the requests from different application domains such as data and computation intensive scientific workflow applications (e.g. weather forecasting and astrophysics) and transaction intensive business workflow applications (e.g. bank transactions and insurance claim applications). Meanwhile, within the same data centre, there are probably many other independent general non-workflow tasks (without precedence relationships but with QoS constraints) running on VMs. Therefore, they should also be considered in cloud workflow scheduling strategies so as to optimise the Task-to-VM assignment in cloud data centres.

To conclude, based on the above problem analysis, the basic requirements for cloud workflow scheduling strategies are: (1) the satisfaction of QoS requirements for individual workflow instances; (2) the minimisation of the cloud workflow system running cost; (3) the ability of assigning fine-grained QoS constraints to facilitate SLA management; (4) good scalability for optimising Task-to-VM assignment in cloud data centres which include both workflow and non-workflow tasks.

3 A market-oriented hierarchical scheduling strategy

3.1 Strategy overview

To satisfy the four basic requirements for cloud workflow scheduling strategies presented above, we propose a market-oriented hierarchical scheduling strategy. The motivation mainly comes from that the fact that the four basic requirements actually demand the scheduling at two different levels, namely service-level scheduling and task-level scheduling. Specifically, for the first and third requirements, a global scheduler which in charges of the service-level scheduling is required to allocate suitable services and assign fine-grained QoS constraints to facilitate the management of service level agreement. As for the second and the fourth requirements, a local scheduler which in charges of the task-level scheduling is required to optimise the Task-to-VM assignments in the cloud data centre. The strategy overview is presented in Table 1.

The input of our strategy mainly includes the cloud workflow specifications (consist of such as task definitions, process structures, and QoS constraints), cloud service providers (the valid service providers registered in the cloud service catalogue) and the unified resources (namely virtual machines). The output of our strategy includes both the service-level scheduling plan which handles individual workflow instances by allocating suitable services for every workflow task and making reservations through SLA; and the task-level scheduling which handles both workflow tasks of multiple workflow instances and non-workflow tasks in the local data centre by optimising the Task-to-VM assignment to minimise the overall running cost of the cloud workflow system. Note that for the purpose of description, the detailed steps described below for each scheduling stage may cover some existing functionalities of general resource management in cloud computing (namely the role of the global

Table 1 Strategy overview

Market-oriented hierarchical scheduling strategy	
Input: Cloud workflow specifications, cloud service providers, unified resources	
Output: Service-level scheduling plan and task-level scheduling plan	
Service-level scheduling	**Global scheduler:** platform layer, static scheduling, workflow instances, service providers
	Step1: search and produce the collection of suitable services based on functional requirements
	Step2: assign fine-grained QoS constraints based on the non-functional QoS requirements
	Step3: allocate suitable services and make reservations
Task-level scheduling	**Local scheduler:** unified resources layer, dynamic scheduling, workflow tasks, unified resources
	Step1: obtain the QoS constraints for each individual tasks (including both workflow and non-workflow tasks)
	Step2: optimise the Task-to-VM assignment
	Step3: implement the optimal scheduling plan

scheduler and local scheduler may overlap with the role of the resource managers) such as resource brokering and SLA management, but they are not the focus of this paper. This paper mainly proposes the hierarchical scheduling framework and investigates the corresponding algorithms for the service-level scheduling and the task-level scheduling which will be further illustrated in Sect. 4 and Sect. 5, respectively.

3.2 Service-level scheduling

The service-level scheduling is a part of the resource management on the platform layer. The service-level scheduling is a type of static scheduling which aims to allocate the suitable service for each workflow task of individual workflow instances. The functionality of service-level scheduling is usually realised by the global schedulers which have the knowledge across different clouds such as a SwinDeW-C coordinator peer.

The service-level scheduling mainly consists of the following three steps:

Step1: *search and produce the collection of suitable services based on functional requirements*. After workflow specifications are submitted to cloud workflow systems, the global scheduler will first search for the available service providers in the global cloud markets given the functional requirements specified in the cloud workflow specifications. In most cases, the required services could be found on the cloud. If not, users may need to upload their own programs to the cloud workflow systems and the cloud workflow systems will deploy the programs automatically in its local data centre. Therefore, in this step, most workflow tasks will be given with more than one candidate service providers.

Step2: *assign fine-grained QoS constraints based on the non-functional QoS requirements*. Since in the global cloud markets, there are many service providers available, the non-functional QoS requirements (such as time, cost, and trust) are the main criteria to select the suitable service providers. However, users normally only specify one global or several coarse-grained QoS constraints for the entire workflow instance or major workflow segments in cloud workflow specifications, global schedulers need to assign fine-grained QoS constraints to each workflow tasks so as to select the suitable service providers for each of them [26, 48]. Since there are many strategies available for setting overall QoS constraints and assigning fine-grained QoS constraints in workflow systems, we will not discuss them in this paper but can be referred in [20, 26, 48].

Step3: *allocate suitable services and make reservations*. As we introduced in Sect. 2, a negotiation process between the user and the cloud workflow system will be conducted in order to sign service contracts. Based on the collection of candidate services obtained in Step1 and the fine-grained QoS constraints obtained in Step2, a static scheduling strategy, as will be illustrated in Sect. 4, is required in order to generate a number of service-level scheduling plans with different task-to-service assignments. Given these service-level scheduling plans, the global scheduler can then estimate and provide the user with competitive prices (which may be determined by some marketing level strategies but normally be able to cover the system running cost) and different levels of QoS (above the level of user requirements). As for those services which are not own by the cloud workflow system, the global scheduler will activate the resource broker to negotiate SLA with the other service providers. Finally, after the selection of a specific service-level scheduling plan, the contract between the user and the cloud workflow system is signed (along with the sub-contracts between cloud workflow systems and other service providers), and reservations will be made on all the services (e.g. specific time slots for computing services and fixed sizes of memory spaces for storage services).

Based on the above three steps, a service-level scheduling plan is selected at the workflow runtime instantiation stage with the static information of candidate services. It can guarantee satisfactory QoS for the workflow instance if the service contract (along with all the sub-contracts) will be successfully fulfilled as promised. However, due to the dynamic nature of cloud computing environments, local schedulers in cloud data centres should be able to adapt to the system changes or exceptions at workflow runtime (e.g. resource upgrade and resource recruitment, or resource unavailability and resource overload). Therefore, at the workflow runtime execution stage, to ensure the successful fulfilment of service contracts, dynamic task-level scheduling in each cloud data centre is required.

3.3 Task-level scheduling

The task-level scheduling is a part of the resource management on the unified resources layer. The task-level scheduling is a type of dynamic scheduling which aims to optimise the Task-to-VM assignment so that the QoS constraints for each individual tasks will be satisfied while the overall running cost of cloud workflow systems can be minimised. The functionality of task-level scheduling is usually realised by local schedulers which are located within a specific cloud such as a SwinDeW-C ordinary peer. Note that local scheduler cannot control the cloud resources which belong to other service providers but only with the obligations of service contracts. Each service provider may have its own task-level scheduling strategies to optimise the system running cost of its own data centre. Therefore, in our strategy, the task-level scheduling only targets the tasks running in a specific data centre, i.e. the data centre which the cloud workflow system belongs to. Moreover, since our scheduling strategy can be implemented simultaneously in many data centres, it can also be employed to minimise the overall running cost for multiple data centres. However, this will be further investigated in our future work.

The task-level scheduling mainly consists of the following three steps:

Step1: *obtain the QoS constraints for each individual tasks (including both workflow and non-workflow tasks).* Given the results of the service-level scheduling, suitable services have been allocated to individual workflow tasks and time slots have been booked in advance. However, since the initial scheduling plan is generated by the service-level scheduling based on static information (e.g. the available time slots and the work load) of cloud services, the runtime service states may be significantly different due to such as the changes of service workload, the unavailability of existing resources and the recruitment of additional resources. Moreover, since the service-level scheduling plan only deals with individual workflow instances, it cannot have a global view of the Task-to-VM assignment for workflow tasks of multiple workflow instances and general non-workflow tasks in the data centre which needs be optimised to minimise the system running cost. However, since the original time slots on each service reserved by service-level scheduling will probably be re-arranged during the optimisation process, the QoS constraints for each individual task need to be obtained for validation purpose, i.e. to validate whether the generated scheduling plans can meet these QoS constraints or not. Moreover, for workflow tasks, their precedence relationships are also required to be obtained besides their QoS constraints.

Step2: *optimise the Task-to-VM assignment.* In cloud data centres, the underlying physical resources are virtualised as unified resources, namely virtual machines (VMs). VMs with specific CPU units and memory spaces can be created dynamically to suit the needs of different cloud applications. Most of the VMs are created with commodity machines which have moderate processing speed. However, due to the needs of some computation intensive tasks such as scientific workflow applications, high performance machines such as supercomputers are also available in the data centre or hired from other service providers which provide IaaS, i.e. infrastructure as a service. Therefore, VMs in a specific data centre may include resources with different processing speeds, and hence with different pricing models. Additionally, the network transfer is usually free within a data centre and the cost on data storage can be paid in a simple way (e.g. the storage cost for each task is linear to the storage time and size since the storage services are normally charged with a unanimous pricing model). Therefore, in this paper, the optimisation of the Task-to-VM assignment focuses on the reduction of the time and cost for computation services. The goal in this step is to generate an optimal (near-optimal) scheduling plan by a specific metaheuristic scheduling algorithm which can significantly reduce the overall system running time and cost for the data centre while satisfy the QoS constraints of both workflow and non-workflow tasks.

Step3: *implement the optimal scheduling plan.* After the optimal (near-optimal) scheduling plan is generated in **Step2** by a specific metaheuristic scheduling algorithm within limited time, it will then be automatically implemented by the local scheduler who is capable of controlling the underlying resources on the fabric layer to carry out the workflow execution.

Based on the above three steps, a task-level scheduling plan is implemented in the cloud data centre to carry out the workflow execution with satisfactory QoS. Meanwhile, the overall running cost of the cloud workflow system has also been minimised.

As mentioned earlier, cloud workflow systems are of instance intensive. Therefore, the service-level scheduling which deals with individual workflow instances will be conducted whenever a new workflow instance arrives and hence in a very high frequency. As for the task-level scheduling, since it deals with the Task-to-VM assignment in the entire data centre, it will be conducted much less frequently so as to allow the optimal scheduling plan to be carried out but also periodically so as to accommodate the coming of new tasks. For those tasks which have not been optimised yet, they will be executed according to their reserved time slots, namely the initial service-level scheduling plan. However, when significant environment changes such as the break down of existing resources or the recruitment of additional resources, the task-level scheduling should often be conducted immediately.

4 Service-level scheduling algorithm

Service-level scheduling is to assign suitable service to each task of individual workflow instances according to their functional and non-functional QoS requirements. As mentioned above, we will not detail the algorithms for the processes in Step1 and Step2 such as service discovery and fine-grained QoS constraints assignment but focus on the algorithm for generating service-level scheduling plans which specify the concrete Task-to-Service assignment.

