Hu et al. Journal of Cloud Computing: Advances, Systems and Applications (2020) 9:39
https://doi.org/10.1186/s13677-020-00177-8

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Intelligent cloud workflow management and scheduling method for big data applications

Yannian Hu[1], Hui Wang[2*] and Wenge Ma[3]

Abstract

With the application and comprehensive development of big data technology, the need for effective research on cloud workflow management and scheduling is becoming increasingly urgent. However, there are currently suitable methods for effective analysis. To determine how to effectively manage and schedule smart cloud workflows, this article studies big data from various aspects and draws the following conclusions: Compared with the original JStorm system, the response time is shortened by a maximum of 58.26% and an average of 23.18%, CPU resource utilization is increased by a maximum of 17.96% and an average of 11.39%, and memory utilization increased by a maximum of 88.7% and an average of 71.16%. In terms of optimizing the dynamic combination of web services, the overall performance of both the MOACO and CCA algorithms is better than that of the GA algorithm, and the average performance of the MOACO algorithm is better than that of the CCA algorithm. This paper also proposes a cloud workflow scheduling strategy based on an intelligent algorithm and realizes the two-tier scheduling of cloud workflow tasks by adjusting the combination strategy for cloud service resources. We have studied three representative intelligent algorithms (ACO, PSO and GA) and improved them for scheduling optimization. It can be clearly seen that in the same scenario, the optimal values of the different algorithms vary greatly for different test cases. However, the optimal solution curve is substantially consistent with the trend of the mean curve.

Keywords: Big data, Cloud workflow, Cloud service resource combination, Scheduling optimization

Introduction

Cloud workflow technology is an effective way to achieve process integration in a cloud computing environment. Cloud workflow management can improve and optimize business processes, improve business efficiency, achieve better business process control, improve business process flexibility, and improve customer service quality. Cloud workflows make it easy to build, execute, manage, and monitor cloud computing applications, enabling cloud computing applications to be automated and efficient. Due to the dynamic, distributed, heterogeneous and scalable nature of cloud computing, some methods and techniques for traditional workflows cannot effectively address related problems in cloud computing environments; instead, corresponding methods should be developed based on the characteristics of cloud computing resources and cloud computing applications. To this end, the cloud computing workflow architecture, the dynamic process model, the resource management model and dynamic scheduling algorithms for cloud computing workflows should be studied.

Recently, many research teams at home and abroad have begun to explore opportunities to use molecular data in cloud computing environments. In [1], the authors quantified the transcriptional expression levels of 12,307 RNA sequencing samples from the Cancer Cell Encyclopedia and the cancer genome map. The author

* Correspondence: conglinwh@wfu.edu.cn
[2]Weifang University, Weifang 261061, Shandong, China
Full list of author information is available at the end of the article

used two cloud-based configurations and examined the performance and cost profiles for each configuration. In [2], the authors found that a cloud infrastructure enables a rich set of management tasks for manipulating computing, storage, and network resources in the cloud. The authors adopted a lightweight, non-intrusive approach that is applicable only to interlaced logs, which are widely available in a cloud infrastructure. The differences found during inspection indicate potential execution issues that may or may not be accompanied by error log messages. In [3], the authors proposed an improved particle swarm optimization (IPSO) algorithm for scheduling applications to cloud resources. The IPSO algorithm minimizes the total cost of placing tasks on available resources. The results showed that the improved algorithm is effective compared to the standard particle swarm algorithm. In [4], the author introduced a process model and resource model for energy-aware workflow scheduling in a cloud computing environment. Numerical examples and simulation experiments showed the feasibility and effectiveness of the proposed method. In [5], the authors proposed an alternative architecture that is designed to be suitable for cloud computing and uses a pure software approach to deploy a dynamic software infrastructure. The authors introduced this architecture to overcome certain limitations while providing ways to handle security and scalability. In [6], the author designed a conceptual model for the overall management of all resources to improve energy efficiency and reduce the carbon footprint of a cloud data centre (CDC). The author discussed the interrelationship between energy and reliability for sustainable cloud computing and laid the foundation for further practical development. In [7, 8], the authors proposed a new cloud computing architecture known as Model as a Service (MaaS). This study presented a flexible and effective method for analysing the uncertainty and time-varying characteristics of parameters. Group data sharing in cloud environments has become a topic of great interest in recent decades. With the increasing popularity of cloud computing, the question of how to achieve secure and efficient data sharing in the cloud environment has become an urgent problem to be solved [9]. In [10], the author developed a method based on an ant colony system (ACS) to achieve the goal of virtual machine placement (VMP). The results showed that the performance of OEMACS is generally better than that of traditional algorithms.

The most important part of a cloud workflow engine is the cloud workflow scheduling strategy. In the cloud computing environment, a workflow management system needs to find a suitable service provider to run the tasks of a workflow in accordance with the user's quality of service constraints in its trusted domain, and the service provider needs to reasonably allocate the virtual computing resources in its data centre to perform the workflow tasks. In the first stage, a cloud service resource selection model can be used to select an appropriate service provider for the execution of a cloud workflow and map the sub-workflows that can be executed in parallel to the corresponding web services; in the second stage, various algorithms can be used to map the sub-workflows to the corresponding virtual computing resources and optimize them.

The greatest difference between the cloud computing environment and the traditional computing environment is that computing services can be obtained on demand and their use can be paid for at any time. At the same time, due to the dynamic, distributed, heterogeneous and autonomous nature of cloud computing, traditional workflow methods and technologies cannot effectively address the problems that arise in cloud workflow management.

Currently, massive-scale data processing technology is a highly active area of research, and much meaningful research has been carried out at home and abroad. In [11], the authors found that big data technology is increasingly used in biomedical and health informatics research. Next-generation sequencing technology can be used to process billions of DNA sequences per day, and the use of electronic health records (EHRs) is resulting in the recording of large amounts of patient data. The application of big data in healthcare is a fast-growing field, and many new discoveries and new methods have been reported in the past 5 years. In [12], the authors found that the operational cost of streaming media application providers can be greatly reduced through flexible resource allocation and centralized cloud management. In the article, the authors considered the optimal deployment problem (ODP) based on the local memory of each viewer. In [13], the authors developed an innovative method of extracting valuable pixel categories with similar evolution for specific parameters of interest over a long period of time to obtain valuable comprehensive information. Unsupervised classification was performed using an original custom method suitable for execution on such an enormous data set. In [14], the authors found that big data applications (such as medical imaging and genetics) typically generate data sets that consist of n observations with p variables, where p is larger than n. The authors considered the classification problem for such $p >> n$ data and proposed a classification method based on linear discriminant analysis (LDA). In [15], the authors found that the increase in the size and complexity of big data available via the Internet has provided unprecedented opportunities for cyber-physical systems (CPSs). To address the related problems, the authors proposed the Cyber-Physical

