Accepted Manuscript

Dynamic resource allocation strategy for latency-critical and computation-intensive applications in cloud-edge environment

Hengliang Tang, Chunlin Li, Jingpan Bai, JiangHang Tang, Youlong Luo

PII: S0140-3664(18)30468-7
DOI: https://doi.org/10.1016/j.comcom.2018.11.011
Reference: COMCOM 5802

To appear in: *Computer Communications*

Received date : 12 May 2018
Revised date : 27 November 2018
Accepted date : 30 November 2018

Please cite this article as: H. Tang, C. Li, J. Bai et al., Dynamic resource allocation strategy for latency-critical and computation-intensive applications in cloud-edge environment, *Computer Communications* (2018), https://doi.org/10.1016/j.comcom.2018.11.011

Dynamic resource allocation strategy for latency-critical and computation-intensive applications in cloud-edge environment

Hengliang Tang[1]， Chunlin Li[1,2,3 *], Jingpan Bai [2], JiangHang Tang[2], Youlong Luo[2]

[1] School of Information, Beijing Wuzi University, Beijing 101149, China

[2] School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430063, China

[3] Hebei Engineering Technology Research Center for IOT Data acquisition & ProcessingNorth China Institute of Science and Technology, Hebei, China

* Corresponding author

Abstract: Edge computing is more and more popular due to its low latency and bandwidth-efficient services. Edge computing is mainly applied to the latency-critical and computation-intensive application. However, there are several challenges to the improvement on the quality of service in edge computing environment. For instance, the reduction of server latency, the network transmission efficiency, etc. In this paper, we propose the dynamic resource allocation algorithm for cloud-edge environment. The dynamic resource allocation algorithm consists of the resource scheduling algorithm and the resource matching algorithm. In the resource scheduling algorithm, a resource scheduling problem can be obtained according to the stored penalty of scheduling contents, the value of scheduling contents and the transmission cost of scheduling contents. Then, tabu search algorithm is applied to find the optimal solution to the resource scheduling problem. Furthermore, the resources are scheduled into the edge servers from cloud datacenter with the optimal solution. In the resource matching algorithm, an optimization problem of the resource matching is built with respect to the resource location, the task priorities and the network transmission cost. For addressing this problem, the optimal problem is converted to an optimal matching problem of the weighted bipartite graph. Moreover, an optimal matching problem of the weighted complete bipartite graph is created by adding the spurious containers. Then, the optimal strategy of the resource matching for tasks on the edge servers is achieved. Finally, the performance of the proposed algorithms and some typical resource allocation algorithms is evaluated via extensive experiments. The results indicate that proposed algorithms can effectively reduce network delay and enhance QoS.

Keyword: Resource Allocation, Cloud-Edge Environment, Resource Scheduling, Resource Matching

Ⅰ Introduction

Over the last decade, cloud computing is regarded as a new paradigm of computing to improve network efficiency and satisfy the internet requirement with connected devices increasing by moving computation, control and data storage into the cloud [1]. However, cloud computing faces some challenges to some more stringent performance, such as latency and bandwidth, which are required by many application services. Edge computing as a new trend in computing is presented to increase infrastructure efficiency with low-latency and bandwidth-efficient services [2]. In edge computing environment, tens of billions of mobile devices are deployed on network edge and the computation is conducted near the end user (EU), whose processor speeds are increasing exponentially. Edge computing has many variants. For example, cloudlet [3], fog computing [4], mobile edge computing [5], etc.. Edge computing with the advantage of low-latency and bandwidth-efficient services has been focused more and more. Edge computing is suitable for many latency-critical and computation-intensive application scenarios. For instance, face recognition [6], augmented reality [6], speech recognition [7], swarms of

unmanned aerial vehicles [8], intelligent traffic [9], smart cities [10], automatic driving [11], interactive online gaming [11], car navigation system [12], etc.. However, there are several challenges to improvement on the quality of service (QoS) in edge computing environment. For instance, the reduction of the server latency, the network transmission efficiency, etc.

Data caching scheme has been widely applied to various application scenarios, such as face recognition [6], augmented reality [6], speech recognition [7] and so on, for reducing the network delay and enhancing user experience. Cache is mainly used to schedule the popular contents into the corresponding edge server (ES). In other words, the popular contents are placed close to the end users, so that the acquisition latency of required resources is reduced and the heavy overhead burden of network backhaul is relived. Optimal resource allocation methods are very important to data-intensive computing with cloud-edge environment, such as swarms of unmanned aerial vehicles [8], intelligent traffic [9], smart cities [10], etc., for reducing server delay and improving QoS. Optimal resource allocation methods can orchestrate the relationships between computation resources, tasks, and other attributes to reduce the distance between tasks and required resources, decrease the network overhead for transmitting required resources and enhance the user experiences. The purpose of the resource allocation is to improve the performance of task scheduling strategies.

In this paper, we propose the dynamic resource allocation algorithm for cloud-edge environment. The dynamic resource allocation algorithm consists of the resource scheduling algorithm and the resource matching algorithm on edge servers. The resource scheduling algorithm is the base of the resource matching algorithm. Fig. 1 shows the overview of researches in this paper.

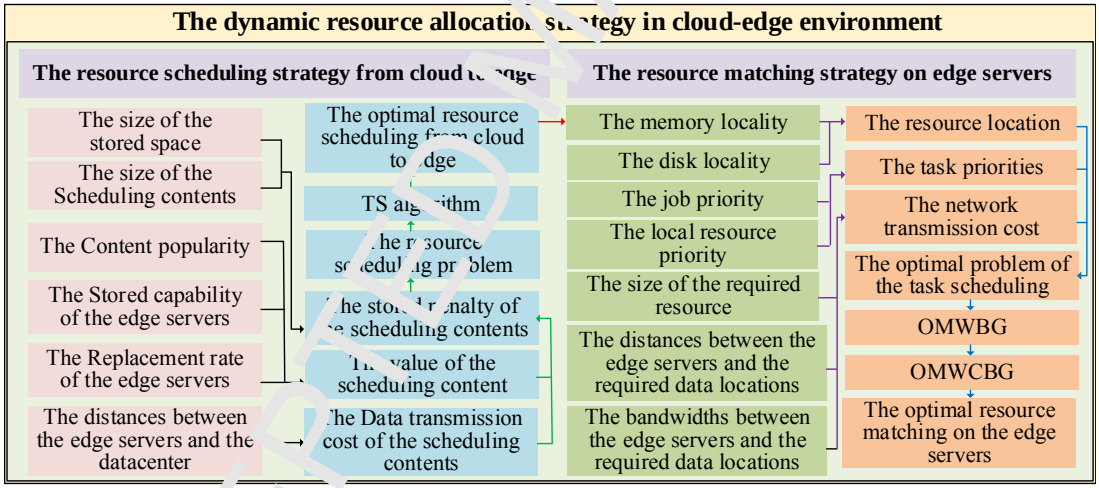

Fig. 1 The overview of researches in this paper

In the resource scheduling strategy from cloud to edge, a resource scheduling problem can be obtained according to the stored penalty of scheduling contents, the value of scheduling contents and the transmission cost of scheduling contents. The stored penalty of a scheduling content is determined by the stored space of ES and the size of scheduling content. When stored space of ES is enough for the scheduling resources, there exists no stored penalty. Otherwise, ES needs to release some contents to make room for storing the scheduling content. In this case, the stored penalty is defined as the transmission cost of the released contents from DC to ES. The value of a scheduling content consists of the content popularity, the stored capability of ES and the replacement rate of ES so that the most popular contents are scheduled into the ES with larger stored capability and with smaller replacement rate, which can hold the contents for a while to improve QoS. The transmission cost of a scheduling content is

computed by the distances between the ESs and DC. Then, tabu search (TS) is applied to the optimal solution to the optimal problem. Furthermore, the cache data are cached into ESs with the optimal solution.