Specifically, given the set of unassigned workflow tasks, the set of available cloud services (obtained in Step1 based on functional requirements), the objective for service-level scheduling algorithm is to generate candidate scheduling plans which can satisfy the following constraints: the fine-grained QoS constraints for each workflow task (obtained in Step2 based on non-functional QoS requirements), the precedence relationship between workflow tasks (defined in workflow specifications). Based on the generated candidate scheduling plans, the global scheduler needs to select the best one from them usually based on the system defined criteria such as minimum cost or minimum time. The best service-level scheduling plan will be applied to specify service contract between the user and the cloud workflow system and further make reservations on the selected services. Therefore, the service-level scheduling algorithm should be able to explore the search space of Task-to-Service in an efficient fashion.

In this paper, we present a package based random scheduling algorithm for service-level scheduling. But before that, we briefly illustrate DAG task graphs. DAGs (Directed Acyclic Graphs) based modelling is widely used in workflow area [4, 6, 22, 40]. As depicted in Fig. 2, in a DAG, each node represents a workflow task and directed links indicate the task dependencies. To facilitate cloud workflow scheduling, each task node in a DAG also associated with its QoS constraints, e.g. the constraints on the execution time and execution cost. In a DAG, each node (except the root node, i.e. the first task) will have one and only one parent node which indicates its parent task. A child task will become ready and can be executed until its parent task is completed. However, it is possible that a child task has more than one parent task in real world processes. In such a case, as depicted in Fig. 2(b), a transformation process will create multiple instances of the child task for each of its parent tasks. Additionally, for an iterative task as depicted in Fig. 2(c), a transformation process will create the same number of instances for the iterative task as its iteration times. Therefore, DAGs can visually represent common workflow structures with single parent-child relationships. This simple representation of DAGs can benefit workflow scheduling algorithms in the efficient validation of the precedence relationships between workflow tasks. An example workflow defined by a DAG is presented in Fig. 2(d).

As depicted in Fig. 3, the service-level scheduling algorithm adopts a two-dimensional representation of scheduling plans where the first dimension $sched_i$ denotes the scheduled tasks and the second dimension $alloc_i$ denotes the allocated services. The packages here consist of co-allocated tasks of the same workflow instance with correct precedence relationships defined in DAGs. For example, if T1, T2 and T3, as depicted in Fig. 2(a), can be executed by the same service (i.e. with the same

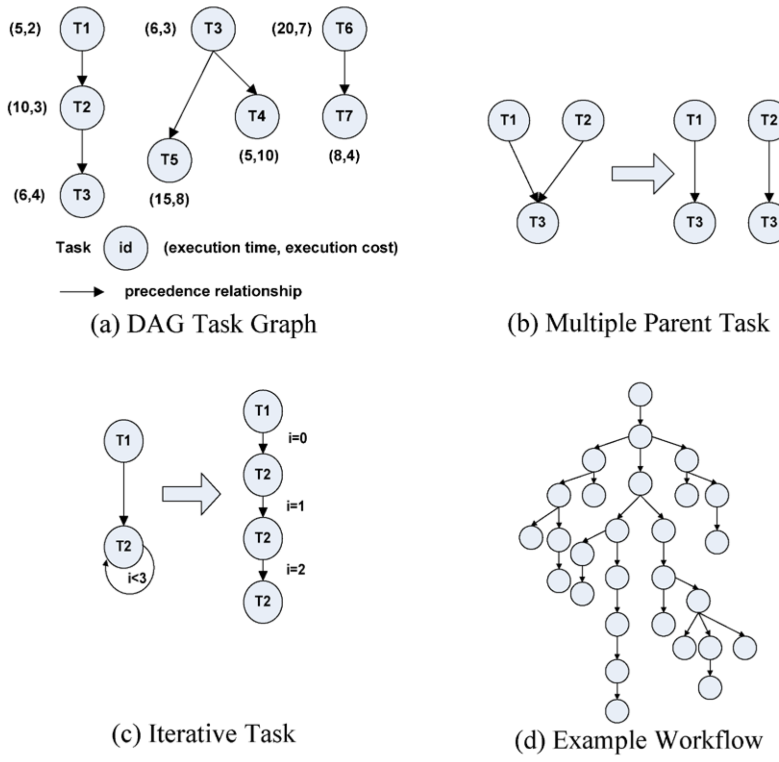
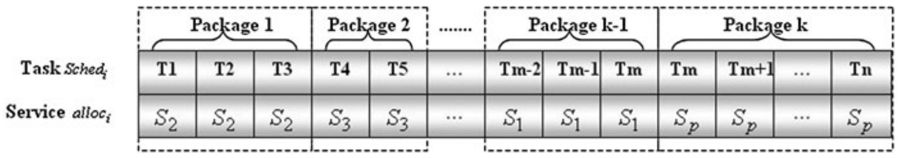


Fig. 2 DAG task graph



	Package 1			Package 2		....	Package k-1			Package k			
Task $Sched_i$	T1	T2	T3	T4	T5	...	Tm-2	Tm-1	Tm	Tm	Tm+1	...	Tn
Service $alloc_i$	S_2	S_2	S_2	S_3	S_3	...	S_1	S_1	S_1	S_p	S_p	...	S_p

Fig. 3 Package based service-level scheduling plan

functional requirements), then they can form a package where the tasks are arranged in correct precedence relationships and share the same candidate service randomly allocated to each package (e.g. Package 1 of the service-level scheduling plan shown in Fig. 3 with service S_2). Note that here without losing generality, we assume every workflow task is an atomic task, i.e. every workflow task only requires a single cloud service. If a workflow task is not an atomic task, it can be split into multiple atomic tasks according to its required capabilities. Every cloud service can have more than one capability. As for multiple instances of workflow tasks (e.g. multiple parent task and iterative task as shown in Figs. 2(b) and 2(c)), it is possible that they are located in different packages and allocated with different services (e.g. the two instances of task Tm shown in Package $k-1$ with service S_1 and Package k with service S_p in Fig. 3). In such a case, the multiple instances of the same task need to be allocated

Algorithm 1 Package based random scheduling

Input: DAGs $DAG\{Ti(Timei, Costi), Ti > Tj \mid 0 < i < j < Size(DAG)\}$
Services $Service\{Si(Capability_i, Speed_i, Price_i)\}$
Output: $\{sched_i, alloc_i\}$**: Service-Level Scheduling Plan**

```
1) for $i = 1$ to size        // size of candidate scheduling plans
2)    for $j = 1$ to $K$
// package based random scheduling
3)       for each $DAG(j)$
// function size() returns the number of tasks in each DAG,
4)          $sched_i\left(\sum_{m=1}^{j-1} size(DAG(m)) + 1 : \sum_{m=1}^{j} size(DAG(m))\right)$
5)                $= Task\left(\sum_{m=1}^{j-1} size(DAG(m)) + 1)\right) : Task\left(\sum_{m=1}^{j} size(DAG(m))\right)$;
// allocate random service to each package
6)          $alloc_i\left(\sum_{m=1}^{j-1} size(DAG(m)) + 1 : \sum_{m=1}^{j} size(DAG(m))\right)$
7)                $= random(Si|Capability_i)$;
8)       end
9)    end
// verification of QoS constraints
10) for each candidate scheduling plan
11)    if    $alloc_i(1:n).Price \le sched_i(1:n).cost$
12)          &&    $alloc_i(1:n).speed \ge 1/sched_i(1:n).time$
13)          $\{sched_i, alloc_i\}.valid = True$
14)    else
15)          $\{sched_i, alloc_i\}.valid = False$
16)    end
17) end
// return the best service-level scheduling plan
18)    $Best\{sched, alloc\} = Compare(\{sched_i, alloc_i\}.valid = True)$
19) Return $Best\{sched, alloc\}$ to global scheduler
```

with the same service, e.g. either S_1 or S_p in a random fashion or based on the service speed or price. The pseudo-code for package based random scheduling algorithm is presented in Algorithm 1.

The input of the scheduling algorithm consists of the DAGs which define the precedence relationships between workflow tasks and their associated fine-grained QoS constraints on time and cost, and the available cloud services with their capability (capabilities), execution speed, and price. The algorithm will repeat for a number of times until a fixed size of candidate scheduling plans are generated (line 1). For each package, the workflow tasks will be arranged with correct precedence relationships according their associated DAGs, and each package is randomly allocated with a service which satisfies the functional requirements (line 2 to line 9). With the generated candidate scheduling plans, the next step is to verify them with the given QoS constraints (line 10 to line 17). For each candidate scheduling plan, it is valid if and only if for all the individual Task-to-Service assignment, the price of the allocated service is not higher than the cost constraint of the task, and the execution speed of the allocated service is not lower than the execution speed required (i.e. the reciprocal of the execution time) by the task. After the verification process, the valid candidate scheduling plans will be compared to obtain the best one based on the criteria such as minimum execution time or minimum execution cost (line 18). Finally, the best service-level scheduling plan (or sometimes the top ranked several candidates) will be returned to the global scheduler to facilitate the signing of service contracts and make reservations on allocated services (line 19).

After the service-level scheduling, workflow tasks of individual cloud workflow instances are mapped to correct cloud services with satisfying functional and non-functional QoS performance. The global scheduler in the cloud workflow system will implement the selected service-level scheduling plan by distributing the corresponding segments of the scheduling plan to the local schedulers for cloud services which are responsible for the run-time Task-to-VM assignment in the local cloud data centre. In the next section, we will present the task-level scheduling in cloud workflow systems.

5 Task-level scheduling algorithm

Task-level scheduling is to optimise the Task-to-VM assignment in the cloud data centres. Since the cloud workflow system cannot directly control the cloud services which are owned by other service providers, the task-level scheduling in our strategy specifically means the optimisation of the Task-to-VM assignment in the local cloud data centre which underneath the cloud workflow system, as depicted in Fig. 1. Therefore, the major goal for task-level scheduling is to decrease the system running cost of cloud workflow systems by dynamically optimising the Task-to-VM assignment. As mentioned earlier, in a cloud data centre, there are a large number of workflow tasks (with both QoS constraints and precedence relationships) and general non-workflow tasks (with QoS constraints but no precedence relationships) running concurrently on unified resources, namely virtual machines. These virtual machines are built on commodity machines with moderate processing speed and/or supercomputers with high processing power to meet the requirements of different applications such as transaction intensive business workflows and computation intensive scientific workflows. Meanwhile, virtual machines with different processing speed will be charged with different prices. Therefore, task-level scheduling in cloud workflow systems needs to deals with the Task-to-VM assignment where the Task set includes both workflow tasks and non-workflow tasks and the VM set includes different virtual machines with different processing speed and prices. Evidently, the input of task-level scheduling algorithms, i.e. the initial Task-to-VM list in the data centre, needs to be obtained in the first place.