Space Event Model (CPSEM) to analyse the impact of events on multiple viewers. In addition, the authors proposed the Event Influence Scope Detection Algorithm (EISDA) to detect the impact range of events in cyberspace and physical space. In [16], the author proposed an innovative incremental processing technology named Stream Cube, which can process large-scale data and streaming data. The system is based on real-time acquisition, real-time processing, real-time analysis and real-time decision-making. In [17, 18], based on the theory of fluid mechanics, the author established the filter cake layer model and proposed a modified method of calculating the filter cake layer porosity. The results showed that the calculated values were in good agreement with the experimental data, and the relative error was less than 10%. In [19], the author developed a method suitable for probe management and data processing. This method is based on an evaluation of laboratory performance and adaptive field protocols for calibration, data processing and validation. In [20], the author developed an open source tool called DRomics, which can be used as an R-package or a web-based service; it has no concentration dependence or high variability and can identify the best model for describing a concentration response curve. In [21, 22], the author improved the resource utilization ratio in terms of the number of CPU cores and the memory size of virtual machines (VMs) and physical machines (PMs) and minimized the number of virtual machines and active PMs instantiated in the cloud environment. In [23], the author proposed a framework that supports mobile applications with a context-aware computing offloading function and proposed an estimation model to automatically select the cloud resources to be offloaded.

To find an effective method for intelligent cloud workflow management and scheduling, this article studies big data from various aspects. Based on the existing open source platform JStorm for real-time big data processing, a dynamic resource scheduling system named D-JStorm is designed and implemented. This paper proposes a cloud workflow scheduling strategy based on intelligent algorithms and a strategy for adjusting the combination of perceived cloud service resources to achieve two-tier scheduling of cloud workflow tasks. Three representative intelligent algorithms are studied and improved for scheduling optimization. Compared with the original JStorm system, the response time is shortened by a maximum of 58.26% and an average of 23.18%, the CPU resource utilization is increased by a maximum of 17.96% and an average of 11.39%, and the memory utilization is increased by a maximum of 88.7% and an average of 71.16%. In terms of optimizing the dynamic composition of web services, the overall performance of both the MOACO and CCA algorithms is better than that of the GA algorithm, and the average performance of the MOACO algorithm is also better than that of the CCA algorithm.

Method

Combination model for resource management based on the ant Colony algorithm

QoS assessment of the management portfolio

Let $WS = \{WS_i | \ i = 1, 2, \ldots, n\}$ be a set of n types of subtasks that need to be completed, and let $ws_j = \{ws_{ij} | \ j = (1, 2, \ldots, m_1)\}$ be a candidate web service class in the UDDI specification that can complete subtask WS_i, where mi is the number of services in the service class. Let $I_i = \{t_i, c_i, r_i, \ldots\}$ be the set of QoS evaluation indicators for service class ws_i, where t_i is the time index, c_i is the price index, r_i is the reliability index, and the ellipsis represents scalable quality indicators. Each service class indicator set is different, and t_i, c_i, and r_i for a web service can be dynamically combined to calculate public evaluation indicators for each service class, that is, QoS = execution time, execution cost, and reliability.

Definition 1 Execution time. Let $T(ws_i)$ be the execution time of service subtask ws_i; then, d $WS_{QoS}^{Tinx} = \sum_{i=1}^{n} T(ws_i)$ is the execution time of the discovery process. When a subtask is executed sequentially for several service components, $T(w_i) = \sum_{j=1}^{k} ws_j$; when the subtask is executed in parallel for several service components, $T(ws_i) = \max(T(ws_j)) \quad j = 1, 2, \ldots, k$.

Definition 2 Execution cost. Let $C(ws_i)$ be the execution cost of web service subtask ws_i; then, $WS_{Q_0S}^{Cost} = \sum_{i=1}^{n} C(ws_i)$ is the execution cost of the web discovery process.

Definition 3 Service reliability. Let $R(ws_i)$ be the service reliability of service subtask ws_i; then, $WS_{QaS}^{reliablity} = \prod_{l=1}^{n} R(ws_i)$ is the reliability of the discovery process.

Multi-objective ant colony algorithm

Since the goal of the dynamic combination problem for a web service is to select a suitable service instance from among the candidate services for each discovered subtask, the pheromone of k^s_{ij} is selected to be τ_{ij} for subtask t^k_i, and the heuristic information n_{ij} of k^s_{ij} is selected for subtask t^k_i. When the algorithm is initialized, initial values $\tau_{ij} = \tau_0, \quad 1 \le i \le n, 1 \le j \le m$, are set for the pheromones. Multiple QoS parameters with different characteristics are considered in the model. To perform multi-objective optimization, different types of heuristic information need to be defined.

Reliability-prioritized heuristic information The RP heuristic information guides ants to select highly reliable web service instances. If an ant's heuristic type is RP, the heuristic information for selecting k^s_{ij} for subtask t^k_i can be expressed as:

$$\eta_{ij} = RP_{ij} = \frac{ks^j \cdot r - min\_reliability_i + 1}{max\_reliability_i - min\_reliability_i + 1} \quad (1)$$

Here, $\min_{reliability_t} = \min_{1 \le 5.5 m_1}\{ks_i^j, r\}$, $\max_{reliability_t} = \max_{1s,sm_1}\{ks_i^j, r\}$. This formula ensures that the heuristic information is normalized to the interval (0,1) and that the higher the reliability of a web service instance is, the greater the value of its heuristic information.

Time-prioritized heuristic information The TP heuristic information guides ants to select a web service instance with a short execution time. If an ant's heuristic type is TP, the heuristic information for selecting k^s_{ij} for subtask t^k_i can be expressed as:

$$\eta_{ij} = TP_{ij} = \frac{max\_time_i - ks_i^j \cdot t + 1}{max\_time_i - min\_time_i + 1} \quad (2)$$

Here, $min\_time_i = min_{1 \le j \le m_i}\{ks_i^j \cdot t\}_i$, $max\_time_i = max_{1 \le j \le m_1}\{ks_i^j \cdot t\}$. This formula ensures that the heuristic information is normalized to the interval (0, 1) and that the shorter the execution time of a web service instance is, the greater its heuristic information value.

Cost-prioritized heuristic information The CP heuristic information guides ants to select a web service instance with a low execution cost. If an ant's heuristic type is CP, the heuristic information for selecting k^s_{ij} for subtask t^k_i can be expressed as:

$$\eta_{ij} = CP_{ij} = \frac{max\_cost_i - ks_i^j \cdot t + 1}{max\_cost_i - min\_cost_i + 1} \quad (3)$$

Here, $min\_\ cost_i = min_{1 \le j \le m_i}\{ks_i^j c\}$, $max\_cast_i = max_{1 \le j \le m_1}\{ks_i^j \cdot c\}$. This formula ensures that the heuristic information is normalized to the interval (0, 1) and that the lower the execution cost of a web service instance is, the greater the value of its heuristic information.

Resource scheduling model for big data processing

Parameter definitions

Definition 1 Assume that the limited set of physical clusters in the current streaming big data processing platform is $N = \{N_1, N_2, ..., N_d\}$, where the resource configuration of each physical machine is $N_d =< total_d^{cpu}, total_d^{mem} >$. To determine the quantitative indicators for the combined scheduling strategy, it is necessary to quantify the resource utilization of each node. In this paper, the different computing resources of the CPU and memory are considered separately to perform scheduling and quantify the resource utilization rate on each node.