In the resource matching strategy on edge servers, a resource matching model is proposed by considering the resource location, the task priorities and the network transmission cost. The resource location includes memory locality and disk locality. The memory locality implies that the tasks and corresponding resources are in the same ES and the resources are stored into the memory of the ES. The disk locality indicates that the tasks and corresponding resources are in the same ES. The resource location is used to build the relationship between tasks and containers in terms of similarity, where a container is taken as a set of the computation resources. The task priorities consist of job priorities and local resource priorities, which make sure the tasks with high job priorities and with corresponding local resources have the priority process chance. The network transmission cost is calculated by the size of the required resources, the distances and bandwidths between the ES and the locations that require the resources, such as local memory, local disk, neighbor ES or remote DC, to evaluate the minimum transmission cost of the required resources from the location that stores the corresponding resources to the location where the tasks wait for the resources. For addressing the resource matching problem, the optimal model is converted to an optimal matching problem of the weighted bipartite graph (OMWBG). Moreover, an optimal matching problem of the weighted complete bipartite graph (OMWCBG) is created by adding the spurious containers. Finally, the optimal resource matching strategy for tasks on the edge servers (ESs) is achieved.

The main contributions to this paper are summarized as follows:

- The stored penalty of a scheduling content is determined by the stored space of ES and the size of scheduling content. When the stored space of ES is enough for the scheduling data, there exists no stored penalty. Otherwise, ES needs to release some contents to make room for storing the scheduling content. In this case, the stored penalty is defined as the transmission cost of the released contents from DC to ES.
- A weighted complete bipartite graph is created to formulate the problem of resource matching. During each heartbeat, the one-to-one match relationships between containers and tasks are built. When the number of containers is less than the number of tasks, the corresponding number of spurious containers is added to satisfy the one-to-one match.
- The performance of the proposed algorithms and some typical resource allocation algorithms is evaluated via extensive experiments. The results indicate that proposed algorithms can effectively reduce network delay and enhance QoS.

The remainder of the paper is organized as follows. Section Ⅱ reviews the related work. The resource allocation problem for cloud-edge environment is formulated in Section Ⅲ. Section Ⅳ depicts the resource allocation algorithm. Extensive experiments are conducted to evaluate the performance of the proposed resource allocation algorithm in Section Ⅴ. Finally, the conclusions and future works are shown in section Ⅵ.

Ⅱ Related Work

In this section, the state-of-the-art resource allocation policies are analyzed as follows.

Data caching scheme has been widely applied to various scenarios, such as face recognition [6], augmented reality [6], speech recognition [7] and so on, for reducing the network delay and enhancing user experience. Data caching scheme is widely applied to optimize the resource allocation and moves the

popular contents into the edge servers, i.e., the popular contents are placed close to end users, so that the acquisition latency of required contents is reduced and the heavy overhead burden of network backhaul is relived. There is a tremendous amount of literature on cache technologies.

Content popularity is the most important factor in cache replacement policies. For 5G wireless networks, a big data-enabled architecture was designed, in which the content popularity was estimated by machine learning based on the vast amount of historical data [13]. The content popularity predictions were optimally exploited to derive the cache update strategy for minimizing the traffic at the core network [14].

Furthermore, it will achieve better QoS that content popularity combines with other attributes in resource allocation. A competitive caching scheme was studied to enhance the quality of experience of their customers on the move, where contents were stored at a given price set by the mobile network operator. The price consisted of the content popularity, the spatial distribution of small cells, the price for cache memory reservation and the effect of competing content under multi-tenancy [15]. A cognitive caching approach for the future fog was proposed, in which the data was assigned a value by considering the age of the data, the popularity of on-demand requests, the delay to receive the requested information and data fidelity. The most valuable ones were cached to edge servers (ESs) [16]. A fitness function consisted of the access probability, the cached document size and the validation time for each cached item was designed to determine removed some items in the cache [17].

Moreover, some researchers pay attention to where the resources will be scheduled [49-52]. A fairness cost function was developed in edge environment to improve data caching fairness [18]. An improved cache replacement strategy for online videos, which set multi-level label indexes for the video contents in the cache list of the content delivery networks node, was discussed to reduce server load and latency [19]. The centralized and distributed transmission aware cache placement strategies were developed by considering the flexible physical-layer transmission schemes and the diverse content preferences of different users in fog radio access networks to minimize users' average download delay subjecting to the storage capacity constraints [20].

However, due to storage capacity constraints of ESs, the contents stored in single ES cannot satisfy the increasing user requirements. So the collaborative resource scheduling strategy receives more attentions. A collaborative caching and processing strategy for on-demand video streaming in mobile edge computing networks was presented by considering the transcoding relationships between different versions to enhance the widely used adaptive bitrate streaming technology [21]. A single cellular network, where the requested contents of mobile users were stored in nearby peer mobile devices and were served by device-to-device communications, was studied by minimizing the distance between two actively offloaded mobile user receivers [22]. Using caches to create coded-multicasting opportunities was a much more significant gain for users with different demands. These coded-multicasting opportunities were enabled by content overlap between the various caches in the network, which was created by a central coordinating server to achieve a rate close to the optimal centralized scheme [23]. A cache placement strategy was introduced by analyzing the access cost of placing a set of object copies in the routing path [24]. A novel content access scheme named Crowd-Cache was proposed, which enabled mobile networking in proximity by exploiting the transient co-location of devices, the epidemic nature of content popularity and the capabilities of smart mobile devices. Crowd-Cache provided mobile users access to popular content cheaply with low latency while improving the overall quality of experience [25]. An information-centric-networking-based resource scheduling approach considering both the mobility of users and the popularity of videos was studied to reduce network traffic and video retrieval delay [26].

Moreover, some scholars propose specialized containers for applications. The edge was taken as a specialized container for recognition applications and formulate the expected latency. A system that used the model along with novel optimizations to minimize latency by adaptively balancing the load between the edge and the cloud, by leveraging spatiotemporal locality of requests, using offline analysis of applications, and online estimates of network conditions [27]. The edge-servers were regarded as specialized containers for compute-intensive recognition applications to decrease recognition algorithms' computation time and improve accuracy [28]. The distributed mobile device caching was applied to randomly store the popular data contents, by which a mobile user can obtain the requested files from other mobile users through device-to-device communication instead of the remote data-source provider for reducing the duplicate data traffics in the core network [29].

In the literature for the resource scheduling policies mentioned above, the researchers focus on the scheduling content selection or the location selection that storing the scheduling contents. Most of the resource scheduling policies pay few attentions to where the scheduling contents are stored. The ones on location selection although consider the combination between content popularity and stored location, few scholars are interested in the contents replaced by scheduling contents. For instance, some contents that have been stored into ESs are replaced by new scheduling contents before they are accessed, which will lead to more replacement cost. For addressing this problem, a penalty function of the scheduling contents is introduced to determine where the scheduling contents will be stored in. Meanwhile, the proposed resource scheduling policy also considers the value of the scheduling contents and the transmission cost of the scheduling contents.