Here, note that given the large scalability of cloud computing, the number of tasks running in a cloud data centre is huge for instance, hundreds of thousands of tasks.

Besides, due to the complexity (NP-Complete) of QoS constraint based scheduling problem in nature [50], it is extremely difficult, if not impossible, to design a global scheduling algorithm to optimise the Task-to-VM assignment of the entire data centre within reasonable overhead, i.e. CPU time. Therefore, in our strategy, the task-level scheduling algorithms only aim to optimise the Task-to-VM assignment of a specific local part of the entire data centre with a reasonable amount of tasks, for instance, several hundreds [10]. Meanwhile, in order to reduce the overall running cost of the data centre, the task-level scheduling algorithms can be run in a parallel fashion. As such, the Task-to-VM assignment of every part of the data centre can be optimised and run with the lowest cost. However, the overall running cost of the data centre may not be optimal but it is considerably optimised. Therefore, as will also be demonstrated in our simulation experiments, it is a reasonable and practical compromise between the overhead of task-level scheduling algorithms and the optimisation on the running cost of the cloud data centre. We will leave the further study on the task-level scheduling algorithms for the global Task-to-VM assignment of the entire data centre as our future work. In this paper, we focus on the task-level scheduling for a specific part of the entire data centre. Therefore, how to define the initial Task-Resource list as the input for the task-level scheduling algorithms is important but remains a challenging issue. Specifically, there are at least two basic requirements for the initial Task-to-VM list: (1) Reasonable partition for the global Task-to-VM list of the entire data centre; (2) Reasonable size of the Task-to-VM list. The first requirement is to ensure that the local Task-to-VM lists, as the input for each parallel task-level scheduling algorithms, are reasonably portioned, i.e. the global Task-to-VM list is fully covered by the collection of these local Task-to-VM lists, meanwhile, these local Task-to-VM lists are not covered with each other. As for the second requirement, as a control over the overhead of the task-level scheduling algorithms, the size of the local Task-to-VM list should be reasonable, e.g. the number of tasks is below 300 and the number of virtual machines is below 20 as in our simulation experiments presented in Sect. 6.

To address such an issue, we present the integrated Task-to-VM list to define the local Task-to-VM list as the input for each parallel task-level scheduling algorithm. As depicted in Fig. 4, the integrated Task-to-VM list is an integrated collection of virtual machines and he integrated DAG task graph which includes workflow tasks defined in DAG task graphs and their co-allocated non-workflow tasks. Here, co-allocated tasks are those which have been allocated to the same virtual machines. For example, as depicted in Fig. 4, the local Task-to-VM list contains four different virtual machines VM_1 to VM_4. Each virtual machine maintains a local task-list, i.e. the job queue inputted by the local scheduler of the data centre. To start the task-level scheduling, the local scheduler will acquire the current task-list of VM_1 to VM_4 and can automatically combine them into an integrated DAG task graph which consists of all the tasks, for instance, a total of n tasks, by assigning a pseudo start task T_{Start} and pseudo end task T_{End}. Hence, one integrated Task-to-VM list $L\{(T_i, VM_j)|i = p+1, \ldots, p+n, j = 1, 2, 3, 4\}$ is built which is a part of the global Task-to-VM list and ready to be optimised by the task-level scheduling algorithms. Here, tasks are defined with $\{T_i, Cost_i, Time_i, DAG\{T_i > T_{i+1}\}\}$ (the QoS constraints on cost and time, and the precedence relationship with other tasks for workflow tasks), and virtual machines are defined with $\{VM_j, Price_j, Speed_j\}$ (the

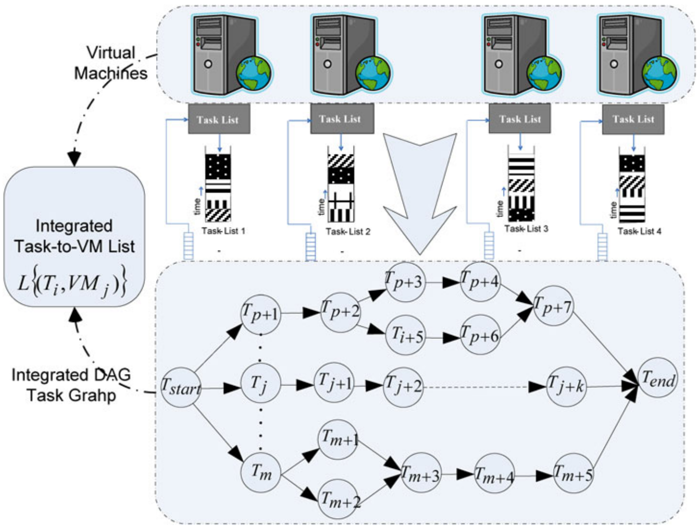


Fig. 4 The local integrated Task-to-VM list

price and the processing speed). Similarly, the same process can be conducted in a parallel fashion on all the local groups of virtual machines in the data centre. Therefore, a partition of the global Task-to-VM list is made as a result of all the integrated Task-to-VM lists. Hence, as mentioned above, the overall running cost of the data centre will be reduced since the task-level scheduling algorithm will optimise the Task-to-VM assignment in each group of virtual machines. For each group, the maximum number of virtual machines is defined by the cloud workflow system given the average length of the local task-list of each virtual machine. Empirically, as shown in our simulation experiments, the empirical ratio between tasks and virtual machines is normally below 20:1, e.g. the maximum number of tasks in an integrated Task-to-VM list is under 300 and the maximum number of virtual machines is under 20. Note that the ratio between tasks and virtual machines probably varies from data centre to data centre and are subjected to changes due to different capabilities of virtual machines.

5.1 Genetic algorithm

GA is a search technique often employed to find the exact or approximate solutions to optimisation and search problems [19, 49]. GA is a specific class of evolutionary algorithms inspired by evolutionary biology. In GA, every solution is represented with a string, also known as a chromosome, which follows the semantics defined by the encoding method. After encoding, the candidate solutions, i.e. the initial population, need to be generated as the basic search space. Within each generation, three basic GA operations, i.e. selection, crossover, and mutation, are conducted to imitate the process of evolution in nature. Finally, after the stopping condition is met, the chromosome with the best fitness value is returned, representing the best solution found in the search space. That ends the whole GA process. In recent years, GA has been adopted to address large complex scheduling problems and proved to be effective

Algorithm 2 GA based task-level scheduling

```
Input: Integrated Task-to-VM list $L\{(T_i, VM_j)|i = 1, 2, .., n; j = i, 2, .., m\}$;
       Tasks $\{T_i, Cost_i, Time_i, DAG\{T_i > T_{i+1}\}\}$;
       Virtual Machines $\{VM_j, Price_j, Speed_j\}$.
Output: Optimised Task-Level Scheduling Plan
//two dimensional encoding
1) ENCODING($L$);
// package based random generation of initial population
2) Create initial *Population* of fixed size;
// **Optimising the overall makespan and cost**
3) While (stopping condition is not met)
  {
   //selecting *Parents* from *Population*
4)    SELECTION(*Population*);
5)    CROSSOVER(*Parents*)→*Children*;
   //change resource peer of a random cell
6)    MUTATION(*Children*);
   //validate with DAG, retain or discard
7)    VALIDATION(*Children*);
   //replace children of the worst fitness with the best one
8)    REPLACE(*WorstChild*, *BestChild*);
   //check with the QoS constraints, retain or discard
9)    CHECK(*Children*);
  }
// **Selecting the *BestSolution* according to user preferences**
10) *BestSolution* = COMPAKE(*SolutionSet*, *UserPref*);
// decoding and deploy
11) DECODING(*BestSolution*);
12) Deploy($L$).
```

in many distributed and dynamic resource environments, such as parallel processor systems and grid workflow systems [30, 39].

As depicted in Algorithm 2, the first searching phase of GA based task-level scheduling is to optimise the overall makespan and cost for the integrated task-resources list through GA (Line 1 to Line 9). The GA algorithm starts from encoding (Line 1). Here, a two-dimension encoding method is adopted. The first dimension represents the scheduled acts and the second represents the resource allocated to the corresponding act in the first dimension. GA starts with the generation of a fixed size initial population (Line 2). Because the quality of initial population is critical for the final outcome, package based random algorithm is applied to generate the initial population. The algorithms for the encoding and package based random generation are the same as described in Sect. 4 (Fig. 3 and Algorithm 1, respectively), and hence omitted here. After that, the algorithm starts searching for the best solution iteratively until the stopping condition, e.g. the maximum generation, is met (Line 3 to Line 9). Three fundamental operations of GA including selection, crossover, and mutation take actions in sequence. Stochastic universal sampling (SUS) [19] for selecting potentially useful solutions as parents for recombination is used in selection operation (Line 4). SUS uses a single random value to sample all of the solutions by choosing them at evenly spaced intervals. The solution candidates generated during initialisation phase are all legal (i.e. they all satisfy the precedence relationship as defined in DAGs); however, conventional crossover will probably make some individuals illegal. To keep the diversity of the population, single point crossover strategy is employed. When two parents are both legal, single point crossover ensures their children are legal. So before the mutation, the whole population is valid [32]. The third genetic operation is mutation (Line 5) where the allocated resource is mutated, i.e. substituted for another resource, at a randomly selected cell of a chromosome. The mutation rate is normally set to a small probability value such as 10% since mutation can easily destroy the correct precedence relationship and result in invalid solutions. The major affect of mutation is that it can introduce diversity into the population to help it jump out of local optimal traps. However, it can make some individuals invalid. These invalid individuals should be eliminated through validation and replace (Line 7, Line 8). Since cloud workflow system is a market-oriented system, the candidate scheduling solution is expected to satisfy the QoS constraints according to the service contract. The last operation of the first phase is check which verifies whether the candidate individual should be retained or discarded.

During the second searching phase, the *SolutionSet* is compared with the user preference, i.e. *UerPref*, and the best scheduling plan L is deployed (Line 10 to Line 12). Both makespan and costs are taken into account in *UerPref* which defines the specific preference of users toward the two factors. For example, the *UerPref* can be the minimum makespan, the minimum cost or a balance ration between makespan and cost. The best scheduling plan (*BestSolution*) is selected from the satisfied population (Line 10). The *BestSolution* is defined as the best solution according to *UerPref* among all the valid solutions. Since the *BestSolution* is represented in a two-dimensional vector, it should be decoded back to L as an integrated Task-to-VM list (Line 11). The last step of the whole algorithm is to deploy the L (Line 12).

5.2 Ant colony optimisation

In recent years, Ant Colony Optimisation (ACO), a type of optimisation algorithm inspired by the foraging behaviour of real ants in the wild, has been adopted to address large complex scheduling problems and proved to be quite effective in many distributed and dynamic resource environments, such as parallel processor systems and grid workflow systems [5, 10].