Definition 2 The node resource utilization U_d is calculated as the ratio of the actual amount of resources occupied on each node to the total amount of resources available at that node during operation. The CPU and memory resource utilization on a node are calculated using the following formulas:

$$U_d^{cpu} = \frac{\sum_{j=1}^{n} R_{dj}^{cpu}}{total_d^{cpu}}$$
$$U_d^{mem} = \frac{\sum_{j=1}^{n} R_{dj}^{mem}}{total_d^{mem}} \quad (4)$$

Here, U^{cpu}_d and U^{mem}_d represent the CPU and memory resource utilization, respectively, of the physical node N_d and $\sum_{j=1}^{n} R_{dj}^{cpu}$ and $\sum_{j=1}^{n} R_{dj}^{mem}$ represent the sums of the memory and CPU resource usage, respectively, of the computing containers running on the physical node N_d.

Scheduling operation timing

For a given computing container, one can first determine whether the computing container requires resource rescheduling. The judgement rule for this purpose is:

$$PR_{i(n+1)}^{cpu} \neq AR_{in}^{\phi u}$$
$$PR_{i(n+1)}^{mem} \neq AR_{in}^{mem} \quad (5)$$

where $PR_{i(n+1)}^{cpu}$ and $PR_{i(n+1)}^{mem}$ represent the predicted CPU and memory resources, respectively, needed for the i-th computing container in the (n + 1)-th time window and AR^{cpu}_{in} and AR^{mem}_{in} represent the CPU and memory resources, respectively, actually assigned to the i-th computing container in the n-th time window That is, as long as the actual allocated resource amount is different from the predicted amount, resource rescheduling must be performed on the computing container, and the computing container is added to the resource rescheduling queue (RSQ).

Calculation of resource increase and decrease

First, the predicted resource value $PR_{i(n+1)} = (PR_{i(n+1)}^{cpu}, PR_{i(n+1)}^{mem})$ for the i-th computing container and the actual configured resource amount $AR_{in} = (AR_{in}^{cpu}, AR_{in}^{mem})$ for

the i-th computing container in the (n + 1)-th time window are obtained; then, the resource adjustment $\Delta R_{i(n+1)}$ for container i in the (n + 1)-th time window can be calculated.

$$\begin{aligned} \Delta TR_{i(n+1)} &= < \Delta R^{cpu}_{i(n+1)}, \Delta R^{mem}_{i(n+1)} > \\ \Delta TR^{(pu)}_{i(n+1)} &= PR^{cpw}_{i(n+1)} - AR^{cpu}_{in} \\ \Delta TR^{mem}_{i(n+1)} &= PR^{mem}_{i(n+1)} - AR^{mem}_{in} \end{aligned} \tag{6}$$

Note that the predicted resource values in terms of CPU and memory for each computing container may be either smaller or greater than the current actual configured resource amount. Accordingly, when $\Delta TR^{cpu}_{i(n+1)}$ or $\Delta TR^{mem}_{i(n+1)}$ is greater than 0, this indicates a resource addition to the CPU or memory. When $\Delta TR^{cpu}_{i(n+1)}, \Delta TR^{mem}_{i(n+1)}$ is less than 0, this means that the CPU or memory resources are reduced.

Theory related to cloud workflows

Scenario model

The core business process analysis of the platform is as follows:

First, a service requester logs into the system using a legal user name and password and starts a service application in accordance with the workflow rules of the company. The application process mainly includes entering the application data, submitting the application, and waiting for the application to be accepted.

Second, the acceptor at the acceptance centre accepts the service application data information, checks the business data, accepts the service application, issues an acceptance opinion, and reviews the workflow.

Then, the dispatcher at the dispatching centre reviews the business data information, reviews the acceptance result, and distributes the event.

Finally, the dispatcher feeds the audit opinion back to the acceptor. The dispatcher distributes the event to the squad leader in accordance with the business demand. The squad leader calculates the allocation and waits for the decision-maker to issue the order, and the implementation department begins the business implementation process after receiving the instruction. After that, the result of the workflow computation is fed back to the acceptor, the acceptor summarizes the information, and the processing result is fed back to the service requester. The service requester performs the next event flow, generates a workflow information table, performs the warehousing process, and completes the workflow by sending a workflow message, which allows the information maintainer and workflow supervisor to maintain and monitor the workflow information at any time.

Role models

The roles of the entities performing a workflow can be abstracted in accordance with their functions during event processing: application requester, service requester, acceptor, dispatcher, squad leader, decision-maker and implementation department. Acceptor functions include information collection, task distribution, acceptance confirmation, programming, emergency monitoring, incident reporting and comprehensive coordination. Service requester functions include service application, information retrieval and alarm issuance. Implementation department functions include information feedback, information retrieval and command reception. Squad leader functions include information collection, information reporting, task distribution, log management, command reception, and event monitoring. Decision-maker functions include situation monitoring, program validation, and event monitoring. Scheduler functions include task signing, duty management, situation monitoring, and service auditing. Workflow monitor functions include situation monitoring and message monitoring. Business manager functions include business management, user management and personal information management. Information maintainer functions include information maintenance, backup maintenance and communication management. Application requester functions include workflow template selection, workflow template configuration, application configuration and data configuration.

Dynamic resource prediction model for big data processing

Parameter definitions

This paper introduces a sliding window function. For each application, the predicted resource usage value for the i-th computing container in the (n + 1)-th time window, W_{n+1}, can be expressed as:

$$PR_{i(n+1)} = g(R_i) \tag{7}$$

where $g(R_i)$ represents a resource usage prediction model. For all computing containers $CC = \{CC_1, CC_2, \cdots, CC_m\}$ in the streaming big data processing platform, the historical resource usage data of each computing container CC_i are obtained by monitoring each time window to form a data stream R_i with temporal properties, as defined below.

Definition 1 Physical resource usage sequence

For the i-th computing container, CC_i ($i \le m$), the corresponding resource usage in the n-th time window is R_{in}, and the resource usage sequence $R_i = \{R_{i1}, R_{i2}, \cdots,$

$R_{in}\}$ of computing container CC_i is obtained as a complete time series, where n is the number of time windows and R_{in} is the amount of resources used by the application's i-th computing container in the n-th time window. Since the resource usage includes both CPU resource usage and memory resource usage, R_{in} can be expressed as $R_{in} = \{R_{in}^{cpu}, R_{in}^{cpu}\}$.

