Engine Design and Resource Scheduling for Scientific Workflow Application Platforms in Supercomputing Environments: Postprint

Authors: Li Yufeng, Mo Zeyao, Xiao Yonghao, Zhao Shicao, Duan Bowen

Date: 2018-07-09T00:00:00+00:00

Abstract

The complexity of high-performance computer architectures imposes heightened requirements on users, and in practical engineering and scientific experiments, it is typically necessary to employ multiple application software systems that collaborate to solve complex problems. To address the usability of supercomputing resources and the integration and collaboration requirements of diverse software systems, we have developed a scientific workflow application platform for supercomputing environments, designed an asynchronous concurrent workflow execution engine, adopted a design strategy that separates scheduling algorithms from schedulers and engines, and proposed a resource scheduling scheme. We present a local resource pooling technology and a resource reservation algorithm, and conduct a comparative analysis of the performance of five commonly used scheduling algorithms, providing recommendations for algorithm selection. Practical applications demonstrate that the designed engine can support flexible execution patterns for complex workflows, and that the proposed resource scheduling scheme can satisfy the efficient execution requirements of workflow applications in supercomputing environments.

Full Text

Engine Design and Resource Scheduling of Scientific Workflow Application Platform in Supercomputing

Li Yufeng[1], Mo Zeyao[1,2], Xiao Yonghao[1]†, Zhao Shicao[1], Duan Bowen[1]

[1]Institute of Computer Application, China Academy of Engineering Physics, Mianyang Sichuan 621900, China

[2]Institute of Applied Physics & Computational Mathematics, Beijing 100094, China

Abstract: The rapid development of HPC hardware has given rise to high complexity in high-performance computer architectures, making it difficult for users to fully utilize HPC resources. In general, numerous software packages must cooperate together to achieve specific objectives in scientific experiments and engineering domains. This paper describes a new scientific workflow application platform in HPC environments. This platform contains an engine with a high-concurrency asynchronous framework to process workflow applications. Scheduler, planner, and engine are decoupled from each other, allowing these three components to develop independently: the scheduler for scheduling algorithm implementation, the planner for collecting scheduling information, and the engine for workflow driving. Scheduler and planner use resource advance reservation and local pooling mechanisms to increase workflow execution performance. This paper also implements and compares five scheduling algorithms regarding their performance on test graph sets, and provides useful guidance for algorithm selection. Real applications demonstrate that the engine can support various execution strategies and that the resource scheduling solution can help increase the efficiency of workflow execution in supercomputing environments.

Key words: scientific workflow; high performance computing; resource scheduling; workflow engine

0 Introduction

Numerical simulation plays a crucial role in scientific experiments and engineering practice. With the rapid development of high-performance computers, complex heterogeneous architectures have become mainstream. For instance, parallel machines ranked at the top of TOP500[1] all employ various acceleration devices and multiple cluster systems. When developing software, developers must consider multi-level nested parallelism and energy consumption issues. When using HPC application software, users still need to be familiar with the computing environment to better leverage the advantages of both hardware and software. In scientific discovery and engineering practice, solving a practical problem typically requires using a series of related software. A typical workflow involves modeling, simulation, and visual analysis, which may have data dependencies among them. Only through mutual collaboration among these software packages can final results be obtained. In 2015, the United States issued a presidential executive order creating the National Strategic Computing Initiative (NSCI)[2], which explicitly identified as its second objective the enhancement of coherence among various technological foundations required for modeling, simulation, and data analysis computing. Scientific workflow is one of the key technologies to address such challenges. In supercomputing environments, utilizing scientific workflow technology to transparentize supercomputing resources, shielding users from details such as job submission and data management, and providing a simple interface for workflow assembly and execution can greatly promote the collaborative development of supercomputing environments and application software.

In the application of scientific workflows, the process is generally divided into static composition stage and dynamic execution stage. The composition stage addresses user customization of workflows, including workflow editing and component property editing. The execution stage includes executing the workflow engine to drive process execution, scheduling component tasks on HPC resources, and monitoring task status. This paper focuses on the design of the workflow engine and resource scheduling during the execution stage. Scientific workflow involves the on-demand composition of steps such as data collection, processing, analysis, and visualization in complex scientific research processes, along with service selection and resource mapping matching, automatically executing in distributed heterogeneous resource environments according to custom logical relationships. Xiao Fei et al.[3] divided workflow development into three stages: the embryonic period characterized by integrated software packages, commercial workflows, and scientific workflows. The main characteristics distinguishing scientific workflows from traditional business workflows are their process complexity, data orientation, and need for dynamic adaptability.