As shown in Algorithm 3, the first searching stage of ACO based task-level scheduling is to optimise the overall execution time and cost for the integrated task-resources list through ACO (Line 1 to Line 10). The ACO algorithm starts from initialisation of pheromone and all parameters (Line 1). In [11], two types of pheromone, i.e. $d\tau_{ij}$ and $c\tau_{ij}$, are defined. Here, $d\tau_{ij}$ denotes the desirability of mapping task a_i to resource R_j from the perspective of execution time while $c\tau_{ij}$ denotes the desirability from the perspective of execution cost. Afterward, the ACO based searching process iterates until the stopping condition, i.e. the maximum iteration times, is satisfied. During each iteration, a group of ants needs to be initialised first (Line 3 to

Algorithm 3 ACO based task-level scheduling

```
Input: Integrated Task-to-VM list $L\{(T_i, VM_j)|i = 1, 2, .., n; j = 1, 2, .., m\}$;
       Tasks $\{T_i, Cost_i, Time_i, DAG\{T_i > T_{i+1}\}\}$;
       Virtual Machines $\{VM_j, Price_j, Speed_j\}$.
Output: Optimised Task-Level Scheduling Plan
//Optimising the overall makespan and cost
//Initialisation of pheromone and other parameters for AGO algorithm
// Package based random generation
1) INITIALISATION(ACO);
2) While (stopping condition is not met)
   {
// initialise each ant
       for each ant
       {
// select heuristics from duration-greedy, cost-greedy and overall-greedy
3)         SELECTION(Heuristics);
// build tackling sequence TS based on the input DAG task graphs
4)         BUILDING(TS, DAGs);
       }
// construct solutions
5)   While (not end of TS)
     {
// choose the resource for the next activity based on its earliest start time, earliest end time, and the bias of $B_{ij}$ (mapping resource $R_j$ to activity $a_i$)
6)         $CHOOSE(a_i, a_i.est, a_i.eet, R\{R_j\}, B_{ij})$;
// update the est and eet for all the subsequent activities;
7)         UPDATE(EST,EET);
// update local pheromone for both duration and cost
8)         $LOCALUPDATE(d\tau_{ij}, d\tau_{ij})$;
     }
// update the global pheromone based on the makespan and cost of the best-so-far solution
9)         $GLOBALUPDATE(d\tau_{ij}, d\tau_{ij}, makespan^{bs}, cost^{bs})$;
// return the best-so-far solution and record into the SolutionSet
10)        Return(Solution, SolutionSet)
   }
// Selecting the BestSolution according to user preferences
11) BestSolution = COMPARE(SolutionSet, UserPref);
// deploy the task-level scheduling plan
12) Deploy(L).
```

Line 4). Each ant starts with selecting one of the heuristics from duration-greedy, cost-greedy or overall-greedy which has specific preference on searching (Line 3). Then the tackling sequence which arranges the order of tasks is built based on the input DAG task graph (Line 4). During the solution construction process (Line 5 to Line 8), each activity is allocated to a specific resource according to its bias B_{ij} which is based on the value of pheromones and the heuristic information (Line 6). Meanwhile, after a specific choice of resources, the earliest start time *est* of the current activity is compared with the earliest end time *eet* of its predecessors to determine whether the current schedule can satisfy the precedence relationships defined in the DAG task graph. After a successful resource allocation, the *est* and *eet* for its subsequent activities are updated (Line 7). Here, a local updating process is conducted to decrease the local pheromone of $d\tau_{ij}$ and $c\tau_{ij}$ so that the following ant can have a higher probability of choosing other resources (Line 8). Evidently, the purpose of local updating is to enhance the diversity of the ACO algorithm. By contrast, after all ants have built their individual solutions, a global updating process is conducted to increase the pheromone along the path for the best-so-far solution so that the subsequent ants have higher probability to choose the same scheduling plan (Line 9). Therefore, the purpose of global updating is to reinforce the best-so-far solution in order to speed up the convergence of the ACO algorithm. Finally, at the end of iteration, the best-so-far solution is returned and added into the *SolutionSet* which serves as the input for the second searching stage (Line 10).

In the second searching stage, the *BestSolution* is retrieved from the *SolutionSet* (Line 11 to Line 12). The process is the same as the one described in GA. The *BestSolution* is selected according to the user preference *UerPref* (Line 11) and then the corresponding integrated Task-to-VM list is deployed.

5.3 Particle swarm optimisation

Particle swarm concept is predominately designed to find solutions for continuous optimisation problems without prior information [51, 52]. To solve the workflow scheduling problem, a discrete version of PSO (DPSO) is presented in this paper as a candidate evolutionary algorithm [9]. Something like conventional PSO, the key issue of DPSO is to define the position and velocity of particle as well as to define their operation rules and the equation of motion according to the features of discrete variables. Suppose that the workflow has m tasks and n resources. The position of particle i is presented as $X_i = (x_{i1}, x_{i2}, \dots, x_{ij}, \dots, x_{im})$, $1 \le j \le m, 1 \le x_{ij} \le n$, a particle i is also associated with a velocity V_i along each dimension $V_i = (v_{i1}, v_{i2}, \dots, v_{ij}, \dots, v_{im})$, $1 \le j \le m, v_{ij} \in \{-1, 0, 1\}$. Each particle is assumed to have a local memory that keeps its previous best position *pbest*. For each particle, *pbest* and the position vector of the best performing particle in the local neighbourhood, *gbest*, are combined to adjust the velocity along each dimension, and the adjusted velocity is then used to adjust the position of the particle. Here, V_i is a selective strategy, when v_{ij} equals to -1, the corresponding bit is selected from *gbest*; when v_{ij} equals to 0, the corresponding bit is selected from X_i; when v_{ij} equals to 1, the corresponding bit is selected from *pbest*. For the sake of clarity, variables and the rules of DPSO for solving workflow scheduling can be depicted in formulas.

(1) Multiply the subtraction of position

$$V_p = c_1(X_{pbest} - X), \tag{1}$$

$$V_p = \begin{cases} 0 & \text{select the corresponding bit value from } X, \\ 1 & \text{select the corresponding bit value from } pbest. \end{cases}$$

Here, we define the threshold $\delta \in [0, 1]$, generate a random number r for each workflow, compare r and δ. When $r \geq \delta$, assign 0 to V_p, otherwise, assign 1 to V_p.

$$V_g = c_2(X_{gbest} - X), \tag{2}$$

$$V_g = \begin{cases} 0 & \text{select the corresponding bit value from } X, \\ -1 & \text{select the corresponding bit value from } gbest. \end{cases}$$

Here, we also generate a random number r for each workflow, compare r and δ. When $r \geq \delta$, assign 0 to V_g, otherwise, assign -1 to V_g.

(2) Motion equations of particle

$$V = c_1(X_{pbest} - X) + c_2(X_{gbest} - X), \tag{3}$$

$$X = X + V. \tag{4}$$

As described in Algorithm 4, the first searching phase here is also to optimise the overall execution time and cost for the integrated task-resources list to be scheduled through DPSO (Line 1 to Line 18). The particles involved in this algorithm are encoded in two dimensions. One dimension indicates the acts from the workflow to be scheduled; the other represents the resources to fulfil the relevant acts. This relationship is maintained by the indices of vectors. The DPSO algorithm starts from initialisation of swarm. To effectively control the trade-off of cup time and solution quality, maximum number of function calls(MAX$_{\text{NFC}}$), minimum number of function calls(MIN$_{\text{NFC}}$), makespan optimisation rate compared to the last iteration(MOR) and cost optimisation rate compared to the last iteration(COR) (Line 2 to Line 5) is defined. Generate a valid solution for each particle, because the initial position of each particle is the personal best position, the generated solution can be assigned to the *pbest* of every particle directly (Line 6 to Line 9). In DPSO, the particle not only learns from its own experience, but also learns from the counterpart in the swarm. So, the particle with the biggest fitness value is assigned to the swarm global best, that is *gbest* (Line10). After this, the process is entering search iterations (Line 12 to Line 21). When the maximum number of function calls is met or the other three conditions is satisfied, the iteration will stop. The three conditions are to ensure iteration time is no less than minimum number of function calls and to ensure the optimisation rate on makespan and cost is no less than 2% during iteration. Each sub-workflow gets a random number to decide whether it will learn from its own experience or not, the same way for it to learn from the global best candidate solution. The velocity, also a selective strategy, for the particle to move is a vector with the value $-1, 0, 1$. When the component is -1, the resource allocated to the act according the corresponding

Algorithm 4 PSO based task-level scheduling

```
Input: Integrated Task-to-VM list $L\{(T_i, VM_j)|i = 1, 2, .., n; j = 1, 2, .., m\}$
       Tasks $\{T_i, Cost_i, Time_i, DAG\{T_i > T_{i+1}\}\}$;
       Virtual machines $\{VM_j, Price_j, Speed_j\}$
Output: Optimised task-level scheduling plan
//Optimising the overall makespan and cost
// swarm initialisation
1) $ps$ = population size;
2) $\mathrm{MAX_{NFC}}$ = maximum number of function calls (NFC);
3) $\mathrm{MIN_{NFC}}$ = minimum number of function calls (NFC);
4) MOR = makespan optimisation ratio compared to the last iteration
5) COR = cost optimisation ratio compared to the last iteration
6) For $j = 1$ to $ps$
7) GENERATE a valid solution $X$;
8) ASSIGN $X$ to $pbest$;
9) FIND $gbest$;
10) GENERATE learning probability $\delta$ and $\chi$;
11) While ((NFC < $\mathrm{MAX_{NFC}}$) and (NFC < $\mathrm{MIN_{NFC}}$ or MOR < 0.02 or
    COR < 0.02))
{
12) For $j = 1$ to $ps$
   /*PSO evolution steps*/
   /* Velocity updating*/
13)    GENERATE a random number $r$ for each sub workflow;
14)    CALCULATE $V_p$ using $pbest$ and $X$;
15)    CALCULATE $V_g$ using $gbest$ and $X$;
16)    CALCULATE $V$ using $V_p$ and $V_g$;
/*Position updating*/
17)    CALCULATE new position $X$ using $X$ and $V$;
/* pbest and gbest updating*/
18)    UPDATE($pbest$);
19)    UPDATE($gbest$);
// return both pbest and gbest and record them into the SolutionSet
20)    Return($Solution$, $SolutionSet$);
}
// Selecting the BestSolution according to user preferences
21) $BestSolution$ = COMPARE($SolutionSet$, $UserPref$);
// deploy the task-level scheduling plan
22) Deploy($L$).
```

component in the global best, when it is 0, the position has no change, when it is 1, the resource allocated as the component in the personal best (Line 14 to Line 17). Once these operations is performed, the particle is in the new position, that is the new solution is produced. Before starting the next iteration, the global best and the personal best of each particle are expected to be updated (Line 19 to Line 20). Finally, at the end of iteration, the best-so-far solution is returned and added into the *SolutionSet* which serves as the input for the second searching stage (Line 21).