Definition 2 Sequence of changes in resource usage

For the i-th computing container CC_i, the difference between the adjacent n-th time window and the (n-1)-th time window is expressed as the change in resource usage $\Delta R_{in} = R_{in} - R_{i(n-1)}$, from which the sequence of changes in resource usage $\Delta R_i = \{\Delta R_{i1}, \Delta R_{i2}, \ldots, \Delta R_{in}\}$ can be obtained for the computing container. Since R_i includes both CPU and memory resources, the sequence of changes in resource usage includes the sequence of changes in CPU usage ΔR_{in}^{cpu} and the sequence of changes in memory usage $\Delta R_{in}^{mem}, \Delta R_{in} = \{\Delta R_{in}^{cpu}, \Delta R_{in}^{mem}\}$, where $\Delta R_{in}^{cpu}, \Delta R_{in}^{mem}$ are calculated as follows:

$$\begin{aligned} \Delta R_{in}^{cpu} &= R_{in}^{cpu} - R_{i(n-1)}^{cpu} \\ \Delta R_{in}^{\mathrm{mem}} &= R_{in}^{\mathrm{mem}} - R_{i(n-1)}^{\mathrm{mem}} \end{aligned} \tag{8}$$

Resource prediction model based on the changes in resource usage

The predicted resource usage value for the i-th computing container in the (n + 1)-th time window is calculated from the historical CPU and memory usage sequences, as expressed below:

$$g(R_i) = f\left(R_i^{cpu}, R_i^{mem}\right) \tag{9}$$

where, as shown in Definition 1, R_i^{cpu} is the sequence of CPU resource usage from the start time of the i-th computing container to the n-th time window and R_i^{mem} is the corresponding sequence of memory resource usage. The resource usage sequence is volatile and continuous, so the CPU and memory resource usage of the i-th computing container in the n-th time window can either increase or decrease depending on the change in usage. To predict the resource usage value in the (n + 1)-th time window, the problem is converted into the following formula:

$$f\left(R_i^{cpu}, R_i^{mem}\right) = \left\{R_i^{cpu} + \Delta R_{i(n+1)}^{cpu'}, R_i^{mem} + \Delta R_{i(n+1)}^{mem'}\right\} \tag{10}$$

Since the CPU resource usage R_{in}^{cpu} and the memory resource usage R_{in}^{mem} in the n-th time window are known, the problem translates into one of finding the changes in resource usage, $\Delta R_{i(n+1)}^{cpu'}$ and $\Delta R_{i(n+1)}^{mem'}$, in the next time window.

Experiment

Test environment

The test environment for the system implemented in this paper consists of 6 physical nodes configured as shown in Table 1, one of which is the master node, four of which are compute nodes, and one of which serves as the client to simulate changing user requests. A change law in the form of a Poisson probability density is continuously sent to the Kafka application over time.

Workflow selection

For the experiments presented this paper, five workflows, which are representative examples from the field of workflow research, are selected, namely, SIPHT, LIGO, Epigenomics, Montage and CyberShake. Among these five workflows, each has a different structure, different data sources and different computational characteristics.

The SIPHT workflow is derived from the Harvard Bioinformatics Project. It represents an automated search for the sRNA-encoding genes of all bacteria in the database of the International Bioinformatics Center. The tasks in this workflow have high CPU processing power and I/O requirements. The LIGO workflow comes from the field of physics. The goal is to analyse and detect gravitational wave data, which is a CPU-intensive task that consumes considerable memory. The Epigenomics workflow is used to automate various operations in genome sequence processing and requires strong CPU processing power. The Montage workflow comes from NASA/IPAC and is an astronomical application for generating specific mosaics based on input images. Most of this workflow consists of I/O-intensive tasks that do not require much CPU processing power. CyberShake is a data-intensive workflow created by the Southern California Earthquake Center to analyse seismic hazards. It also requires considerable memory and CPU support. These five workflows are used to represent five different applications in the experiments presented in this paper. In each application, the workflow consists of 50, 100, 200, 300, 400, 500, 600, 700, 800, 900 or 1000 tasks. The parameters and run time of each workflow are determined based on real workflow logs.

Performance evaluation indicators

Forecast performance evaluation index

This paper uses the absolute error (AE) as a measure. The AE is used to measure the difference between the predicted CPU or memory resource usage and the actual usage within a single time window, as calculated by the following formula:

Table 1 Test environment configuration

Resource Type	Resource Name	Resource Configuration
Hardware	CPU	Intel(R) Core(TM) i5–3470 CPU @ 3.20 GHz * 4
Software	RAM	8 GB
	External storage	1 TB
	Internet	Gigabit Ethernet
	Operating system	Centos 6.5
	JVM	jdk 1.7.0_65
	Middleware	Zookeeper3.4.5
	Compiler environment	Maven 3.2.2

$$AE = X_{observe,n} - X_{predict,n} \tag{11}$$

In this formula, $X_{oberve,n}$ represents the observed CPU or memory resource usage of the computing container in the n-th time window, and $X_{predict,n}$ represents the predicted CPU or memory resource usage of the computing container in the n-th time window.

Overall performance evaluation index

The time-sensitive demand satisfaction rate is expressed as the ratio of the number of requests successfully processed within the deadline to the total number of requests sent during a certain period of time. The calculation method is shown below:

$$P_{qos} = \frac{K(T)}{N(T)} \tag{12}$$

where N(T) represents the total amount of data arriving within time window T and K(T) represents the amount of data successfully processed within the deadline in time window T.

The application processing response time is expressed as the average time spent successfully processing each data point per unit time. The total amount of data processed per unit time can be expressed as the number of tuples processed per unit time. This indicator represents the difference between the time the data are sent from the application to the time they are fully processed by the application and a result is returned:

$$L(T_k) = \frac{\sum_{i=0}^{n(T_k)} L_i}{n(T_k)} \tag{13}$$

where T_k represents the current k-th time window, $n(T_k)$ represents the total amount of data processed in time window T_k, and L_i represents the processing delay of the i-th data point in time window T_k.

The application resource utilization is expressed as the ratio of the amount of CPU or memory resources $R_{allocate}$ allocated to the application to the actual resource usage R_{use} in a unit time window. The resource usage of the application over the entire steady-state time period T is used. The resource usage is then averaged to represent the average resource utilization of the application. The calculation formula is shown below:

$$R(T) = \frac{1}{n}\sum_{w=1}^{n} \frac{R_{use,w}}{R_{albcate}} \tag{14}$$

This formula calculates the value of the resource usage within a time window w.

Results and discussion

Performance analysis under different data arrival intensities

In these experiments, the SequenceTest program was used for testing. The data arrival rate was divided into three groups in order from low to high. In Table 1, the horizontal header indicates the number of the parameter value test group, and the value in each column represents the data arrival rate in the corresponding group, where tps/s represents the average number of tuples sent per second. The default initial resource allocation is < 1 core, 2 GB > for each computing container. This value is the default resource configuration for the JStorm cluster. The time guarantee requirement is 400 ms, which is the average data arrival rate in the default configuration (Table 2).

In terms of the timeliness requirements, as shown in Fig. 1, compared with the JStorm system, the maximum increase is 15.26%, the lowest increase is 11.35%, and the average increase is 13.38%.