Many workflow platforms have emerged both domestically and internationally. For example, the Kepler[4] system developed by researchers at the University of California originated in 2002 and was developed based on the Ptolemy II system from UC Berkeley. Kepler is an active open-source project that has currently released version 2.5. Researchers from many fields, including ecology, molecular biology, chemistry, computer science, electronic engineering, and oceanography, have contributed to Kepler. The Pegasus[5] system developed by the Information Sciences Institute (ISI) at the University of Southern California can map abstract workflows to concrete execution environments and execute them. The technology provided can deploy workflows to different environments, including desktop environments, campus clusters, grids, and clouds. As of May 2018, the latest version 4.8.2 has been released, and the system has been applied in the LIGO project to assist in gravitational wave detection analysis. Taverna[6] is a workflow suite developed by the School of Computer Science at the University of Manchester, originating from the myGrid consortium. MyGrid aims to implement e-Science and e-Laboratories, initiated by multiple organizations collaborating to provide many practical tools for scientists to conduct electronic research, many of which have been widely applied in biological sciences, social sciences, chemistry, astronomy, music, and multimedia fields. Taverna is one such tool used to design, edit, and execute scientific workflows. It performs pipeline assembly of distributed network services and local tools to complete complex analysis processes. It can execute not only in local desktop environments but also on large-scale infrastructures such as supercomputing environments, grids, and cloud computing environments. Over 350 academic and commercial organizations worldwide use Taverna to accelerate their innovative research. Wang Hongxia[7] designed a workflow engine based on the GT4 grid environment, but the focus was on grid services and their collaborative relationships rather than the scientific workflow system of concern in this paper. Shen Yu et al.[8] designed and implemented a high-performance resource unified man-

agement system focusing on the unified management and auditing of multiple high-performance resources. The resource pre-allocation and timed auditing mechanisms for users can make more full use of resources, while the resource reservation and pooling strategy proposed in this paper can better meet the efficient execution requirements of workflow applications.

Existing scientific workflow systems commonly suffer from several problems: task granularity is too fine, with even simple sorting and algebraic operations being set as components, resulting in cumbersome usage; configuration of underlying systems is complex, leading to high deployment costs; most workflow execution relies on the scheduling methods of the execution system without workflow-specific scheduling strategies, resulting in low scheduling efficiency. The HSWAP platform from the Institute of Computer Application, China Academy of Engineering Physics, uses high-performance application software as the basic unit for component encapsulation, features simple configuration and easy deployment, and supports cross-platform application collaboration. The scheduling adopts resource reservation and local resource pooling technology, supports multiple resource scheduling algorithms, and possesses process-level DAG scheduling algorithms that can effectively improve process execution efficiency, making it a powerful tool for integrating multiple types of software for numerical simulation and data analysis in HPC environments.

1 HPC Scientific Workflow Application Platform HSWAP

1.1 HSWAP Overview

HSWAP is a supercomputing coherent computing platform developed by the Supercomputing Application Team at the Institute of Computer Application, China Academy of Engineering Physics, aimed at using scientific workflow technology in HPC environments to provide integrated supercomputing service models that help researchers improve work efficiency. In scientific workflows, a job refers to a complete process; a job consists of several tasks, each encapsulated as an execution component in HSWAP, representing a task of modeling, simulation, or visual analysis. Components can have user-defined attributes, such as input and output settings. Components can have data dependency relationships among them. A workflow can be represented by a directed acyclic graph (DAG), denoted as G=(V,E), where V represents the set of task components and E represents the set of dependency relationships among task nodes. Directed edges represent dependencies between tasks, with the starting point task called the predecessor task node of the endpoint task, and the endpoint task called the successor task node of the starting point task.

The HSWAP interface is shown in Figure 1 [Figure 1: see original paper]. The left side of the interface displays the component library, allowing users to add instances to the workflow diagram through drag-and-drop. The right side is the workflow composition and display area, where component instances are added via drag-and-drop, component properties are edited by double-clicking, and de-

pendencies between components are defined by drawing lines with the mouse. Control buttons for execution are provided at the top. During the execution stage, different colors can display the running status. When switching to the application tab, real-time interaction with component application graphical interfaces is possible.