In the second searching stage, the *BestSolution* is retrieved from the *SolutionSet* according to the user preference *UerPref* (Line 21 to Line 22). The process is the same as the above two described in GA and ACO. Finally, the corresponding integrated Task-to-VM list L is deployed.

6 Evaluation

6.1 Simulation environment

SwinDeW-C (Swinburne Decentralised Workflow for Cloud) [46] is developed based on SwinDeW [44] and SwinDeW-G [45]. It is currently running at Swinburne University of Technology as a virtualised environment which is physically composed of 10 servers and 10 high-end PCs. To simulate the cloud computing environment, we set up VMware [41] software on the physical servers and create virtual clusters as data centres. Figure 2 shows our simulation environment.

As depicted in Fig. 5(a), every data centre created is composed of 8 virtual computing nodes with storages, and we deploy an independent Hadoop [18] file system on each data centre. SwinDeW-C runs on these virtual data centres that can send and retrieve data to and from each other. Through a user interface at the applications layer, which is a Web based portal, we can deploy workflows and upload application data. SwinDeW-C is designed for large scale workflow applications in the cloud computing environments. In Fig. 5(b), we illustrate the key system components of SwinDeW-C.

User Interface Module: The cloud computing platform is built on the Internet and a Web browser is normally the only software needed at the client side. This interface is a Web portal by which users can visit the system and deploy the applications. The Uploading Component is for users to upload application data and workflows, and the Monitoring Component is for users, as well as system administrators to monitor workflow execution.

Resource Management Module: Resource management module is the major component in the cloud workflow system. The workflow scheduling module plays an important role in the support of cloud workflow execution and the management of cloud resources. As proposed in this paper, SwinDeW-C employs the hierarchical scheduling strategy which consists of service-level scheduling and task-level scheduling. As for its other major functionalities such as resource brokering, pricing, auditing, and SLA management they can directly inherit their counterparts in the general cloud computing environment.

QoS Management Module: In market-oriented cloud computing environments, service providers need to deliver satisfactory QoS in order to fulfil service contracts and make profits. Otherwise, they cannot sustain in the global cloud markets. Generally speaking, QoS management includes three main components, namely QoS setting, QoS monitoring and exception handling. Taking temporal QoS management, one of the major dimensions of workflow QoS [48] as an example, it consists of temporal constraint setting which specifies temporal constraints in cloud workflow

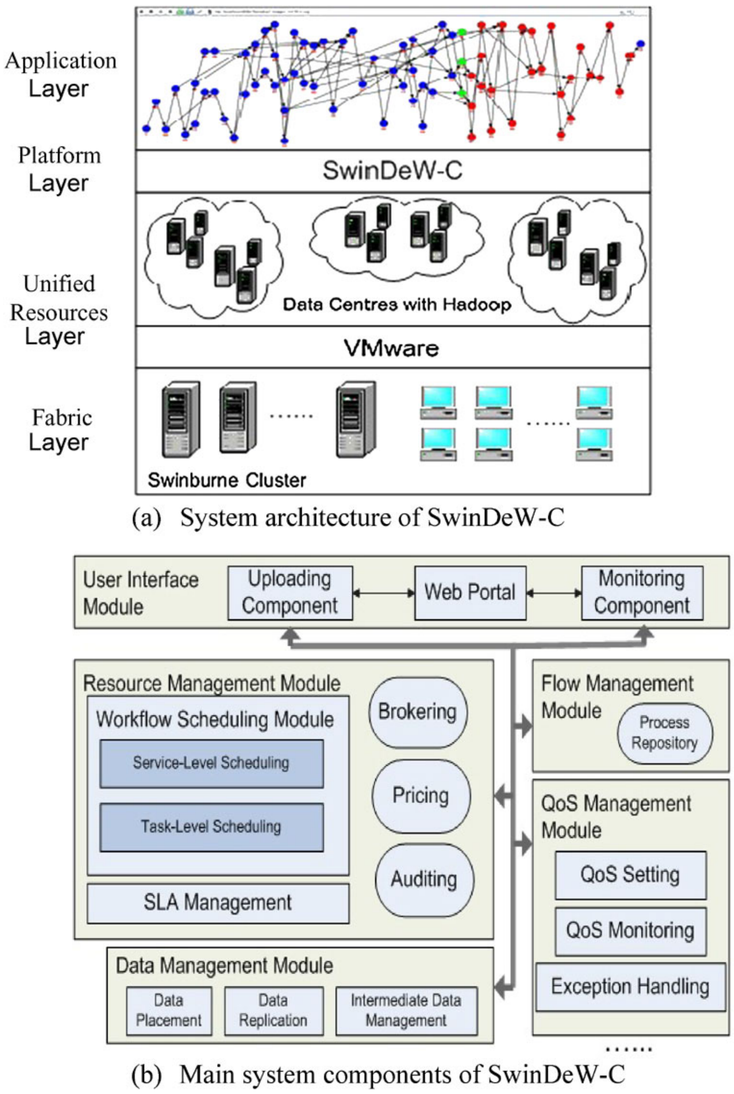


Fig. 5 Simulation environment of SwinDeW-C

specifications at build time [26]; temporal checkpoint selection and temporal verification which dynamically monitors the temporal consistency state along workflow execution [7, 8], and temporal adjustment which handles detected temporal violations to ensure satisfactory temporal QoS [7, 35].

Other Modules: The Data Management Module includes the strategies for data placement, the data catalogue for service registry and lookup, and other components for cloud data management. The Flow Management Module has a Process Repository that stores all the workflow instances running in the system.

As part of the cloud resource management module, our hierarchical scheduling strategy is currently being implemented in SwinDeW-C. Specifically, the package based random algorithm is implemented as a functionality of the global scheduler in SwinDeW-C to facilitate service-level scheduling, i.e. allocating suitable services and making reservations for individual tasks; the ACO based task-level scheduling algorithm is implemented as a functionality of the local scheduler in local data centres to facilitate task-level scheduling, i.e. optimising the Task-to-VM assignment to reduce cloud workflow system running cost (ACO is the best candidate among the three representative algorithms including GA, ACO, and PSO, as will be illustrated with the experimental results in Sect. 6.3).

6.2 Experiment settings

6.2.1 Parameter settings for workflows

The workflow processes are randomly generated as DAG task graphs as described in Sect. 4 where each workflow segment is with a random size of 3 to 5 activities. The mean duration of each task is randomly selected from 30 to 3000 basic time units and its standard deviation is defined as 33% of its mean (a large standard deviation for valid normal distribution models where the samples are all positive numbers according to the "3σ" rule [23]) to represent the highly dynamic performance of underlying resources. Each resource contains three attributes including resource ID, the execution speed, and the execution cost. Here, the execution speed is defined as a random integer from 1 to 5 where the execution time is equal to the mean duration divided by the execution speed. In each of the ten experiment scenarios, half of the virtual machines are with speed of 1. To simplify the setting, the price of each resource in our experiment is defined as the execution speed plus a random number ranging from 1 to 3. For example, if a task is allocated to a resource with the execution speed of 2, then 2 basic units plus an additional random 1 to 3 basic units, e.g. 4 basic cost units, will be charged for every basic time unit consumed on such a resource (namely the price of the resource is 4). Three attributes is defined for the integrated task-resource list, that is, the number of total tasks, the number of workflow segments and the number of resources. Specifically, the number of total tasks ranges from 50 to 300 including both workflow and non-workflow activities. The number of workflow segments increase accordingly from 5 to 50. The number of resources is constrained in the range of 3 to 20 since high performance resources in scientific workflow systems usually maintain long job queues. QoS constraints including time constraint and cost constraint for each task are defined where time constraint is defined as the mean duration plus $1.28 \times \sqrt{variance}$ and cost constraint is defined as the triple of the corresponding time constraint. The makespan of a workflow is defined as the latest finished time on all the virtual machines and the total cost of a workflow is defined as the sum of task durations multiply the prices of their allocated virtual machines. As for the three basic performance measurements, the optimisation rate on makespan equals to the mean makespan subtract the minimum makespan, then divides by the mean makespan; the optimisation rate on cost equals to the mean cost subtract the minimum cost, then divided by the mean cost; the CPU time used

Table 2 Experiment scenarios

Scenario No.	Number of total tasks	Number of workflow segments	Number of virtual machines
1	50	6	3
2	80	10	5
3	120	15	6
4	150	20	8
5	180	25	10
6	200	28	12
7	220	30	15
8	250	35	16
9	280	40	18
10	300	50	20

is defined as the average execution time of each algorithm running on a standard SwinDeW-C node. The detail of each experiment scenario is shown in Table 2. Note that in these settings, the number of workflow tasks and the number of non-workflow tasks are kept on the same level to simulate the application scenarios in a general cloud data centre.

6.2.2 Parameter settings for genetic algorithm

In GA, 50 new individuals are created during each iteration. The crossover rate is set to 0.7 and the mutation rate is 0.1. Single cross over method is applied to recombine the chromosomes. When an invalid chromosomes is produced by the operation of mutation, the best chromosomes in the population will replace it. The fitness value of a solution is defined as the reciprocal of the sum consisting of the makespan and 5% of the cost (the purpose of decreasing the cost here is to balance the weight of makespan and cost since their original amount is not at the same level 5% is an empirical value here according to our experiment setting and it is subject to changes in particular system environments). To make a trade-off between effectiveness and efficiency, we design a compound stopping condition with four parameters: the minimum iteration times, the maximum iteration times, the minimum increase of optimisation rate on time (the increase of optimisation rate on time: the minimum makespan of last iteration subtracts the minimum makespan of the current iteration and divided by the one of the last iteration), the minimum increase of optimisation rate on cost (similar to that of makespan). Specifically, the evolutionary process iterates at least a minimum of 100 times. After 100 times iterations, the iteration will stop on condition that the maximum iteration times are met; or the increase of optimisation rate on time is less than 0.02; or the increase of optimisation rate on cost is less than 0.02.

6.2.3 Parameter settings for ant colony optimisation

In ACO, 50 new ants are created in each iteration. Since we focus on both the reduction of makespan and the reduction of cost, half of them are created as duration-greed and another half as cost-greedy. The maximum iteration times are set as 1000 and the minimum iteration times are 100. The weights of pheromone and heuristic information are set to be 1 and 2. The probability of selecting the implementation with the largest value of B_{ij} is 0.8. Local pheromone updating rate is 0.1 and the global pheromone updating rate is also 0.1. For fairness, the fitness value and the stopping condition are the same as defined in GA.

6.2.4 Parameter settings for particle swarm optimisation

In PSO, 50 new particles are created in iteration. PSO employs the package-based random generating method same as in GA to produce the initial valid positions. Two learning probabilities are defined. The probability learning from its *pbest* for each particle is set to 0.4 and the probability learning from its *gbest* for each particle is set to 0.5. To keep the swarm owning diversity, the particle learns more from its own experience than from the global best particle. The parameters c_1 and c_2 in Formula (3) are both set as 2.0. For fairness, the fitness value and the stopping condition are the same as defined in GA.