Ensuring timeliness is the primary goal of a streaming big data processing platform, and the response time is a direct indicator of whether time-sensitive requirements can be satisfied. The improvements in timeliness achieved

Table 2 Groupings in terms of data arrival intensity

Value group	Bin 1	Bin 2	Bin 3
Average arrival rate	420 tps/s	1120 tps/s	2160 tps/s

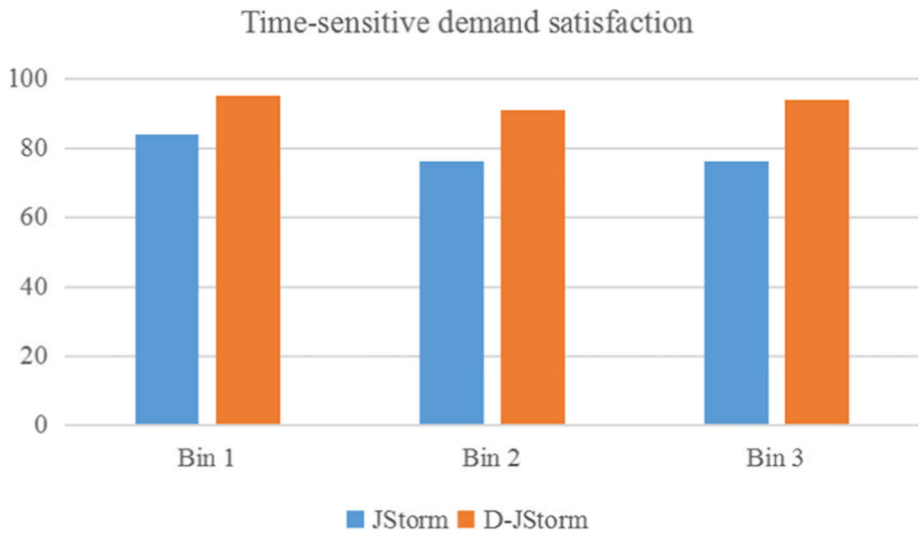
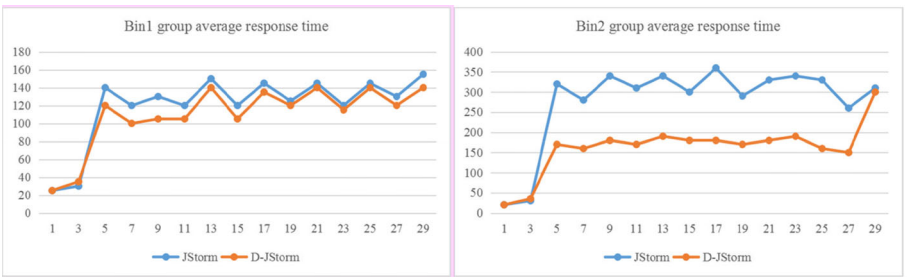


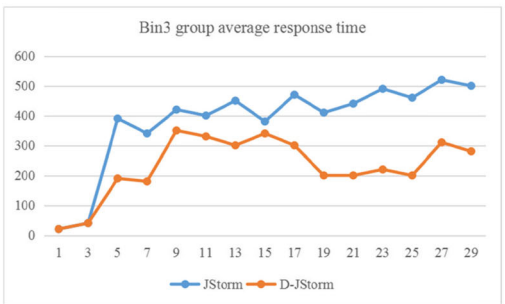
Fig. 1 Comparison of the satisfaction of time-dependent demand under different data arrival strengths

in this paper are mainly reflected in the fact that when the application load reaches a peak, the proposed platform can dynamically allocate sufficient CPU and memory resources for each computing container in advance, ensuring that all computing containers have sufficient calculation and storage capacity at the peak of the resource demand. Moreover, when the load decreases, due to the reduced demand for resources, the arriving data requests can be processed in sufficient time. Based on these two factors, the timeliness of data processing is maximized due to resource pre-feeding at peak times and resource demand satisfaction at load trough times.

In terms of response time, as shown in Fig. 2, compared with the JStorm system, the minimum reduction is 15.69%, the maximum reduction is 35.1%, and the average reduction is 26.76%.



(a) Average response time curves for the Bin 1 group (b) Average response time curves for the Bin 2 group



(c) Average response time curves for the Bin 3 group

Fig. 2 Comparison of average response time under different data arrival intensities. **a** Average response time curves for the Bin 1 group. **b** Average response time curves for the Bin 2 group. **c** Average response time curves for the Bin 3 group

The improvements in the response time compared with the original JStorm system are mainly reflected in the peak load time. In this experiment, each node was individually selected to run a computing container to avoid interactions among the computing resource and storage resources of multiple computing containers. Dynamic allocation allows a single computing container to obtain more resources for calculation at peak load times while reducing the data dwell time in the cache queue; thus, the amount of data queued in each component is reduced. The processing time is also greatly reduced, which, in turn, leads to a reduction in the overall time elapsed between when the data flow into the system and when they are completely processed.

Performance analysis under different initial resource configurations

To analyse the impact of the initial resource allocation in the D-JStorm system, the initial resource allocation configurations for different computing containers were divided into three groups, representing resource shortage, the default resource configuration and resource affluence, as shown in Table 3.

In terms of the timeliness requirements, as shown in Fig. 3, compared with the JStorm system, although the Bin 1 group shows a decrease of 4.17%, the Bin 2 and Bin 3 groups show increases of 13.5% and 11.16%, respectively, corresponding to an average increase of 6.83%.

For the Bin 1 group, the initial resource configuration <CPU, memory> corresponds to < 0.5 cores, 2 GB>, and the CPU demand is resource-intensive. Since the prediction method selected in this paper is based on the historical time window sequence, a certain level of guaranteed time-sensitive resource allocation is required among the historical values; however, the configuration of 0.5 cores keeps the resources in a state of tension, preventing the predictive component from having sufficient historical low-latency response values to determine how many resources should be allocated to achieve a given time-sensitive requirement. As a result, the system cannot obtain sufficient resources to process the data at load peaks, leading to excessive data accumulation, increased latency, and a reduced timeliness guarantee. By contrast, in the Bin 2 and Bin 3 groups, sufficient CPU resources are allocated such that sufficient historical experience can be obtained at the start of the program to guide the resource prediction process. At peak load times, it can be ensured that there are sufficient resources to process the data and reduce the amount of data waiting, thereby improving the timeliness guarantee.

Table 3 Test load configuration groupings for each compute unit

Value group	Bin 1	Bin 2	Bin 3
CPU cores	0.5	1	4
Memory (GB)	2	2	8

In terms of the response time, as shown in Fig. 4, compared with the JStorm system, except for the Bin 1 group, the response time is improved by a maximum of 58.26% and an average of 27.68%.