1.2 HSWAP Architecture Design

To support numerical simulation services and product design processes in different fields and integrate the commonalities of application technologies across various scenarios, the overall HSWAP architecture is designed as shown in Figure 2 [Figure 2: see original paper]. The platform is implemented using a B/S model, requiring only a web browser on the client side, with specific business logic implemented on the server side. Deployment is very convenient, requiring only a single server-side platform software deployment without client deployment, enabling multi-user concurrent access.

The control layer is responsible for workflow design API interfaces and workflow execution API interfaces, primarily connecting users' visual editing and execution control on the graphical browser client with backend command implementation. Flexible and powerful API interface design ensures platform stability. The business logic layer describes business logic within various industry domains, such as the modeling-simulation-visual analysis process in structural mechanics. In other fields like biology and materials, similar processes exist, though the workflow structure may differ. The infrastructure services layer and data layer are the core implementations of the platform. The infrastructure services layer provides functions such as component management, workflow creation, workflow execution engine, resource scheduling, and application proxy, while the data layer manages various types of metadata. HSWAP' s distinctive features include pushing remote application software interfaces to embedded browser clients to support real-time interaction, supporting cross-platform (Windows, Linux, and other remote systems) software application component configuration and collaborative execution for the same workflow instance, and flexible execution control strategies.

2 Asynchronous Concurrent Workflow Engine Design

The platform' s scientific workflows are represented as directed acyclic graphs. The purpose of workflow engine design is to provide process execution logic, drive process execution, monitor the running status of tasks within the process, and promptly feed back user operation instructions. The relationship between the engine and other modules is shown in Figure 3 [Figure 3: see original paper]. The user layer sends instructions to the engine, such as starting projects, changing execution strategies, pausing, or terminating workflow execution. The engine responds to these commands, performs related operations, and promptly feeds back task status in the workflow to users. For example, when a user starts a project, the engine retrieves the abstract workflow from the workflow creation

database, performs process parsing and execution. When executing task components, subtasks are submitted to application proxy servers on high-performance computers, which monitor subtask running status and feed back status to the engine in a timely manner. Task dependencies in the workflow are interpreted and processed by the engine. Related data copying and migration are also first triggered by the engine and then handled by the data management module.

In scientific workflow applications, since the engine must both monitor task component execution and promptly handle user instructions from the frontend, the engine design must adopt a concurrent pattern. Additionally, due to the close correlation between user operations and component status changes, and because some operations require longer processing times, asynchronous design is adopted to improve response time and enhance user experience. The asynchronous concurrent workflow engine consists of an event loop main body. Figure 4 [Figure 4: see original paper] shows a schematic diagram of its basic execution architecture. Around a workflow instance, a series of steps are performed, including DAG topological sorting, runnable job generation, resource scheduling, and job submission. External API calls can be responded to at various stages, including execution control APIs and job status setting APIs. The former is triggered by the user interface frontend, while the latter is triggered by the server-side application proxy. The workflow engine responds to external API calls in a timely manner and decides on appropriate actions based on current job running status and workflow state.

The engine is implemented in Python, using asynchronous IO to implement the event loop functionality and a high-performance RPC library to implement external API call interfaces, decoupling the engine from other platform modules. It has the characteristic of independent operation and can be deployed as a standalone process, supporting concurrent deployment to enable parallel execution of multiple workflows with multiple engines.

3 Resource Scheduling Based on Workflow

3.1 HSWAP Resource Scheduling Scheme

In supercomputing environments, system software is deployed with a job scheduling system, such as PBS, LSF, or SLURM, for job scheduling and allocating required resources for computing tasks. Generally, user-submitted jobs are independent and are scheduled separately by the scheduling system, each going through states of submission, queuing, startup, running, and completion. In the execution environment of scientific workflows, tasks within a workflow job have correlations—they may be parallel or have data/control dependency relationships. If these relationships are not considered and each task is directly submitted to the scheduling system of the supercomputing environment, each task may experience long queue waiting times. Moreover, subsequent tasks can only be submitted after their predecessor tasks are completed, resulting in excessively long overall workflow execution time and low efficiency.

Since task dependencies are known during scheduling, how to fully utilize these relationships to improve scheduling performance is worth studying. In HSWAP, efficient workflow execution is ensured through resource reservation and pooling technology, along with efficient DAG scheduling algorithms.