6.3 Experimental results

In the following sections, the experimental results on the three basic measurements including the optimisation rate on makespan, the optimisation rate on cost and the CPU time are presented. As explained in the parameter setting for each algorithm above, the workflow makespan and cost are given equal weights in the optimisation process. Therefore, the best solution can be selected either according to makespan or cost. In our experiment, the best solution is selected as the one with the minimum makespan and its corresponding cost is regarded as the best cost (note that the cost of the best solution is normally not the minimum cost found in the searching process). For each measurement, an overall view for the ten experiment scenarios is first presented. Afterward, as an example, 10 randomly selected independent test cases in scenario 6 where the workflow is composed of 200 tasks, 28 workflow segments, and 12 virtual machines, are presented as the detailed view. Here, since GA is the most popular metaheuristic used in workflow scheduling, the average makespan and cost by GA is adopted as the benchmark for comparison purpose.

6.3.1 Results for makespan optimisation

Figure 6 shows the experimental results on the optimisation rate of makespan. Figure 6(a) presents the optimisation rate on overall makespan in 10 different scenarios; Fig. 6(b), 6(c), 6(d) show the best and average makespan of scenario 6 by GA, ACO, and PSO, respectively.

From Fig. 6(a), it can be seen that when the workflow size is small, the optimisation rate on makespan by GA, ACO, and PSO are similar but very low. For example, in the first scenario with 50 tasks, the makespan optimisation rate of ACO is about 5%, PSO is about 5.6% and GA is about 5.8%. When the workflow size increases, the

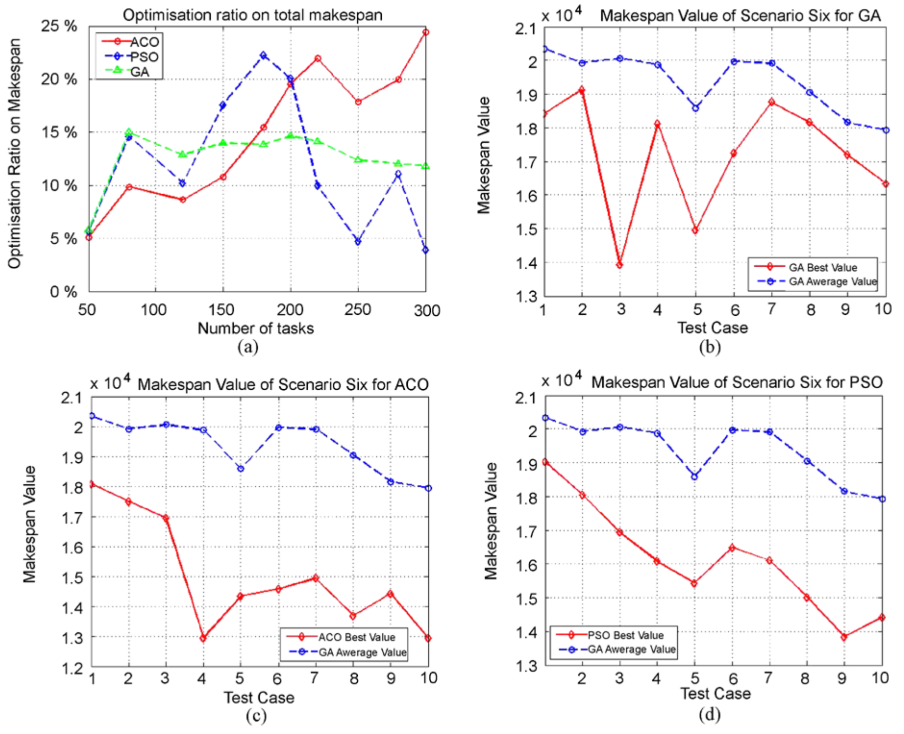


Fig. 6 Makespan optimisation of GA, ACO and PSO

performance of GA remains relatively stable but is getting worse; the performance of PSO is on the rise at first and then deteriorates dramatically. On the contrary, ACO has a potential ability to search good solution in workflow scheduling when the number of the activities becomes large. For example, in the last scenario with 300 tasks, the optimisation rate on makespan of ACO is 24.4%, GA is 11.8%, and PSO is only 3.88%. It is interesting to see that PSO achieved better performance than the others when the workflow size is ranging from 150 to 200. For example, the makespan optimisation rate of PSO is 22.2% when the number of tasks is 180 while GA is 13.8% and ACO is 15.4%.

In Fig. 6(b), 6(c), and 6(d), 10 independent test cases are shown for scenario 6. The vertical distance between the curve of best value and the curve of average value indicates the reduced makespan. A clear phenomenon is that though in the same scenario, the best value found by each algorithm in different test cases is very dynamic. However, it can be seen that the curve for the best value and the curve for the average value have the similar trend but in a coarse-grained sense. Upon our observation, the quality of the initial solutions mainly accounts for such a phenomenon besides the random nature of these algorithms. Therefore, effective methods for generating high quality initial solutions such as the package based random generation methods employed in this paper and others can be further investigated

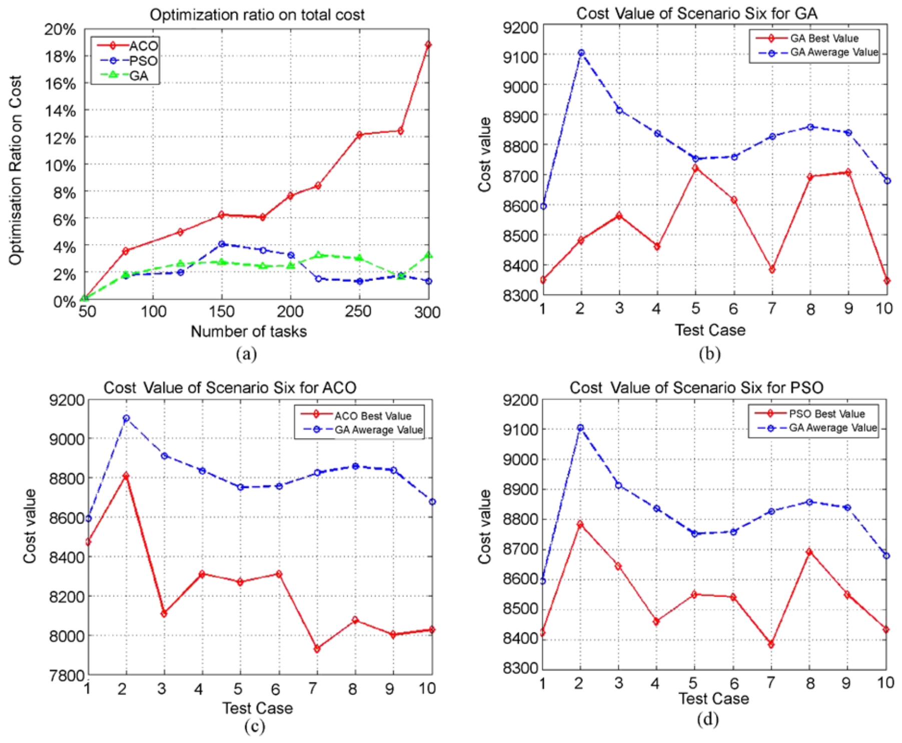


Fig. 7 Cost optimisation of GA, ACO and PSO

to improve the performance of these scheduling algorithms as well as their stability.

6.3.2 *Results for cost optimisation*

Figure 7 shows the experimental results on the optimisation rate of cost. Figure 7(a) presents the optimisation rate on cost in 10 different scenarios; Fig. 7(b), 7(c), 7(d) show the best and average cost of scenario 6 by GA, ACO, and PSO, respectively.

As can be seen in Fig. 7(a), ACO has an overall better performance than others and it shows an increasing trend. GA and PSO have similar and relatively stable performance, but much lower than that of ACO, especially when the workflow size is becoming larger. As for the detailed results in scenario 6, the best cost value by ACO is ranging from 7930.28 to 8810.87 with a mean value 8309.73; the best cost value by GA is ranging from 8346.70 to 8719.63 with a mean value 8613.42; the best cost value by PSO is ranging from 8389.50 to 8783.00 with a mean value 8541.00. This implies that ACO explores a larger searching space than the other two and it is more effective in constructing solutions with smaller cost. Meanwhile, similar to the optimisation on makespan, the best cost found by each algorithm in the ten test cases is very different but normally behaves similarly to their average cost.

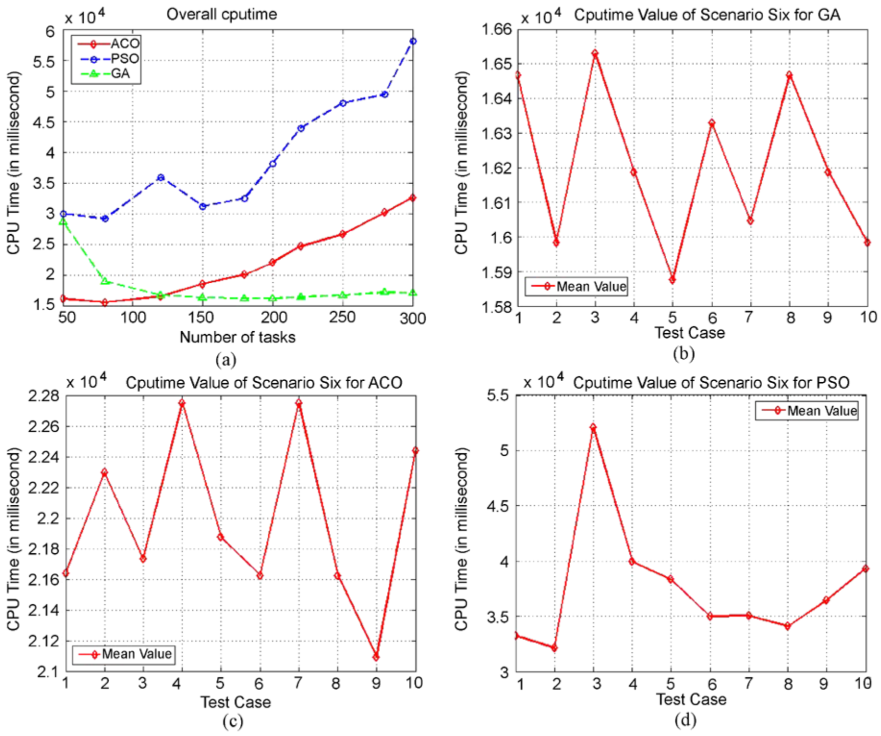


Fig. 8 CPU time of GA, ACO and PSO

6.3.3 Results for CPU time

Figure 8 shows the experimental results on the CPU time of each algorithm. Figure 8(a) presents the CPU time in 10 different scenarios; Fig. 8(b), 8(c), 8(d) show the CPU time of scenario 6 of GA, ACO, and PSO, respectively.