An increase in response time directly leads to a reduction in the timeliness guarantee. A decrease in response time compared to the JStorm system occurs when the resource configuration is < 0.5 cores, 2 GB>. According to the data analysis, memory resources are not the bottleneck of this system. Therefore, at peak load times, due to the failure to provide a timely resource supply, the tight allocation of CPU resources causes a large amount of data to accumulate in the cache queue. The processing rate of data in the computing containers is also greatly reduced, which, in turn, leads to an increase in the response time for all data. As seen from the Bin 2 and Bin 3 groups, as the initial resource allocation increases, the average response time of the system also gradually increases. This is because the sufficient initial resource configuration enables the prediction algorithm to make accurate predictions of resource requirements and response time. The historical data are used as the basis for prediction, so as the prediction accuracy gradually improves, more resources can be adjusted in time to better support more computing containers; consequently, the processing rate of the data is increased, thereby reducing the response time.

Analysis of combined resource management based on the ant Colony algorithm

Figure 5 shows the results obtained with a population size of 1000 as the number of web service candidates per service class varies from 1 to 50. The coincidence rate between the MOACO algorithm's optimal fitness and the actual optimal fitness (the proportion of instances with the same fitness value) is 97%. The optimal fitness curve of the algorithm basically coincides with the actual optimal fitness curve. The average number of generations before termination is 80.5, and the best fitness is 0.868. The coincidence rate between the optimal fitness and the actual optimal fitness for the CCA algorithm is 87%, the average number of generations before termination is 90.56, and the optimal fitness is 0.8006. The coincidence rate between the optimal fitness and the actual optimal fitness for the GA algorithm is 79%, the average number of generations before termination is 110.43, and the optimal fitness is 0.647. The experimental results show that with an increase in the number of web service candidates per service class, the maximum

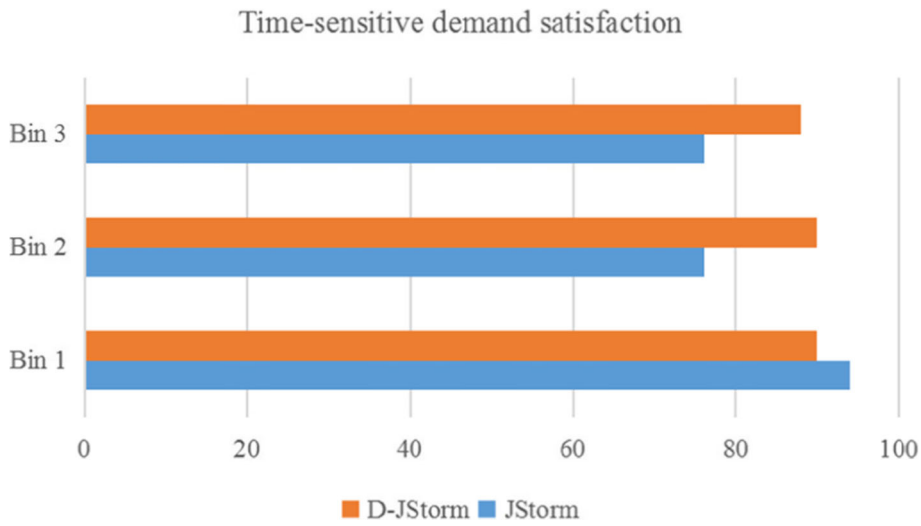
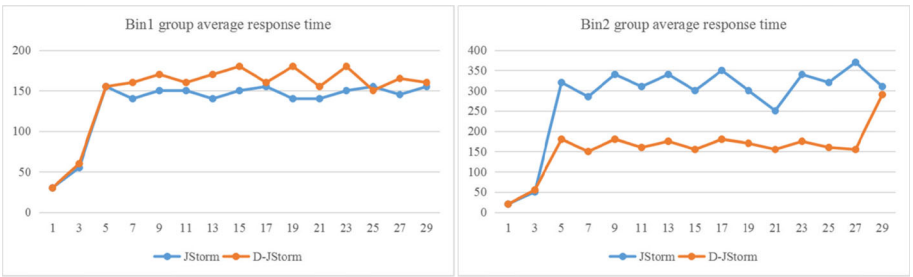


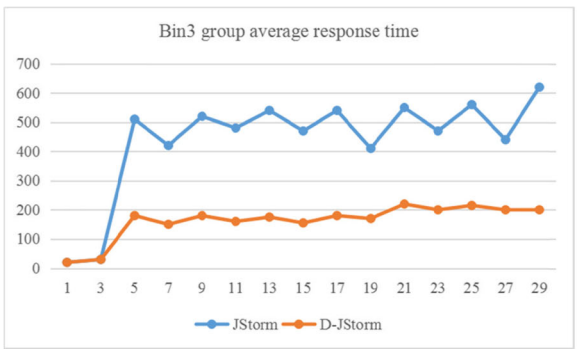
Fig. 3 Comparison of time-based demand satisfaction degree under different initial resource configurations

fitness generally increases, reflecting the dynamic combination of web services. As the number of web services published by the service provider increases, the WS core process can better select the optimal services.

An adaptation-aware policy can improve the execution success rate of cloud workflows. Due to the dynamic nature of the cloud computing environment, web services that execute cloud workflow tasks may fail to run, which will affect the execution success rate of the entire cloud workflow. In this experiment, six cloud workflow instances with 50, 100, 150, 200, 250, and 300 tasks were established; each group of cloud workflows was run 50 times, and the average execution success rate was taken. From the experimental results in Fig. 6, we can see that the adaptation-aware strategy can significantly improve the execution success rate of a cloud workflow. When



(a) Average response time curves for the Bin 1 group (b) Average response time curves for the Bin 2 group



(c) Average response time curves for the Bin 3 group

Fig. 4 Average response time curve in different initial configurations. **a** Average response time curves for the Bin 1 group. **b** Average response time curves for the Bin 2 group. **c** Average response time curves for the Bin 3 group

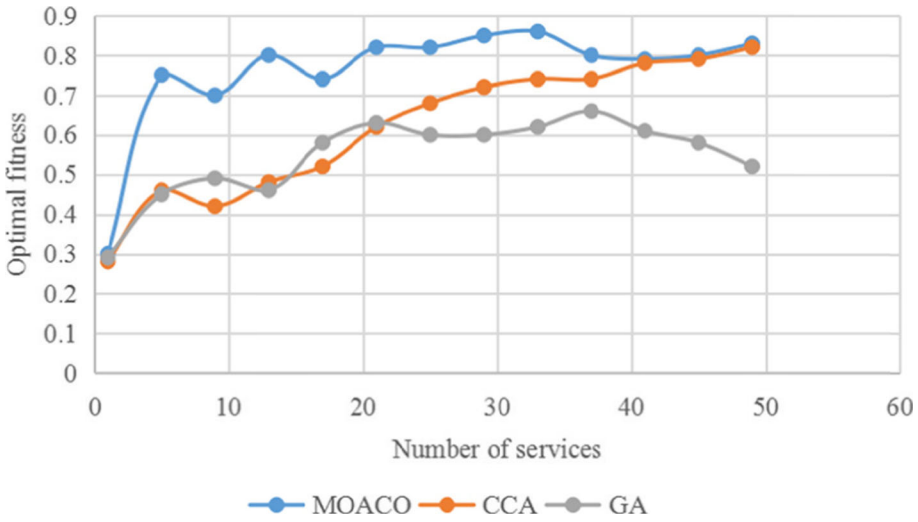


Fig. 5 Optimal fitness at different service scales

the number of tasks is 50, the execution success rate with the non-adaptation-aware strategy is 92%, and the execution success rate with the adaptation-aware strategy is 98%. When the number of tasks is 150, the execution success rate with the non-adaptation-aware strategy is 89%, and the execution success rate with the adaptation-aware strategy is 95%. When the number of tasks is 300, the execution success rate with the non-adaptation-aware strategy is 82%, and the execution success rate with the adaptation-aware strategy is 88%. As the number of tasks increases, the execution success rate decreases for both strategies. This is because the more tasks there are, the more web services are needed, and the greater the interaction among services, hindering successful execution.