Optimal scheduling methods are very important to data-intensive computing with cloud-edge environment, such as swarms of unmanned aerial vehicles [8], intelligent traffic [9], smart cities [10], etc. It can orchestrate the relationships between computation resources, tasks and other attributes to reduce server delay and improve QoS. For instance, a novel scheduling mechanism was designed by considering relationships between data nodes, computation nodes and tasks to optimize access latencies while maintaining the benefit of low storage cost [30].

Caching has a large impact on optimal scheduling, which is usually applied to improve local disk access times while providing data from the main memory without disk accesses. A resource matching method for cloud-edge environment was proposed to improve the overall job execution time for various workload types and data sizes, in which the resources were scheduled into the operating system's buffer and the tasks were assigned to nodes with the scheduling contents [31]. In mobile edge computing environment, the computation offloading decision, resource allocation and content caching strategy were formulated as a utility function to improve QoS [32]. Similarly, a virtual resource allocation strategy, which consisted of virtualization, caching and computing, was discussed to reduce the computational complexity and signaling overhead in mobile edge computing environment [33].

The purpose of caching is to guarantee data locality, which is a major factor of optimal scheduling methods to reduce data movement and thus execution time, with increasing the possibility to meet deadlines [34]. A data-aware scheduling policy was presented by taking into the workload and the resource features. Meanwhile, the impact of data locality on the performance of a SaaS cloud was investigated via simulation, in which real-time, data-intensive bags-of-tasks were scheduled dynamically to effectively exploit data locality [35]. A novel scheduling algorithm, which was used to minimize the number of transfers by taking advantage of data caching and file locality, was presented to reduce cost and meet deadline constraints on infrastructure as a service cloud [36].

However, there is a conflict between fairness in scheduling and data locality, which has been described via an experiment of a 600-node Hadoop cluster of Facebook [37]. For addressing this problem, a simple algorithm named as delay scheduling was proposed, where if the job that should be scheduled next according to fairness could not launch a local task, it must wait for a while, letting other jobs launch tasks instead [37]. Furthermore, fairness and data locality were integrated into a unified algorithm to easily adjust the tradeoffs between data locality and fairness [38].

Some researchers propose task scheduling from other angles. In mobile edge computing environment, a task scheduling strategy was presented by considering the queuing state of the task buffers, the execution state of the local processing unit, as well as the state of the transmission unit to improve the quality of computation experience [39]. The optimal schedules tolerant to out-of-date network knowledge were generated asymptotically, thereby relieving stringent requirements on feedbacks in mobile edge computing environment [40]. Pipeline Improvement Support with Critical Chain Estimation Scheduling was presented to provide better support for multi-job applications, where a job dependency DAG for currently running jobs was developed by considering their input and output directories to facilitate the data pipeline between the output phase of an upstream job and the map phase of a downstream job. This offered a new execution overlapping between dependent jobs in MapReduce which effectively reduced the application runtime [41].

In the literature on resource allocation methods, the main impact factors can be summarized as the locations of scheduling contents, the task locations and the bandwidth states. In cloud-edge environment, jointly addressing these issues together is significantly effective against reducing computation time and improving QoS. Thereby, a dynamic resource allocation strategy with respect to the resource locations, the disk locality, the remote servers, as well as the bandwidth states is presented to provide low-latency mobile services for latency-critical and computation intensive applications in cloud-edge environment.

Ⅲ Problem Formulation

In this section, a resource scheduling strategy from cloud to edge, which consists of the stored penalty of scheduling contents, the value of scheduling contents and the transmission cost of scheduling contents, is proposed for cloud-edge environment. Furthermore, based on the resource location, a resource matching method on edge servers is presented to reduce the response latency and improve QoS in cloud-edge environment. Table 1 summarizes the main notations that will be used in this paper.

Table 1 Summary of notations

Notations	Definition
QoS	Quality of service
RS	Resource scheduling
RM	Resource matching
TS	Tabu search
ES	Edge server
EU	Edge user
WBG	Weighted bipartite graph
WCBG	Weighted complete bipartite graph
AP	Access point
EO	Edge orchestrator

DC	Datacenter
P_{ij}	The stored penalty of content i on ES j
D_j	The available space for storing the scheduling content i on ES j
B_i	The set of the replaced contents for scheduling content i
b_k	The size of kth replaced content in B_i
$band_j$	The bandwidth between DC and ES j
V_{ij}	The value of scheduling content i from DC to ES j
H_i	the popularity of ith content
C_j	the stored capability of jth ES
R_j	the replacement rate of jth ES
S_{ij}	the transmission cost scheduling content i from DC to ES j
$h(j,j')$	the network distance between ES j and j'
$Sim(h_z,u_r)$	The similarity between task h_z and container u_r
$y(h_z)$	The priority of task h_z
$e(h_z,u_r)$	The network transmission cost
$\phi(h_z,u_r)$	the weight between task h_z and container u_r

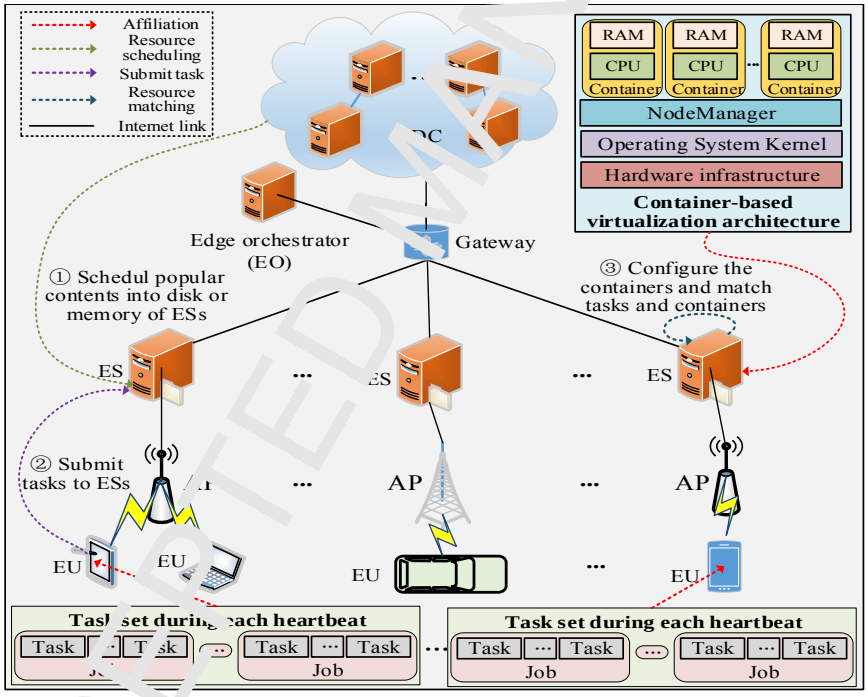

Fig. The architecture of dynamic resource allocation strategy in cloud-edge environment

Fig. 2 depicts the architecture of the dynamic resource allocation algorithm in cloud-edge environment. In this paper, a periodic scheduling mechanism is adopted in each ES. Each period is called a heartbeat. During each heartbeat, the resources from DC are chosen and scheduled to the memory or disk in each ES. Then, each end user (EU) submits his/her jobs to the corresponding ES by the access point (AP). Each job consists of multiple tasks. The jobs are submitted to the edge servers during each heartbeat, i.e., the tasks are submitted to the edge servers during each heartbeat. The tasks are indivisible, independent and non-preemptible. After the jobs are submitted, each ES will configure the corresponding containers for the tasks. Each ES can launch multi containers. Each container consists of corresponding RAM and CPU for the corresponding task. In addition, it is worthy of note that each container only

processes a task at once. Finally, each ES matches its containers and tasks for responding to its EUs. It is noteworthy that the containers and the tasks are matched again at the beginning of each heartbeat.