As can be seen in Fig. 8(a), PSO consumes much more CPU time than others. In the first scenario with 50 tasks, PSO and GA use about 30 seconds while ACO only needs 16 seconds. When the workflow size increases, the CPU time of PSO is growing dramatically. For example, for the last scenario with 300 tasks, it takes PSO about 58 seconds while ACO only uses 33 seconds. An interesting phenomenon is that the CPU time used by GA goes down at first and keeps steady at about 17 seconds. As can be seen in Figs. 8(b), 8(c), 8(d), the CPU time of each algorithm is very dynamic though in the same scenario. Furthermore, if we take a look at the three basic measurements for the same test case of specific algorithm, it shows that a larger CPU time often associates with a lager optimisation rate on makespan (or cost). For example, the CPU time of test case 4 is larger than that of test 3, and accordingly as can be seen in Fig. 6(c), the optimisation rate on makespan of test case 4 is larger than that of test 3. Another example, the CPU time of test case 7 is larger than that of test 8, and accordingly as can be seen in Fig. 7(c), the optimisation rate on cost of test case 7 is larger than that of test 8. However, larger CPU time normally cannot guarantee larger optimisation rate on makespan and cost simultaneously. This is reasonable since larger optimisation rate on makespan normally implies the more usage of expensive virtual machines with faster speed, and hence increase the cost.

6.3.4 Further discussion

(1) *For the optimisation rate on makespan:* A distinctive phenomenon observed is that when the number of tasks is more than 200, ACO yields a better performance. This implies that the ACO is more effective in solving large size discrete multiple constraints optimisation problem. One of the possible reasons for that is because ACO constructs the valid solutions task by task while PSO and GA search for valid solutions randomly in the searching space. Therefore, ACO generated solutions are all constraints satisfied. But GA and PSO cannot guarantee the validity of generated solutions. Meanwhile, due to the same reason, ACO can normally explore a much larger searching space than GA and PSO. Therefore, ACO can still maintain a good performance when the workflow size is getting larger. In contrast, GA and PSO tend to have premature convergence due to the limited searching space.

(2) *For the optimisation rate on cost:* In this paper, the scheduling problem is a two-constraint optimisation problem where the fitness value for each solution is defined based on both makespan and cost. Solutions with higher fitness values are regarded as better solutions. As explained, in our experiment, the best solution is selected as the one with the minimum makespan and its corresponding cost is regarded as the best cost. Therefore, the optimisation rate on cost is highly related to the optimisation rate on makespan. For example, in scenario six, ACO can achieve the makespan optimisation rate of 19.5% and at the same time achieve the cost optimisation rate of 7.6%. Based on Fig. 6(a) and Fig. 7(a), the conclusion can be drawn that ACO has a better ability to optimise cost than optimise makespan. One of the possible reasons is that in ACO, 50% ants are duration-greedy, i.e. dedicate to makespan optimisation only, while the other 50% are cost-greedy, i.e. dedicate to cost optimisation only. In GA and PSO, the balance between makespan optimisation and cost optimisation is only adjusted by fitness value. In other words, each individual in ACO has its social role for either makespan optimisation or to cost optimisation. In GA and PSO, individuals are evolving to achieve higher fitness value but without the difference of social roles. Therefore, it implies that the performance of algorithms having specialised social roles is better than that of those without.

(3) *For CPU time:* Generally speaking, the problem of larger overheads is the main drawback of metaheuristics based algorithms. Therefore, in our experiments, compound stopping condition is designed to control but also give reasonable CPU time for the optimisation process. Since ACO constructs and optimises the solutions task by task, its CPU time increases steadily with the growth of workflow size. By tracing the program, it is found that the most time consuming part of PSO is UPDATE (*pbest*) (Line 18 in the pseudo code of PSO). For UPDATE (*pbest*), every particle's fitness needs to be calculated where the calculation of makespan and the calculation of cost are invoked at the same time. As for the interesting phenomenon that the CPU time used by GA goes down at first and keeps steady at about 17 seconds, also by tracing the program, it is shown that when the workflow size is small, e.g. 50, GA

iterates more than the minimum iteration times (i.e. 100). However, when the workflow size goes up over 100, GA encounters the problem of premature convergence, and hence the iteration times are normally equal to 100 or just a little more than that. Therefore, the CPU time is almost the same, and thus also explains why GA has a relatively stable performance.

6.3.5 Summary

Given the experimental results presented above, there is not a single candidate among GA, ACO, and PSO of which the average performance is distinctively better than others in all scenarios on the three basic measurements. However, we can have the following conclusions:

(1) *For optimisation rate on makespan:* GA has a stable and moderate performance among the three; the performance of ACO is increasing rapidly when the workflow size grows; the performance of PSO is quite dynamic and it is on a down trend when the workflow size becoming larger. (2) *For optimisation rate on cost:* GA and PSO both have a stable and moderate performance among the three; ACO behaves better than other and especially when the workflow size is becoming larger. (3) *For CPU time:* PSO has much larger CPU time than others. The CPU time of GA is becoming stable when the workflow size is large. The CPU time of ACO is larger than that of GA but much less than that of PSO.

To sum up, GA has the moderate performance among the three. PSO is a better candidate for medium size scheduling problem, e.g. the number of workflow tasks is ranging from 150 to 200. The performance of ACO is on the rise with the growing of the workflow size. When the workflow size is small, its overall performance is the best or closest to the best. Therefore, given such results, we recommend ACO based task-scheduling algorithm as the best candidate among the three for task-level scheduling in our hierarchal scheduling strategy.

7 Related work

A cloud workflow system is built on the novel cloud computing infrastructure [2]. Cloud computing is the latest computing paradigm which adopts the market-oriented business model where users are charged for their consumption of computing resources, similar to the consumption of utilities such as water, gas, and electricity in our everyday life. The advent of cloud computing is based on the recent development and application in the area of high performance distributed computing such as cluster, peer to peer (P2P), and grid computing [16, 45, 47]. Especially with the marketed-oriented grid computing [34], the traditional community based computing paradigm is involved into utility based. Such an innovation brings many challenges for resource management in cloud workflow systems, and among many others, cloud workflow scheduling is one of the most important issues.

Workflow scheduling are classical NP-complete problems [43, 50]. Therefore, many heuristic algorithms are proposed. The work in [50] has presented a systematic overview of workflow scheduling algorithms for grid computing. The major grid workflow scheduling algorithms have been classified into two basic categories which are best-effort based scheduling and QoS-constraint based scheduling. In traditional community based computing paradigms, best-effort based scheduling strategies are often applied to only minimise the execution time without considering the monetary cost since resources are shared freely among system users. On the contrary, in market-oriented computing paradigms, QoS-constraint based scheduling strategies are employed to optimise performance under important QoS constraints, e.g. makespan minimisation under cost constraints or cost minimisation under time constraints. Many heuristic algorithms such as Minimum Execution Time, Minimum Completion Time, Min-min, Max-min are used as candidates for best-effort based scheduling strategies [39]. As for QoS-constraint based scheduling, some metaheuristic methods such as GA (Genetic Algorithm), ACO (Ant Colony Optimisation) and PSO (Particle Swarm Optimisation) have been proposed and exhibit satisfactory performance [9, 10, 25, 49, 50, 52].

In cloud workflow systems, QoS-constraint based scheduling strategies, namely metaheuristic algorithms, are required. The three most representative metaheuristic algorithms for workflow scheduling investigated in this paper are GA, ACO, and PSO. In recent years, GA has been adopted to address large complex scheduling problems and proved to be quite effective in many distributed and dynamic resource environments, such as parallel processor systems and grid workflow systems [27, 28, 30, 32, 49]. ACO, a type of optimisation algorithm inspired by the foraging behaviour of real ants in the wild, has been adopted to address large complex scheduling problems and proved to be quite effective in many application scenarios [11, 25, 31, 51]. The work in [10] proposes an ant colony optimisation approach to address scientific workflow scheduling problems with various QoS requirements such as reliability constraints, makespan constraints and cost constraints. A balanced ACO algorithm for job scheduling is proposed in [5] which can balance the entire system load while trying to minimise the makespan of a given set of jobs. PSO is a relatively new type of searching algorithm. PSO is a stochastic, population-based algorithm modelled on swarm intelligence that finds a solution to an optimisation problem in a search space [51]. In recent years, PSO has been applied to address scheduling problems [21, 52] in many different application scenarios such as grid computing, service-flow and flow-shop.

Up to date, the research on cloud computing is in its infancy so is the research on cloud workflow scheduling. To the best of our knowledge, this is the first paper that systematically analyses the problem of cloud workflow scheduling and proposes a practical solution, i.e. a hierarchical scheduling strategy.

8 Conclusions and future work

With the emerging of cloud computing, cloud workflow systems are designed to facilitate the cloud infrastructure to support large scale distributed collaborative e-business and e-science applications. One of the most important aspects which differentiate a cloud workflow system from its other counterparts is the market-oriented business model where users are charged for their consumption of the cloud resources such as computing, storage and network. This is a significant innovation which brings many challenges to the resource management in conventional workflow systems, especially for workflow scheduling strategies. Specifically, as introduced in Sect. 2.2, there are at least two basic changes which bring major challenges for workflow scheduling in market-oriented cloud workflow systems: (1) from best-effort based scheduling to QoS-constraint based scheduling; (2) from specific application oriented scheduling to general service oriented scheduling. The first change requires QoS constraint based workflow scheduling strategies in cloud workflow systems instead of conventional best-effort based strategies. The second change requires that cloud workflow scheduling strategies can deal with cloud services which are either within or outside their own data centre, and have the ability to handle intensive requests of workflow applications.

In order to address these challenges, this paper proposed a market-oriented hierarchical scheduling strategy which consists of a service-level scheduling and a task-level scheduling. The service-level scheduling deals with the Task-to-Service assignment where tasks of individual workflow instances are mapped to cloud services in the global cloud markets based on their functional and non-functional QoS requirements; the task-level scheduling deals with the optimisation of the Task-to-VM assignment in local cloud data centres where the overall running cost of cloud workflow systems will be minimised given the satisfaction of QoS constraints for individual tasks. Based on a systematic analysis of workflow scheduling in cloud workflow systems in Sect. 2, the overview of our hierarchical scheduling strategy with the detailed processes for both service-level scheduling and task-level scheduling are described in Sect. 3. Afterward, under the hierarchical scheduling strategy, a package based random scheduling algorithm has been presented in Sect. 4 as the candidate for service-level scheduling which can assign workflow tasks to suitable cloud services with correct functional and non-functional QoS requirements in an efficient fashion. As for task-level scheduling, three representative metaheuristics based scheduling algorithms including genetic algorithm (GA), ant colony optimisation (ACO), and particle swarm optimisation (PSO) have been adapted, implemented, and analysed as candidate task-level scheduling algorithms in Sect. 5. In this paper, the three metaheuristics have been adapted so that they can optimise both makespan and cost simultaneously. Meanwhile, considering the enormous overhead of optimising the Task-to-VM assignment in the entire data centre, in our strategy, the task-level scheduling algorithms are run in a parallel fashion where each instance only deals with a local Task-to-VM list. Each individual local Task-to-VM list is defined by an integrated Task-to-VM list for a group of virtual machines, and they together cover the Task-to-VM list of the entire data centre. As such, the overall running cost of the data centre will be reduced since the task-level scheduling algorithm will optimise the Task-to-VM assignment in each group of virtual machines.