The HSWAP resource scheduling process is shown in Figure 5 [Figure 5: see original paper]. The purpose of the application proxy is to provide a unified interface for task execution and monitoring. The platform establishes a dynamic resource pool for HSWAP based on the application proxy and high-performance computing hardware/software environments, configuring a scheduling algorithm library for the scheduler (step in the figure). After configuration, the user' s workflow task scheduling uses resources from the dynamic resource pool, and the scheduler uses scheduling algorithms from the algorithm library. When a user executes the workflow engine, the engine passes complete workflow information to the scheduling planner (step). The scheduling planner obtains available resources from the dynamic resource pool, performs resource screening, and passes qualified resources and workflow information to the scheduler (steps -). The scheduler calls a specific algorithm and passes the scheduling result back to the planner (step). The planner returns the result to the workflow engine (step). The workflow engine submits tasks to corresponding supercomputing resources through the application proxy according to the scheduling results (steps -). Finally, the application proxy starts task execution in the actual environment and monitors the running status.

3.2 Resource Pooling and Reservation Algorithm

Implementing resource pooling management and advance reservation of resources required for workflow execution is an effective approach to shortening the overall completion time of all tasks in a workflow job and improving workflow execution efficiency.

To support multi-branch concurrent execution in workflows, sufficient resources need to be reserved to improve concurrency. However, from the perspective of resource utilization, to avoid resource idleness and waste, the best approach is to enable resource reuse among tasks in a workflow, especially between tasks with sequential constraints, aiming to meet requirements using the minimum number of resources. If a local resource pool is established through overall resource reservation to obtain resource control rights in advance, it can satisfy the execution of workflows without waiting delays. In actual environments where requirements cannot be met, resources close to the size of the overall reservation pool are reserved from available resources, and the resource pool is dynamically changed during execution to achieve dynamic scheduling. The related process is shown in Figure 7 [Figure 7: see original paper].

Today' s supercomputing centers or data centers have multiple computing resource systems with obvious heterogeneous characteristics. Tasks in workflows also have different resource requirements. This paper implements and tests five

heuristic algorithms—Min-Min, Max-Min, HEFT, HDSC, and DCP—in heterogeneous environments, analyzing their performance for scheduling various types of DAGs on resources with distributed heterogeneous features.

3.3 Scheduling Algorithm Analysis

Since the scheduling problem has NP-hard complexity, most research focuses on proposing and improving heuristic algorithms, with many studies also concentrating on cost optimization and dynamic self-adaptive scheduling with fault tolerance in distributed computing infrastructures such as grids and clouds. The Min-Min algorithm is the simplest and most commonly used heuristic algorithm, with the idea of scheduling tasks that can be completed fastest first. The Max-Min algorithm selects the task with the maximum minimum completion time (across all available resources) for scheduling. Neither Min-Min nor Max-Min algorithms consider the characteristics of the DAG graph itself, making only local decisions for ready tasks and available resources. Topcuoglu et al.[9] proposed the HEFT algorithm in 2002, which uses the upward rank value of task nodes in the DAG graph as priority for scheduling ready tasks, achieving excellent results and becoming a commonly used heuristic scheduling algorithm that has served as a benchmark for algorithm comparison since its proposal. Shi et al.[10] established a DAG representation for workflows and an undirected weighted graph representation for resources, and built a model for data communication between tasks, improving the HEFT algorithm under conditions of significant processor capability differences and proposing the AEFT algorithm. Kwok[11] proposed the DCP algorithm from the perspective of the dynamic nature of the critical path, using dynamic critical paths and inserting time slots for scheduling. Rahman[12,13] extended the dynamic critical path algorithm DCP to heterogeneous computing environments and provided detailed descriptions of various classical algorithms in the literature. In 1997, Chan et al.[14] proposed the low-complexity HDSC algorithm for heterogeneous environments, which originated from the Dominant Sequence Clustering algorithm proposed by Tao Yang in 1994, with the idea of clustering based on the critical path of the scheduling graph to optimize the scheduling scheme. George[15] proposed the MMGWS algorithm for workflow scheduling on grids, aiming for minimum completion time and studying mechanisms using advance resource reservation. Patil et al.[16] used the Maui scheduling tool and SLURM resource management and scheduling system to study scheduling methods considering energy consumption in HPC environments. Chen et al.[17] studied how subsequent job scheduling affects previously reserved resources under reservation mechanisms and automatically adjusts, considering issues caused by discrepancies between actual and estimated task execution times. Wu et al.[18] provided a survey of workflow scheduling in cloud computing environments, introducing static scheduling, dynamic scheduling, and methods combining static planning with dynamic scheduling.