A. Resource scheduling from cloud to edge

In this section, a resource scheduling strategy from cloud to edge (RSSFCE) is presented by considering the stored penalty of scheduling contents, the value of scheduling contents and the transmission cost of scheduling contents. The stored penalty of a scheduling content is determined by the stored space of ES and the size of scheduling content. When stored space of ES is enough for the scheduling data, there exists no stored penalty. Otherwise, ES needs to release some contents to make room for storing the scheduling content. In this case, the stored penalty is defined as the transmission cost of the released contents from DC to ES. The value of a scheduling content consists of the content popularity, the stored capability of ES and the replacement rate of ES such that the most popular contents are scheduled into the ES with larger stored capability and with smaller replacement rate, which can hold the contents for a while to improve QoS. The transmission cost of a scheduling content is computed by the distances between the ESs and DC.

The function for the stored penalty of content i on ES j is defined as P_{ij}, which takes the form

$$P_{ij} = \begin{cases} 0, d_i \le D_j \\ \sum_{k=1}^{|B_i|} b_k / band_j, otherwise \end{cases}, \quad (1)$$

where d_i denotes the size of content i. D_j is taken as the available space for storing the scheduling contents on ES j. A set of the replaced contents for scheduling content i is defined by B_i. $|B_i|$ is the size of the set. The size of kth replaced content in B_i is taken as b_k. $band_j$ is the bandwidth between DC and ES j. When the stored space of ES is enough for storing the scheduling content, there is no stored penalty. Otherwise, ES need releases some contents to make room for storing the scheduling content. In this case, the stored penalty is defined as the transmission cost of the released contents from DC to ES.

The value V_{ij} of scheduling content i on ES j takes the form

$$V_{ij} = H_i C_j R_j^{-1}, \quad (2)$$

where H_i is the popularity of ith content. C_j denotes the stored capability of jth ES and can be computed by the product of the availability ratio of stored space on ES j, the availability ratio of CPU and the availability ratio of memory. These three physical performances can be traced by workload tracer. R_j is taken as the replacement rate of jth ES. It has been reported that the content popularity follows a Zipf distribution [42]. Thus,

$$H_i = \frac{i^{-\gamma}}{\sum_{i'=1}^{n} (i')^{-\gamma}}, \quad (3)$$

where n is the number of scheduling contents. γ is the Zipf exponent characterizing the distribution.

$$R_j = \begin{cases} 1, & d_i \le D_j \\ 1 + \sum_{k=1}^{|B_i|} b_k / D_j, & d_i > D_j \end{cases}, \quad (4)$$

where R_j is taken as the replacement rate of jth ES, which embodies the size of replaced content by ES regarding unit stored resources. When ES has enough space for storing the scheduling content, there is no replacement rate on ES. Otherwise, there exists the replacement rate on ES. With larger value of

replacement rate, the absence rate of requirements sent by the corresponding ES for storing the scheduling contents is increasing.

The transmission cost of a scheduling content is defined as follows:

$$S_{ij} = \mu h\left(j, j^{'}\right), \quad j^{'} \in \{1,2,\ldots,m\}, \tag{5}$$

where S_{ij} presents the transmission cost of scheduling content i between ES j and $j^{'}$. μ is an adjusting parameter ($\mu > 0$). $h(j, j^{'})$ denotes the network distance between ES j and $j^{'}$. Obviously, $h(j, j^{'}) > 0$ because $j \neq j^{'}$.

Then, the resource scheduling problem from cloud to edge can be formulated as problem (P1) below:

$$\text{(P1):} \quad \max \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} A_{ij}, \tag{6}$$

s.t.

$$x_{ij} \in \{0,1\}, \sum_{j=1}^{m} x_{ij} = 1 \tag{7}$$

$$A_{ij} = V_{ij} - P_{ij} - S_{ij} \tag{8}$$

$$i = 1,\cdots,n, \quad j = 1,\cdots,m, \tag{9}$$

(1), (2), (5), (9),

where $x_{ij} \in \{0,1\}$. If ith content is scheduled into jth ES, $x_{ij} = 1$. Otherwise, $x_{ij} = 0$. (12) represents that each content is only scheduled into one ES during each heartbeat. (1) is the stored penalty, which is determined by the stored space of ES and the size of scheduling content. When stored space of ES is enough for storing the scheduling content, there is no stored penalty. Otherwise, ES need releases some contents to make room for storing the scheduling content. In this case, the stored penalty is defined as the transmission cost of the released contents from DC to ES. (2) is the value of the scheduling content, which consists of the content popularity, the stored capability of ES and replacement rate of ES so that the most popular contents are scheduled into the ES with larger stored capability and with smaller replacement rate, which can hold the contents for a while to improve QoS. (5) is the transmission cost of the scheduling content, which is computed by the distances between the ESs and DC. Obviously, problem P1 is a 0-1 integer programming problem, which is an NP-hard problem. For addressing problem P1, the TS algorithm is applied to achieve the solution of problem P1 [43]. The detail description is shown in algorithm 1 in Section 4.

B. Resource matching strategy on edge servers

In this section, a resource matching strategy on edge servers is proposed by considering resource location, task priorities and network transmission cost. The resource location includes memory locality and disk locality. The memory locality implies that the tasks and corresponding resources are in the same ES and the resources are stored into the memory of the ES. The disk locality indicates that the tasks and corresponding resources are in the same ES. The resource location is used to build the relationship between tasks and containers in terms of similarity, where a container is taken as a set of the computation resources. The task priorities consist of job priorities and local resource priorities, which make sure the tasks with high job priorities and with corresponding local resources have the priority process chance. The network transmission cost is calculated by the size of the required resources, the distances and bandwidths between the ES and the locations that require the resources, such as local memory, local disk, neighbor ES or remote DC, to evaluate the minimum transmission cost of the required resources from the location that

stores the corresponding resources to the location where the tasks wait for the resources. The relationship between jobs and tasks, the characteristics of tasks and the definition of the container are described, respectively, as follows.

EUs send f Job requests to the corresponding ES, where each job z ($0 \le z \le f$) consists of a total number of l_z tasks. The tasks are indivisible, independent and non-preemptible. Each task h_z ($1 \le h_z \le l_z$) is presented by a tuple (v_h^z, s_h^z), where v_h^z is taken as the size of computation required to accomplish task h_z and s_h^z is defined as the size of memory required for the computation of task h_z. The set of containers is defined by $U = \{u_1, \cdots, u_p\}$, where p is the number of containers. Here $u_r \triangleq (v_p, s_p)$ $(r = 1, \cdots, p)$, where v_p denotes the capacity of CPU and s_p is taken as the size of memory for container u_r.

The similarity between task h_z and container u_r is determined by

$$Sim(h_z, u_r) = \frac{h_z \cdot u_r^{'}}{|h_z||u_r|}, \tag{10}$$

where $u_r^{'}$ denotes the transpose of vector u_r, $|*|$ is taken as the modulus of vector $*$.