The simulation experiments conducted in our SwinDeW-C cloud workflow system have demonstrated the effectiveness of our hierarchical scheduling strategy. Meanwhile, the experimental results show that ACO performs better than GA and PSO in the overall performance on three basic measurements: the optimisation rate on makespan, the optimisation rate on cost and the CPU time. Therefore, a recommended solution for workflow scheduling in cloud workflow systems is our market-oriented hierarchical scheduling strategy where the service-level scheduling adopts the package based random scheduling algorithm and the task-level scheduling adopts the ACO based scheduling algorithm. But note that based on our hierarchical scheduling strategy, different system designers may select different candidate scheduling algorithms according to his/her personal preferences on the three basic measurements and more.

In the future, more heuristics and metaheuristics will be investigated for both service-level scheduling and task-level scheduling algorithms. Meanwhile, in order to overcome the enormous overhead, our current strategy decomposes the optimisation of global Task-To-VM assignment in the entire data centre into parallel optimisation of local Task-To-VM assignment in many small groups of virtual machines. However, the optimal solution for the global Task-To-VM assignment may not be found. Therefore, scheduling algorithms which can effectively tackle large size scheduling problems in cloud workflow systems are required to be investigated in the future. Meanwhile, some real world applications will be implemented in our SwinDeW-C workflow system to further evaluate the performance of these scheduling strategies. These results will be demonstrated in our future work.

Acknowledgements This work is partially supported by Australian Research Council under Linkage Project LP0990393, the National Natural Science Foundation of China project under Grant No. 70871033.

References

1. Ardagna D, Pernici B (2007) Adaptive service composition in flexible processes. IEEE Trans Softw Eng 33(6):369–384
2. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616
3. Cao H, Jin H, Wu X, Wu S (2009) ServiceFlow: QoS-based hybrid service-oriented grid workflow system. J Supercomput 53(3):371–393
4. Cao H, Jin H, Wu X, Wu S, Shi X (2009) DAGMap: efficient and dependable scheduling of DAG workflow job in grid. J Supercomput 51(2):201–223
5. Chang R, Chang J, Lin P (2009) An ant algorithm for balanced job scheduling in grids. Future Gener Comput Syst 25(1):20–27
6. Chen J, Yang Y (2007) Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in grid workflow systems. ACM Trans Auton Adaptive Syst 2(6)
7. Chen J, Yang Y (2007) Multiple states based temporal consistency for dynamic verification of fixed-time constraints in grid workflow systems. Concurr Comput, Pract Exp 19(7):965–982
8. Chen J, Yang Y (2010) Temporal dependency based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems. In: ACM trans on software engineering and methodology, in press. http://www.ict.swin.edu.au/personal/yyang/papers/ACM-TOSEM-checkpoint.pdf. Accessed on 1st August 2010
9. Chen W, Zhang J, Chung HSH, Zhong W, Wu W, Shi Y (2009) A novel set-based particle swarm optimization method for discrete optimization problems. IEEE Trans Evol Comput 14(2):278–300
10. Chen W, Zhang J (2009) An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. IEEE Trans Syst Man Cybern, Part C, Appl Rev 39(1):29–43
11. Chen W, Zhang J, Yu Y (2007) Workflow scheduling in grids: an ant colony optimization approach. In: Proc 2007 IEEE congress on evolutionary computation, pp 3308–3315
12. Chin S, Suh T, Yu H (2009) Adaptive service scheduling for workflow applications in service-oriented grid. J Supercomput 52(3):253–283
13. Deelman E, Gannon D, Shields M, Taylor I (2008) Workflows and e-science: an overview of workflow system features and capabilities. Future Gener Comput Syst 25(6):528–540

14. Dou WC, Chen JJ, Liu JX, Cheung SC, Chen GH, Fan SK (2008) A workflow engine-driven SOA-based cooperative computing paradigm in grid environments. Int J High Perform Comput Appl 22(3):284–300
15. Thomas E (2008) SOA: principles of service design. Prentice Hall, New York
16. Foster I, Kesselman C (2004) The grid: blueprint for a new computing infrastructure, 2nd edn. Morgan Kaufmann, San Mateo
17. Foster I, Yong Z, Raicu I, Lu S (2008) Cloud computing and grid computing 360-degree compared. In: Proc 2008 grid computing environments workshop, pp 1–10
18. Hadoop. http://hadoop.apache.org/. Accessed on 1st August 2010
19. Han JW, Kamber M (2006) Data mining: concepts and techniques, 2nd edn. Elsevier, Amsterdam
20. Kao B, Molina HG (1997) Deadline assignment in a distributed soft real-time system. IEEE Trans Parallel Distrib Syst 8(12):1268–1274
21. Kuo I, Horng S, Kao T, Lin T, Lee C, Terano T, Pan Y (2009) An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. Expert Syst Appl 36(3):7027–7032, Part 2
22. Kwok Y, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multi-processors. ACM Comput Surv 31(4):406–471
23. Law AM, Kelton WD (2007) Simulation modelling and analysis, 4th edn. McGraw-Hill, New York
24. Liu K, Chen J, Yang Y, Jin H (2008) A throughput maximization strategy for scheduling transaction-intensive workflows on SwinDeW-G. Concurr Comput, Pract Exp 20(15):1807–1820
25. Liu X, Chen J, Wu Z, Ni Z, Yuan D, Yang Y (2010) Handling recoverable temporal violations in scientific workflow systems: a workflow rescheduling based strategy. In: Proc 10th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid10), Melbourne, Australia, pp 534–537
26. Liu X, Chen J, Yang Y (2008) A probabilistic strategy for setting temporal constraints in scientific workflows. In: Proc 6th international conference on business process management (BPM2008), Milan, Italy, vol 5204, pp 180–195
27. Liu X, Yang Y, Chen J, Wang Q, Li M (2009) Achieving on-time delivery: a two-stage probabilistic scheduling strategy for software projects. In: Proc the 2009 international conference on software process: trustworthy software development processes, Vancouver, BC, Canada. Springer, Berlin
28. Liu X, Yang Y, Jiang Y, Chen J (2010) Preventing temporal violations in scientific workflows: where and how. IEEE Trans Softw Eng. doi:10.1109/TSE.2010.99. http://www.ict.swin.edu.au/personal/xliu/papers/TSE-2010-01-0018.pdf. Accessed on 1st August 2010
29. Martinez A, Alfaro FJ, Sanchez JL, Quiles FJ, Duato J (2007) A new cost-effective technique for QoS support in clusters. IEEE Trans Parallel Distrib Syst 18(12):1714–1726
30. Moore M (2004) An accurate parallel genetic algorithm to schedule tasks on a cluster. Parallel Comput 30(1):567–583
31. Mullen RJ, Monekosso D, Barman S, Remagnino P (2009) A review of ant algorithms. Expert Syst Appl 36(6):9608–9617
32. Oh J, Wu C (2004) Genetic-algorithm-based real-time task scheduling with multiple goals. J Syst Softw 71:245–258
33. Raghavan B, Ramabhadran S, Yocum K, Snoeren AC (2007) Cloud control with distributed rate limiting. In: Proc 2007 ACM SIGCOMM, Kyoto, Japan, pp 337–348
34. Buyya R, Bubendorfer K (2009) Market oriented grid and utility computing. Wiley, New York
35. Russell N, van der Aalst WMP, ter Hofstede AHM (2006) Exception handling patterns in process-aware information systems. Technical Report BPM-06-04, BPMcenter.org
36. Sahai A, Durante A, Machiraju V (2002) Towards automated SLA management for web services. Technical Report HPL-2001-310 (R.1), HP Laboratories Palo Alto
37. SECES (2008) Proc first international workshop on software engineering for computational science and engineering, in conjunction with the 30th international conference on software engineering (ICSE2008), Leipzig, Germany, May 2008
38. Taylor IJ, Deelman E, Gannon DB, Shields M (2007) Workflows for e-science: scientific workflows for grids. Springer, Berlin
39. Tracy DB, Howard JS, Noah B (2001) Comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distrib Comput 61(6):810–837
40. van der Aalst WMP, Hee KMV (2002) Workflow management: models, methods, and systems. MIT Press, Cambridge

41. VMware. http://www.vmware.com/. Accessed on 1st August 2010
42. Wang M, Kotagiri R, Chen J (2009) Trust-based robust scheduling and runtime adaptation of scientific workflow. Concurr Comput, Pract Exp 21(16):1982–1998
43. Wieczorek M, Prodan R, Hoheisel A (2007) Taxonomies of the multi-criteria gridworkflow scheduling problem. CoreGRID Technical Report Number TR-0106, August 30
44. Yan J, Yang Y, Raikundalia GK (2006) SwinDeW—a peer-to-peer based decentralized workflow management system. IEEE Trans Syst Man Cybern, Part A, Syst Hum 36(5):922–935
45. Yang Y, Liu K, Chen J, Lignier J, Jin H (2007) Peer-to-peer based grid workflow runtime environment of SwinDeW-G. In: Proc 3rd international conference on e-science and grid computing (e-Science07), Bangalore, India, pp 51–58
46. Yang Y, Liu K, Chen J, Liu X, Yuan D, Jin H (2008) An algorithm in SwinDeW-C for scheduling transaction-intensive cost-constrained cloud workflows. In: Proc 4th IEEE international conference on e-science (e-Science08), Indianapolis, USA, pp 374–375
47. Yang Z, Koelbel C, Cooper K (2009) Hybrid Re-scheduling mechanisms for workflow applications on multi-cluster grid. In: Proc 9th IEEE/ACM international symposium on cluster computing and the grid, pp 116–123
48. Yu J, Buyya R (2005) A taxonomy of workflow management systems for grid computing. J Grid Comput 3:171–200
49. Yu J, Buyya R (2006) Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci Program 14(3–4):217–230
50. Yu J, Buyya R (2008) Workflow scheduling algorithms for grid computing. In: Xhafa F, Abraham A (eds) Metaheuristics for scheduling in distributed computing environments. ISBN: 978-3-540-69260-7. Springer, Berlin
51. Zhang L, Chen Y, Yang B (2006) Task scheduling based on PSO algorithm in computational grid. In: Proceedings—ISDA 2006: Sixth international conference on intelligent systems design and applications, Jinan, vol 2, pp 696–701
52. Zhang XD, Li XP, Wang Q, Yuan YC (2008) Hybrid particle swarm optimization algorithm for cost minimization in service-workflows with due dates. Tongxin Xuebao/Commun 29(8)