Cloud workflow task scheduling analysis

Figure 7 shows the optimization rates for the overall completion time in 10 different experimental scenarios. As can be seen from Fig. 7, when the workflow size is small, the completion time optimization rates of GA, ACO, and PSO are similar, but they are all low. For example, in the first set of experiments involving a 50-task cloud workflow, the completion time optimization rate was approximately 5% for ACO, approximately 5.6% for PSO, and approximately 5.8% for GA. As the scale of the workflow increases, the GA performance remains relatively stable, with a slight downward trend, and the PSO performance initially shows an upward trend and then falls sharply, whereas the ACO performance generally increases with the workflow scale. For example, in the

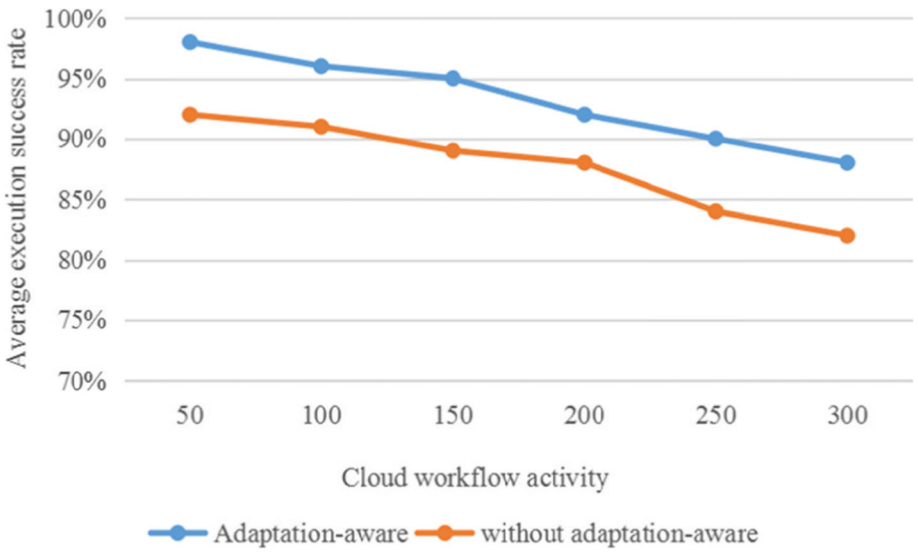


Fig. 6 Effect of adjusting the perception policy on execution success rate

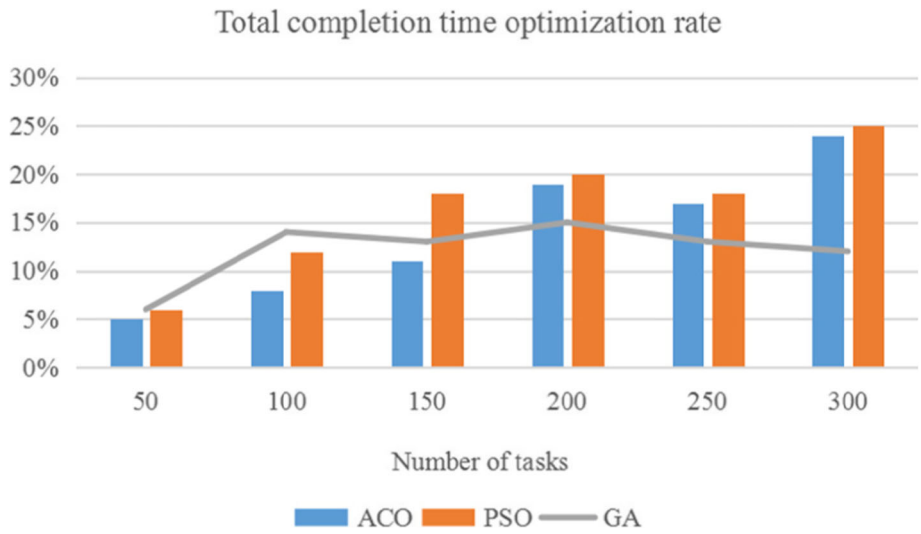


Fig. 7 Optimization rate of overall completion time

last set of experiments involving a 300-task workflow, the completion time optimization rate of the ACO algorithm was 24.4%, while that of GA was 11.8% and that of PSO was only 3.88%. It is worth noting that PSO achieved better performance than the other two algorithms as the workflow scale increased from 150 to 200 tasks. For example, when the number of tasks was 180, the completion time optimization rate of the PSO algorithm was 22.2%, while that of GA was 13.8% and that of ACO was 15.4%.

A unique phenomenon observed here is that the ACO algorithm performs better when the number of tasks is greater than 200. This shows that the ACO algorithm is more effective in solving large-scale discrete multi-constrained optimization problems. This is likely because the ACO algorithm constructs an efficient solution in a task-by-task manner, whereas the PSO algorithm and the GA algorithm randomly search for solutions in the solution space of the problem. Therefore, the solutions generated by the ACO algorithm can satisfy all constraints, whereas the GA algorithm and the PSO algorithm cannot guarantee that the generated solutions are feasible. Because of this, the ACO algorithm can maintain higher performance as the workflow size increases, whereas the GA algorithm and the PSO algorithm exhibit premature convergence due to the limited representation space.

Conclusions

To find ways to effectively manage and schedule intelligent cloud workflows, this article has studied big data processing from various aspects and reached the following conclusions:

(1) Based on the existing open source platform JStorm for real-time big data processing, a dynamic resource scheduling system called D-JStorm is designed and implemented in this paper. The performance analysis of the D-JStorm system shows that compared with the original JStorm system, the response time is reduced by a maximum of 58.26% and an average of 23.18%, the CPU resource utilization rate is increased by a maximum of 17.96% and an average of 11.39%, and the memory resource utilization rate is increased by a maximum of 88.7% and an average of 71.16%.

(2) The average number of generations before termination is 116.46 for GA, 103.9 for CCA, and 89.62 for MOACO. It can be seen that the convergence of the GA algorithm is the worst, while the convergence of the MOACO algorithm is the best, whereas the convergence of the CCA algorithm lies between the other two. When the number of web services exceeds 43, the growth in the number of generations before termination is significantly accelerated for the GA and CCA algorithms, with the growth rate for the GA algorithm being the largest. By contrast, the number of generations of the MOACO algorithm is slow to grow. This shows that the MOACO algorithm is more suitable for optimizing large-scale dynamic web service discovery processes. Overall, the overall performance of the MOACO and CCA algorithms in optimizing the dynamic combination of web services is better than that of the GA algorithm, and the average performance of the MOACO algorithm is better than that of the CCA algorithm.