The priority of task h_z can be defined as $y(h_z)$, which can be computed by

$$y(h_z) = \eta y(z) - (1-\eta) y_h^z \tag{11}$$

where $y(z)$ denotes the priority of job z, which is determined by the scheduling priority in the data-processing system. y_h^z is defined as the priority of the resource required by task h_z. η is the weight coefficient. With larger value of $y(h_z)$, the priority of task h_z becomes higher. y_h^z is determined according to the location of the required resource. When the required resource is stored into the memory of local ES, $y_h^z = 2$. When the required resource is stored into the disk of local ES, $y_h^z = 1$. Otherwise, $y_h^z = 1/h(le, lo)$, where le denotes location of local ES. lo is the location of the required resource. $h(le, lo)$ is taken as the minimum network distance between local ES and the location of the required resource.

The network transmission cost is taken as $e(h_z, u_r)$, which takes the form

$$e(h_z, u_r) = \sum_{t=1}^{\delta} \frac{w_t h(le, w_t)}{band(le, w_t)}, \tag{12}$$

where w_t is defined as the size of tth required content. $h(le, w_t)$ denotes the minimum network distance between local ES and the location of the required content. $band(le, w_t)$ is taken as the bandwidth between local ES and the location of the required content. δ is the number of the required contents. When the required content is stored into local ES, $h(le, w_t) = 0$ and $band(le, w_t) = 1$.

Then, the resource matching (RM) problem on edge servers can be formulated as problem (P2) below:

$$\text{(P2):} \quad \max \sum_{z=1}^{f} \sum_{r=1}^{p} \sum_{h_z=1}^{l_z} \frac{\pi_{hr}^z Sim(h_z, u_r) y(h_z)}{\left[e(h_z, u_r) + 1\right]}, \tag{13}$$

s.t.

$$\pi_{hr}^z \in \{0, 1\}, \tag{14}$$

$$\sum_{r=1}^{p} \pi_{hr}^z = 1, \tag{15}$$

$$r = 1, \cdots, p, \quad h = 1, \cdots, l_z, \tag{16}$$

(14), (15), (16),

where $\pi_{hr}^{z} \in \{0,1\}$. If hth task is matched with rth container, $\pi_{hr}^{z} = 1$. Otherwise, $\pi_{hr}^{z} = 0$. (19) represents that each task is only matched with one container during each heartbeat. (14) is taken as the similarities between tasks and containers, which is determined by the data locality. (15) is used to calculate the task priorities which consist of job priorities and local resource priorities. It is used to make sure the tasks with high job priorities and with corresponding local resources have the priority process chance. (16) is the network transmission cost calculated by the size of the required resources, the distances and bandwidths between the ES and the locations that require the resources, such as local memory, local disk, neighbor ES or remote DC, to evaluate the minimum transmission cost of the required resources from the location that stores the corresponding resources to the location where the tasks wait for the resources. Obviously, problem P2 is a 0-1 integer programming problem, which is an NP-hard problem. For addressing the resource matching problem, we transfer problem P2 to an optimal matching problem of the weighted bipartite graph (OMWBG). The detail description is shown as follows.

In the weighted bipartite graph (WBG), the weight between task h_z and container u_r, which consists of the similarity between them, the priority of tasks and the network transmission cost between them, takes the form

$$\phi(h_z,u_r) = \frac{\lambda_1 Sim(h_z,u_r)}{\overline{Sim(h_z,u_r)}} + \frac{\lambda_2 y(h_z)}{\overline{y(h_z)}} - \frac{\lambda_3 e(h_z,u_r)}{\overline{e(h_z,u_r)}}, \tag{17}$$

where $\phi(h_z,u_r)$ denotes the weight between task h_z and container u_r. λ_1, λ_2 and λ_3 are the weight coefficient for the similarity, the priority and the network transmission cost, respectively.

$$\overline{Sim(h_z,u_r)} = \sum_{z=1}^{f}\sum_{r=1}^{p}\sum_{h_z=1}^{l_z} Sim(h_z,u_r) \Big/ \sum_{z=1}^{f}\sum_{h_z=1}^{l_z} p, \tag{18}$$

$$\overline{y(h_z)} = \sum_{z=1}^{f}\sum_{h_z=1}^{l_z} y(h_z) \Big/ \sum_{z=1}^{f} l_z, \tag{19}$$

$$\overline{e(h_z,u_r)} = \sum_{z=1}^{f}\sum_{r=1}^{p}\sum_{h_z=1}^{z_g'} e(h_z,u_r) \Big/ \sum_{z=1}^{f}\sum_{h_z=1}^{l_z} p, \tag{20}$$

Then, OMWBG can be formulated as problem (P3) below:

$$(P3): \quad \max \sum_{z=1}^{f}\sum_{r=1}^{p}\sum_{h_z=1}^{l_z}\left[\pi_{h_z r}^{z}\phi(h_z,u_r)\right], \tag{21}$$

s.t.

$$\pi_{hr}^{z} \in \{0,1\}, \tag{22}$$

$$\sum_{r=1}^{p}\pi_{hr}^{z} = 1, \tag{23}$$

$$r = 1,\cdots,p, \quad h = 1,\cdots,l_z, \tag{24}$$

(14), (15), (16), (21).

When $\sum_{z=1}^{f} l_z > p$, we need add $\sum_{z=1}^{f}(l_z) - p$ spurious containers into container set so that there is a one-to-one match between each task and each container. The value of the weights between tasks and spurious containers is equal to zero. In this case, OMWBG can be regarded as the optimal matching problem of the weighted complete bipartite graph (OMWCBG). Kuhn-Munkres (KM) algorithm is a useful method to address OMWCBG. In this paper, we will adopt KM algorithm to achieve the solution of OMWCBG, i.e., the solution of problem (P2).

Ⅳ The dynamic resource allocation algorithm for cloud-edge environment

In this section, the dynamic resource allocation (DRA) algorithm for cloud-edge environment is presented. It consists of the resource scheduling algorithm and the resource matching algorithm, as shown below.

A. The resource scheduling algorithm for cloud-edge environment

Algorithm 1 represents the pseudo-code of the resource scheduling (RS) algorithm. Firstly, the popularity of each scheduling content can be achieved by equation (3). Secondly, the replacement rate for each ES is computed according to equation (4). Thirdly, the contents with most value are scheduled into ES. Therefore, the initial solution for problem (P1) can be obtained. Finally, the solution for problem (P1) is achieved by TS algorithm based on the initial solution. The time complexity of RS algorithm is $O\left(N_{iter}\cdot N_d^2\right)$, where N_{iter} is taken as the iteration number of TS algorithm, N_d is defined as the number of the scheduling contents.