For different DAG characteristics, this experiment selected two features: num-

ber of tasks and computation-to-communication ratio. For each combination, 25 DAGs were randomly generated, and the average performance of each algorithm on these graphs was compared. The evaluation metric was NSL (normalized scheduling length), using the execution time of the DAG' s critical path on the fastest resource as the baseline. NSL is defined as the ratio of the algorithm' s scheduling length for a graph to the graph' s baseline value. Clearly, the minimum NSL value is 1, and smaller values indicate better algorithm performance. The simulated resource environment consisted of a hybrid resource environment with two clusters, each comprising 5 machines with varying performance and different communication bandwidths within and between clusters.

Graph sizes (number of tasks) ranged from 10 to 100, sampled at intervals of 10, for a total of 10 cases. The communication-to-computation ratio (CCR) was divided into three types. Assuming the data scale processed by a task is N and the output data byte size is $8N^2$, Type I defines task computational complexity as $O(N^2)$; Type II defines task computational complexity as $O(N^2\log_2(N^2))$; Type III defines task computational complexity as $O(N^3)$. Type I has relatively larger data communication volume compared to computational volume, while Type III has smaller communication volume. For the three CCR types, experimental simulation results are shown in Figures 8 [Figure 8: see original paper] through 10 [Figure 10: see original paper]. The simulated DAG data came from the random graph generation tool daggen[19], and the resource environment was simulated using the SimGrid[20] tool. SimGrid is a general-purpose distributed system simulator that can simulate grid and cloud resource environments, as well as MPI program environments and high-performance computing resource systems. The five scheduling algorithms—Min-Min, Max-Min, HEFT, HDSC, and DCP—were implemented in C/C++. Based on SimGrid' s resource environment representation and host and task encapsulation APIs, the scheduling performance of the five algorithms on various DAG workflow graphs was tested.

From the figures, the following conclusions can be drawn: (a) HEFT, DCP, and HDSC algorithms perform better than Min-Min and Max-Min algorithms, indicating that algorithms considering the critical path have better performance. The HEFT algorithm has good average performance, confirming its widespread use; (b) In compute-intensive applications (as shown in Figures 9 [Figure 9: see original paper] and 10 [Figure 10: see original paper]), the HDSC algorithm performs best; (c) The DCP algorithm is suitable for DAG scheduling with large communication volume but small computation volume (such as CCR Type I), while for cases with small communication volume but large computation volume, performance slightly decreases for large-scale workflow scheduling. Additionally, the curves overall show an upward trend (with fluctuations) because, during large-scale DAG workflow scheduling, heuristic scheduling makes local decisions, causing performance gaps to accumulate compared to the ideal baseline. NSL values in Figure 8 are generally larger than those in Figures 9 and 10 because CCR Type II in Figure 9 and Type III in Figure 10 correspond to compute-intensive workflows with smaller communication data volumes (or equivalently, communication resources with high bandwidth), while CCR Type I in Figure 8

corresponds to cases with more data transmission or larger data volumes. The baseline used for calculating NSL does not consider data communication, only considering the execution time of computational tasks on the critical path on infinitely many resources with the fastest processors. When data transmission is more frequent or data volume is larger, the actual NSL of scheduling inevitably increases.

The HDSC algorithm uses backtracking techniques and clustering technology to reduce data communication time and improve scheduling efficiency. However, it only employs one layer of backtracking and clustering of adjacent single-parent-child tasks, leaving room for optimization, which provides enlightenment for developing new scheduling algorithms. Additionally, scheduling of HPC scientific workflows on large-scale computing resources can draw on excellent methods and latest algorithms from grid and cloud scheduling, such as resource reservation billing strategies and dynamic adaptive scheduling. Research in this area will effectively improve supercomputing resource utilization and workflow execution efficiency.