(3) A cloud workflow scheduling strategy based on an intelligent algorithm and an adaptive cloud service resource combination strategy is proposed to realize two-layer scheduling of cloud workflow tasks. We have studied three representative intelligent algorithms (ACO, PSO and GA) and improved them for scheduling optimization. Based on the nature of these three algorithms, we tested the performance for different numbers of workflow tasks based on these intelligent algorithms. It can be clearly seen that although the optimal values of the different algorithms vary greatly among different test cases in the same scenario, the optimal solution curves are substantially consistent with the trend of the mean curve.

Acknowledgements
The authors thank the editor and anonymous reviewers for their helpful comments and valuable suggestions. I would like to acknowledge all our team members.

Authors' contributions
All authors take part in the discussion of the work described in this paper. The authors read and approved the final manuscript.

Authors' information
Yannian Hu was born in Gaomi, Shandong P.R. China, in 1976. He received the bachelor's degree from Qingdao University. Now, he works in Big Data Buro of Weifang. His research interests include computational intelligence, information security and big data analysis etc..
E-mail: hu_yannian@163.com
Hui Wang was born in Gaomi, Shandong P.R. China, in 1978. She received a master's degree from Guangxi Normal University. Now, she works in Weifang University. Her research interests include big data analysis, ideological and political educationand and communication theory etc..
E-mail: conglinwh@wfu.edu.cn
Wenge Ma was born in Yuncheng County, Shandong Province, China in 1987. He received a master's degree from Beijing Technology and Business University. Now he works in Shandong Modern Education Science Research Institute Science. Research interests are AI education and big data technology.
E-mail: sdjkyb@126.com

Availability of data and materials
All the data and materials in this article are real and available.

Ethics approval and consent to participate
Approved.

Consent for publication
Approved.

Competing interests
These no potential competing interests in our paper. And all authors have seen the manuscript and approved to submit to your journal. We confirm that the content of the manuscript has not been published or submitted for publication elsewhere.

Author details
[1]Big Data Buro of Weifang, Weifang 261061, Shandong, China. [2]Weifang University, Weifang 261061, Shandong, China. [3]Shandong Provincial Institute of Modern Educational Science, Weifang 261061, Shandong, China.

Received: 9 December 2019 Accepted: 22 May 2020
Published online: 21 July 2020

References

1. Tatlow PJ, Piccolo SR (2016) A cloud-based workflow to quantify transcript-expression levels in public cancer compendia. Sci Rep 6(1):39259
2. Yu X, Joshi P, Xu J, Jin G, Zhang H, Jiang G (2016) Cloudseer: workflow monitoring of cloud infrastructures via interleaved logs. Acm Sigarch Comput Arch News 44(2):489–502
3. Sadhasivam N, Thangaraj P (2016) Design of an improved pso algorithm for workflow scheduling in cloud computing environment. Intell Automation Soft Comput 23(3):1–8
4. Xie Y, Tianta HE, Qianyun NI, Hanqing WU (2017) Scheduling for improving the energy efficiency of cloud workflow execution. Syst Eng Theory Pract 37(4):1056–1071
5. Cartwright R (2018) An internet of things architecture for cloud-fit professional media workflow. Smpte Motion Imaging J 127(5):14–25
6. Buyya R, Gill SS (2018) Sustainable cloud computing: foundations and future directions. Cutter IT J 21(6):1–9
7. Chen C, Chen D, Yan YN, Zhang GF, Zhou QG, Zhou R (2018) Integration of numerical model and cloud computing. Futur Gener Comput Syst 79(3):396–407
8. Wang Y, Li J, Wang HH (2019) Cluster and cloud computing framework for scientific metrology in flow control. Clust Comput 22(1):1–10
9. Shen J, Member IEEE, Zhou T, Chen X (2018) Anonymous and traceable group data sharing in cloud computing. IEEE Trans Inf Forensic Secur 13(4):912–925
10. Liu X-F, Student Member, IEEE, Zhan Z-H, Member (2018) An energy efficient ant colony system for virtual machine placement in cloud computing. IEEE Trans Evol Comput 22(1):113–128
11. Luo J, Wu M, Gopukumar D, Zhao Y (2016) Big data application in biomedical research and health care: a literature review. Biomed Inform Insights 8(8):1–10
12. Wu T, Dou W, Fan W, Tang S, Hu C, Chen J (2016) A deployment optimization scheme over multimedia big data for large-scale media streaming application. Acm Trans Multimedia Comput Commun Appl 12(5s):1–23
13. Cucudumitrescu C, Constantin S (2017) Extraction of regions with similar temporal evolution using earth observation big data. Application to water turbidity dynamics. Remote Sensing Lett 8(7):627–636
14. Ulfarsson MO, Palsson F, Sigurdsson J, Sveinsson JR (2016) Classification of big data with application to imaging genetics. Proc IEEE 104(11):2137–2154
15. Xue Y, Xu L, Jie Y, Zhang G (2017) A hot event influence scope assessment method in cyber-physical space for big data application. Intell Automation Soft Comput 24(1):1–9
16. Zheng T, Chen G, Wang X, Chen C, Luo S (2019) Real-time intelligent big data processing: technology, platform, and applications. Sci China Inf Sci 62(8):82101
17. Zheng B, Tang X, Zhang Z, Zong B (2019) Modeling of continuous cross-flow microfiltration process in an airlift external-loop slurry reactor. China Petroleum Process Petrochemical Technol 21(1):117–122
18. Tian W, Shao Y, Yanzhu H (2018) 3d damage identification of soil rock mixture based on image processing technology. Iop Conference 394(5): 052003
19. Papias S, Masson M, Pelletant S, Prost-Boucle S, Boutin C (2018) In situ continuous monitoring of nitrogen with ion-selective electrodes in a constructed wetland receiving treated wastewater: an operating protocol to obtain reliable data. Water Sci Technol 77(6):1706–1713
20. Larras F, Billoir E, Baillard V, Siberchicot A, Delignette-Muller ML (2018) Dromics: a turnkey tool to support the use of the dose–response framework for omics data in ecological risk assessment. Environ Sci Technol 52(24): 14461–14468
21. Hussein MK, Mousa MH, Alqarni MA (2019) A placement architecture for a container as a service (caas) in a cloud environment. J Cloud Comput 8(1):7
22. Pääkkönen P, Heikkinen A, Aihkisalo T (2019) Online architecture for predicting live video transcoding resources. J Cloud Comput 8(1):9
23. Chen X, Chen S, Zeng X, Zheng X, Rong C (2017) Framework for context-aware computation offloading in mobile cloud computing. J Cloud Comput 6(1):1

Publisher's Note