Require: The set of the contents $Content=\left\{content_i \middle| i=1,2,\ldots,n\right\}$, the set of ESs $ES=\left\{es_j \middle| j=1,2,\ldots,m\right\}$

Ensure: The solution for problem (P1) $HashMap\langle Content, ES\rangle$

$HashMap\langle content, es\rangle \leftarrow \varnothing$, iniHashMap$\langle content, es\rangle$

for each $content_i \in Content$ **do**

Calculate H_i // The popularity of each content

$TR \leftarrow 0$

for each $es_j \in ES$ **do**

Calculate R_j // The replacement rate for each ES es_j

$V_{ij} \leftarrow H_i C_j / R_j$ // The value of scheduling content

if $TR < V_{ij}$ **then**

$TR \leftarrow V_{ij}$, $iniHashMap\langle content, es\rangle \leftarrow HashMap\langle content_i, es_j\rangle$

end if

end for each

end for each

$SP_{initial} \leftarrow P1\left(iniHashMap\langle content, es\rangle\right)$ // The initial solution for problem (P1)

$\max P1\left(HashMap\langle content, es\rangle\right) \leftarrow TS\left(SP_{initial}\right)$ // The optimal solution for problem (P1)

return $HashMap\langle content, es\rangle$

Algorithm 1. The resource scheduling algorithm

B. The resource matching algorithm for cloud-edge environment

Algorithm 2 describes the pseudo-code of the resource matching (RM) algorithm. Firstly, the similarities between tasks and containers, the task priorities and the network transmission cost for the required resources are computed by equation (10), (11) and (12), respectively. Secondly, the edge weights in WBG are achieved according to equation (17). Then, the OMWCBG is built during each heartbeat. The solution for OMWBG is obtained. The time complexity of RM algorithm is $O\left(\left\lceil\left(\sum_{g=1}^{f} z_g^{'}\right)^3\right\rceil \middle/ p\right)$, where $\sum_{g=1}^{f} z_g^{'}$ is the number of tasks and p is taken as the number of containers.

Require: The set of tasks: $H=\{h_1,\cdots,h_{l_z};\ z=1,\cdots,f\}$, the set of containers: $U=\{u_1,\cdots,u_p\}$,
the resource matching result: $HashMap\langle Content,ES\rangle$

Ensure: The solution for problem (P3)

for each $h_z\in H$, $u_r\in U$ **do**
Calculate $Sim(h_z,u_r)$ // The similarity between task h_z and container u_r
Calculate $y(h_z)$ // The priority for task h_z
Calculate $e(h_z,u_r)$ // The network transmission cost for the resources required by task h_z
end for each
for each heartbeat
if $\sum_{z=1}^{f} l_z > p$ **then**
Add $\sum_{z=1}^{f}(l_z)-p$ spurious containers //The weighted complete bipartite graph (WCBG)
Repeat
Randomly select an equality subgraph G' of WCBG
Until G' has a perfect matching $(task,container)$
else
Add $p-\sum_{z=1}^{f}(l_z)$ spurious containers
Record $(task,container)$ // The perfect matching of the equality subgraph of WCBG
end if
Update $(task,container)$
end for each
return $(task,container)$ for problem (P3)

Algorithm 2. The resource matching algorithm

C. Application Scenario: car navigation system with cloud-edge environment

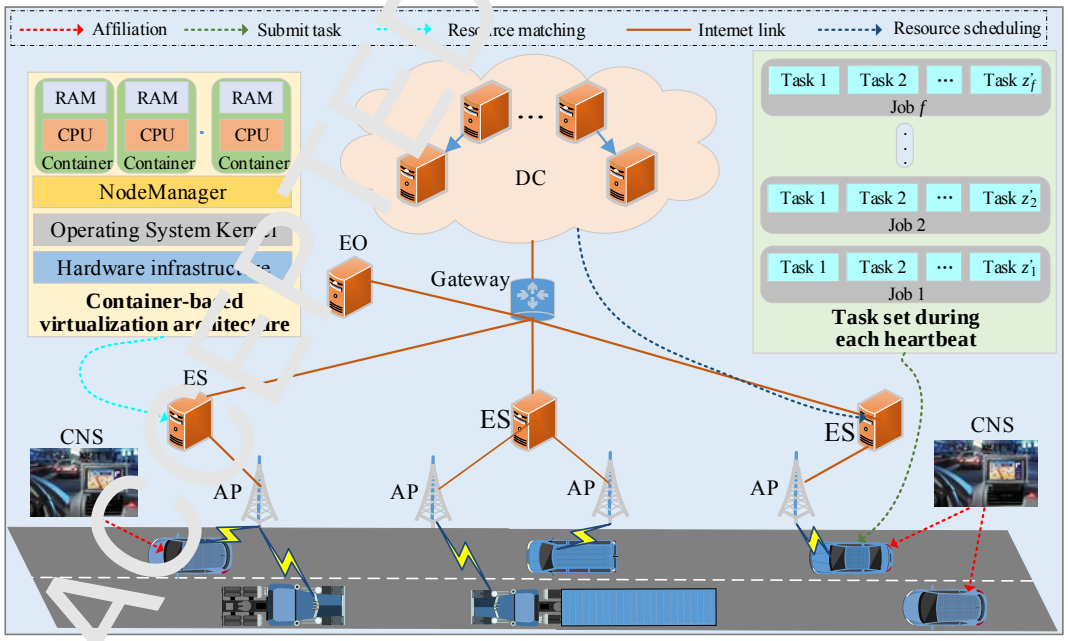

Fig. 3 The car navigation system with cloud-edge environment

In this section, an application scenario is illustrated to present the validity for DRA algorithm. With the development of transportation road, the latency-critical and computation-intensive requirements of car navigation system (CNS) have become increasing evident. In this case, it cannot satisfy the requirements

that the tasks are delivered to DC for computation or the required resources are transmitted from DC to EUs. Therefore, it will be an available method that the required resources are scheduled into corresponding ESs in advance. When the tasks from EUs are submitted into the ESs, the computation is conducted on ESs to reduce the response time and improve QoS. Fig. 3 describes the application scenarios on car navigation system with cloud-edge environment. CNS is widely used to plan a route, search information and so on. In this application scenario, the function of route planning of CNS in cloud-edge environment is illustrated as follows.

(1) Resource scheduling from cloud to edge. In the application scenarios on car navigation system with cloud-edge environment. For reducing the network delay and enhance the user experiences, the required resources on traffic information from DC are determined and scheduled into the memory or disk in each ES during each heartbeat. With RS algorithm, EO schedules the required resources and moves them into the corresponding ESs. In this paper, the disk and memory cooperatively store the required resources. During each heartbeat, the most popular traffic information is scheduled into the memory of ESs.

(2) Resource matching. Each CNS submits its job of route planning to the corresponding ES by AP. Each job consists of multiple tasks. Then, the ES will configure the corresponding containers for the tasks. Each ES can launch multi containers. Each container consists of corresponding RAM and CPU for the tasks. In addition, it is worthy of note that each container only processes a task at once. Finally, each ES matches its containers and tasks for responding to its CNS by RM algorithm.

Ⅴ Performance Evaluation

A. Experiment environment

In this section, the dynamic resource allocation algorithm for cloud-edge environment will be experimentally verified. In the experiments, the Hadoop 2.7.1, which consists of YARN and MapReduce, is applied, where the computation architecture is built based on YARN. The cloud-edge environment consists of 1 edge orchestrator (EO) and 10 ESs, where the operating system of each ES is Ubuntu 17.10.1. The version of the Java Development Kit is JDK 1.7.0-45 and the MyEclipse 10 is the development environment. The end users (EU) communicate with ESs by APs and all members in cloud-edge environment can communicate with the networks. The DC is connected to the entire network.