4 Conclusion

This paper introduced the supercomputing coherent computing platform HSWAP and its workflow engine design and resource scheduling scheme. The platform shields users from the complexity of using underlying high-performance computing hardware and software resources on one hand, and provides users with functions such as custom workflows, workflow templates, and custom task attributes on the other hand, greatly improving user work efficiency. The asynchronous concurrent workflow engine design can simultaneously meet the needs of user interactive control and workflow runtime control. The resource scheduling module adopts resource pooling technology and a design that separates resource scheduling algorithms from the scheduling planner and engine, facilitating the addition and extension of scheduling algorithms. Five commonly used heuristic scheduling algorithms were implemented in heterogeneous environments, and their performance in DAG workflow scheduling was tested and analyzed. Currently, the implemented scheduling algorithms rely on estimated task times. How to accurately estimate required task time needs further research, such as using machine learning techniques based on performance history databases to provide task runtime estimates. Another worthwhile research direction in scheduling is the impact of task estimated time errors on scheduling algorithms and adaptive adjustment mechanisms.

References

[1] Top500.org. Top 500 [EB/OL]. (2017-11-30) [2018-05-08]. https://www.top500.org/lists/2017/11/.

[2] Obama. NSCI executive order 13702 [EB/OL]. (2017-07-20) [2018-05-08]. https://obamawhitehouse.archives.gov/the-press-office/2015/07/29/executive-order-creating-national-strategic-computing-initiative.

[3] Xiao Fei, Zhang Weihua, Wang Donghui. Overview of workflow technology in scientific process [J]. Application Research of Computers, 2011, 28 (11): 4013-4019.

[4] Bertram L, Berkley A C, et al. Scientific workflow management and the Kepler system [J]. Concurrency and Computation: Practice & Experience, 2006, 18: 1039-1065.

[5] Deelman E, Singh G, Su M H, et al. Pegasus: a framework for mapping complex scientific workflows onto distributed systems [J]. Scientific Programming, 2005, 13 (3): 219-237.

[6] Wolstencroft K, Haines R, Fellows D, et al. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud [J]. Nucleic Acids Research, 2013, 1 (1): 1–5.

[7] Wang Hongxia. Design and realization of grid workflow engine [J]. Computer Engineering and Design, 2011, 32 (2): 430-433.

[8] Shen Yu, Li Juan, Chang Biao, et al. Design and Implementation of the Uniform Resource Management System of HPC [J]. Computing Technology and Automation, 2014, 33 (1): 83-90.

[9] Topcuoglu, Hariri, Y Wu. Performance effective and low complexity task scheduling for heterogeneous computing [J], IEEE Trans on Parallel and Distributed System, 2002, 13 (3): 260-274.

[10] Shi Zhiao, Dongarra J J. Scheduling workflow applications on processors with different capabilities [J]. Future Generation Computer Systems, 2006, 22: 665–675.

[11] Kwok Y K, Ahmad. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors [J]. IEEE Trans on Parallel and Distributed System, 1996, 7 (5): 506-521.

[12] Venugopal R, Buyya R. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids [C]// Proc of IEEE International Conference on e-Science and Grid Computing. 2008: 35-42.

[13] Rafiul R, Hassan M, Ranjan R, et al. Adaptive workflow scheduling for dynamic grid and cloud computing environment [J]. Concurrency and Computation: Practice & Experience, 2013, 25: 1816–1842.

[14] Chan W Y, Li C K. Heterogeneous dominant sequence cluster (HDSC): a low complexity heterogeneous scheduling algorithm [C]// Proc of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. 1997.

[15] Amalarethinam G, Selvi F K M. A minimum makespan grid workflow scheduling algorithms [C]// Proc of Conference on Computer Communication and Informatics. 2012.

[16] Patil V A, Chaudhary V. Rack aware scheduling in HPC data centers: an energy conservation strategy [J]. Cluster Computing, 2013, 16 (3): 559-573.

[17] Chen Wei, Lee Y C, Fekete A, et al. Adaptive multiple-workflow scheduling with task rearrangement [J]. The Journal of Supercomputing, 2015, 71 (4): 1297-1317.

[18] Wu Fuhui, Wu Qingbo, Tan Yusong. Workflow scheduling in cloud: a survey [J]. Journal of Supercomputing, 2015, 71 (9): 1-46.

[19] Frederic Suter. DAGGEN [EB/OL]. (2017-07-26) [2018-05-08]. https://github.com/frs69wq/daggen.

[20] Casanova H, Giersch A, Legrand A, et al. Versatile, scalable, and accurate simulation of distributed applications and platforms [J]. Journal of Parallel and Distributed Computing, 2014, 74 (10): 2899-2917.

Note: Figure translations are in progress. See original paper for figures.