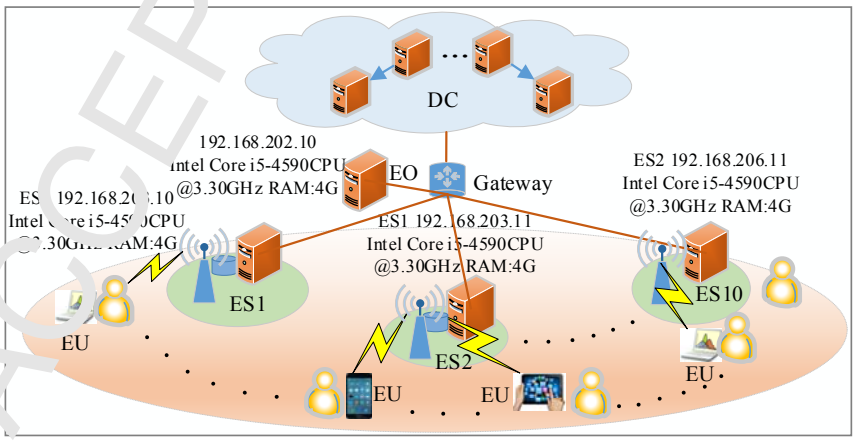

Fig. 4 The experimental environment for cloud-edge environment

Fig. 4 shows the experimental environment for cloud-edge environment. The RS algorithm will be run on EO. During each heartbeat, EO schedules the resources from DC to ESs by running the RS

algorithm. The jobs of EUs are submitted to ESs by AP. Based on the requests of EUs, the resource manager in each ES will assign corresponding containers and report the container information to resource scheduler. Due to the limitation of resources in each ES, the number of containers may be less than that of tasks. This problem can be addressed by our proposed RM algorithm. The RM algorithm runs on resource scheduler to match containers up with tasks.

The running process for RS algorithm is described in Fig. 5.

Firstly, the information on the set of contents and the set of ESs are achieved. For each scheduling content, the value of each scheduling content is calculated by Equation (2). The stored penalty for each scheduling content is computed by Equation (1). The transmission cost of each scheduling content between ESs and DC is determined by Equation (5). The resource scheduling model is initialized based on the information on the set of contents and the set of ESs.

Then, the scheduling contents and edge servers are selected. The initialization of the resource scheduling model is regarded as the local optimal solution and the optimal solution of the resource scheduling problem. The end condition of RS algorithm is either maximum number of iterations or finding the optimal solution of resource scheduling model. The local optimal solution needs to be checked to make sure it is an optimal solution or not. If the local optimal solution is the optimal solution of the resource scheduling problem, RS algorithm ends. Otherwise, the local optimal solution will be added into the taboo list. The taboo list is updated and RS algorithm continues until the end conditions of RS algorithm are met.

Finally, the resources are scheduled into ESs. With the help from EO, the contents are chosen and are scheduled into the disk or memory on ESs. In this paper, the disk and memory coordinately store the resources. During each heartbeat, the most popular contents are scheduled into memory.

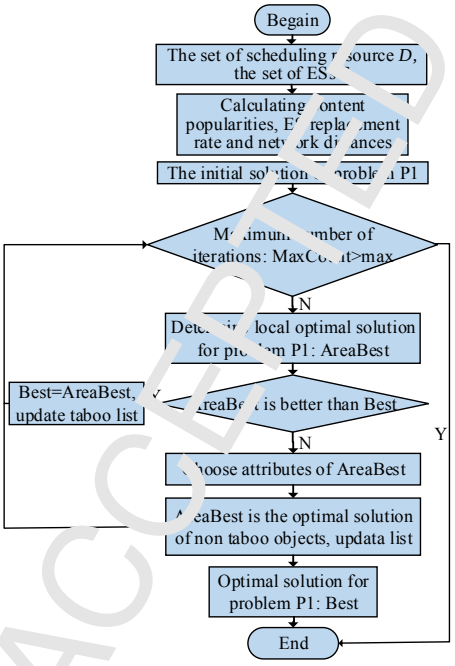

Fig. 5 The running process for RS algorithm

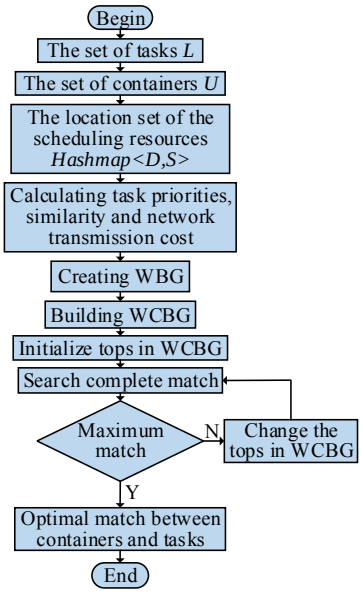

Fig. 6 The running process for RM algorithm

The running process for RM algorithm is described in Fig. 6.

Firstly, task submission. The information on scheduling contents, such as location, size, etc., is

achieved by the resource scheduler. In each ES, the requests for tasks are sent to the resource manager by application master.

Secondly, container creation. Based on the requests for tasks, the resource manager creates the corresponding containers for tasks in turn and reports the container information, such as CPU, RAM, etc., to application master.

Then, resource matching preparation. RM algorithm runs on the resource scheduler to match the containers up with tasks. In RM algorithm, with the information on the location of the scheduling resources, the containers and the tasks, the similarities between containers and tasks, the priorities of tasks and the network transmission cost of resources are calculated by Equation (10), (11) and (12), respectively. Then, the edge weights in the weighted bipartite graph are computed by Equation (17). To address the problem of an unequal number of containers and tasks, the spurious containers need to be added into the weighted bipartite graph so that the weighted bipartite graph is converted to the weighted complete bipartite graph. KM algorithm is applied to obtain the optimal matching relationship between tasks and containers.

Finally, matching between tasks and containers. According to the solution of problem P3, the resources manager matches the tasks and containers for reducing server latency and improving the network transmission efficiency in cloud-edge environment.

B. Test case

In the experiments, all data are derived from the Stanford Network Analysis Project (SNAP) [44]. The dataset consists of 96 million memes from the Meme tracker. A meme tracker is a tool for analyzing blog posts and discovering what web pages are being discussed or cited most often on the World Wide Web [45][46]. For the dataset, the quotes and phrases for the entire online news spectrum, which appear most frequently during August 2008 to April 2009, are tracked by Meme tracker. The dataset is raw trace data is enormous in volume, approximately 13..6G and comprises texts including URL of the document, time of the post, phrase extracted from the text of the document, hyperlinks in the document, etc. The dataset covers more than 17 million different phrases, where about 54% of them appear on the blog and 46% are in news media. The trace for task execution is collected by standard tools in Hadoop, i.e., by Web GUI in Hadoop YARN.

In the experiments, the test case is the Wordcount, which is the stand tool in Hadoop, for word counts. The stored space of ESs is different so that there is a better verification for the performance of RS algorithm. The size of the stored resource is 64M and the size of each task is 64M. For evaluating the performance of RS algorithm, the resource scheduling algorithm provided by Hadoop (RSH) and the greedy chunk placement (GCP) algorithm [47] are taken as the benchmark algorithms. In order to verify the performance of RM algorithm, the resource matching algorithm provided by Hadoop (RMH) and the data-locality aware workflow scheduling (D-LAWS) algorithm [48] are regarded as the benchmark algorithms.

C. Evaluation metrics

In the verification experiments of RS algorithm, the server rate (SR) is given to evaluate the algorithm performance. SR is computed based on the percentage of the response times of the local data among the overall response times in one ES as shown in Equation (25).

$$SR = \frac{response\ times\ of\ the\ local\ data}{overall\ response\ times}. \tag{25}$$

In the verification experiments of RM algorithm, the content hit rate (CHR) and the data transmission time (DTT) are taken as the metrics to evaluate the performance of RM algorithm. CHR is the percentage of the number of tasks hit by scheduling contents among overall number of tasks. DTT is the transmission time of the resources required by tasks. When the required resources are stored into local disk or memory, DTT is equal to zero.

D. Experimental results

In this section, the experimental results will be described. The experimental results mainly include the experimental results for RS algorithm and that for RM algorithm, which are described as follows.

1) Experimental results for RS algorithm

Fig. 7 depicts SR with different available storage space. With larger available storage space, SR for each algorithm increases. SR for RS algorithm is largest than that of other two algorithms. SR for RSH algorithm is smallest than that of other two algorithms. But with the size of available storage space increasing, the difference between SRs for three algorithms is smaller. For instance, when the size of available storage space is 50, SR for RS algorithm is improved up to 10.75% comparing with that for RSH algorithm, up to 4.51% comparing with that for GCP algorithm. However, when the size of available storage space is 55, SR for RS algorithm can only achieve up to 1.97% and 0.56% improvement over that for RSH algorithm and GCP algorithm, respectively. For this case, the reason is that, with larger size of available storage space, ES can contain more scheduling resources so that SR for each algorithm is improved and the difference between SRs for three algorithms becomes small. Therefore, with larger size of available storage space, the difference between SRs for three algorithms becomes small, even equal. In other words, the SR difference between algorithms becomes more obvious with the size of available storage space below 25 and becomes smaller or equal to the size of available storage space above 55. RSH algorithm only focuses on the size of the available storage space. GCP algorithm and RS algorithm pay close attention to the content popularities. In GCP algorithm, the contents with larger popularities take more scheduling chances. However, in RS algorithm, the contents with larger popularities not only are scheduled, but are scheduled into the ESs with smaller replacement rate so that the popular contents are not replaced easily and are stored for a while. So, the performance on SR for RS algorithm is better than that for other two algorithms.

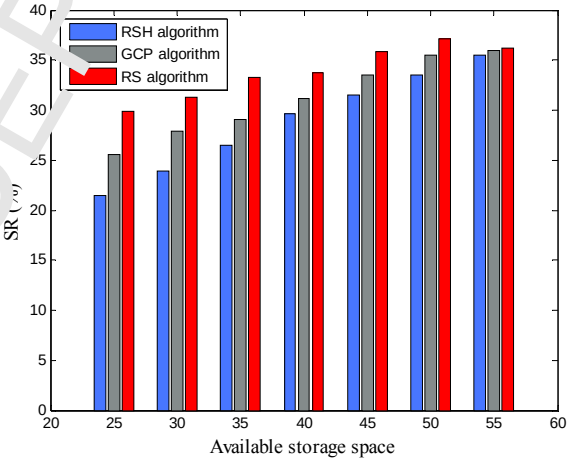

Fig. 7 SR with different available storage space

Fig. 8 depicts SR with different number of accessed contents. With larger number of accessed

contents, SR for each algorithm increases. For example, SR for RS algorithm with 256 accessed contents can achieve up to 22.83% improvement over that with 16 accessed contents. SR for RS algorithm is largest than that of other two algorithms. SR for RSH algorithm is smallest than that of other two algorithms. For instance, with 16 accessed contents, SR for RS algorithm is improved up to 27.64% comparing with that for RSH algorithm, up to 14.93% comparing with that for GCP algorithm. For each algorithm, with larger number of accessed contents, the response times of local data in ES are improved so that SR becomes larger. In GCP algorithm, the contents with larger popularities take more scheduling chances. In RS algorithm, the contents with larger popularities not only are scheduled, but are scheduled into the ESs with smaller replacement rate so that the popular contents are not replaced easily and are stored for a while. Therefore, the performance on SR for RS algorithm and GCP algorithm are better than that for RSH algorithm. Furthermore, the performance on SR for RS algorithm is better than that for GCP algorithm.

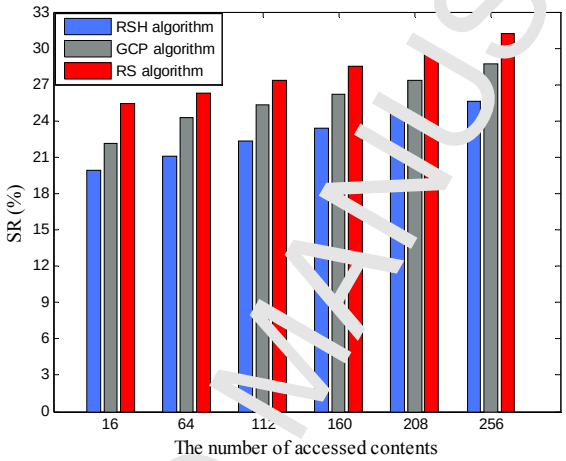

Fig. 8 SR with different number of accessed contents

2) Experimental results for RM algorithm

Fig. 9 describes CHR with different available storage space. With larger size of the available storage space, CHR for each algorithm increases. For example, when the size of the available storage space changes from 25 to 55, CHR for RM algorithm can achieve up to 62.31% improvement. CHR for RM algorithm is largest than that of other two algorithms. CHR for RMH algorithm is smallest than that of other two algorithms. For instance, when the size of the available storage space is 55, CHR for RM algorithm is improved up to 107.05% comparing with that for RMH algorithm, up to 24.23% comparing with that for D-LAWS algorithm. For this case, the reason is that, with larger size of the available storage space, ES can contain more scheduling contents so that CHR for each algorithm is improved. Except the factors considered by RMH and D-LAWS algorithm, RM algorithm considers the resource location. Therefore, the performance on CHR for RM algorithm is better than that for other two algorithms. Moreover, when the size of available storage space is more than 55 or less than 25, CHR for D-LAWS algorithm and RMH algorithm cannot exceed that for RM algorithm with the same size of available storage space.

Fig. 10 shows DTT with different available storage space. With larger size of the available storage space, DTT for each algorithm reduces. For example, when the size of the available storage space changes from 25 to 55, DTT for RM algorithm can achieve up to 69.62% reduction. DTT for RM algorithm is smallest than that of other two algorithms. DTT for RMH algorithm is largest than that of other two

algorithms. For instance, when the size of available storage space is 55, DTT for RM algorithm is reduced up to 37.78% comparing with that for RMH algorithm, up to 30.03% comparing with that for D-LAWS algorithm. It is well known that the transmission time of local data resources is much less than that of non-local ones. With larger size of the available storage space, ES can contain more resources so that DTT for each algorithm is improved. In D-LAWS algorithm, the contents with larger popularities take more scheduling chances so that the utilization of available storage space is improved. Moreover, in RM algorithm, the contents with larger popularities not only are scheduled, but are scheduled into the ESs with smaller replacement rate so that the popular contents are not replaced easily and are stored for a while. Therefore, the performance on DTT for RM algorithm is better than that for other two algorithms. Moreover, when the size of the available storage space is more than 55 or less than 25, DTT for D-LAWS algorithm and RMH algorithm cannot exceed that for RM algorithm with the same size of available storage space.

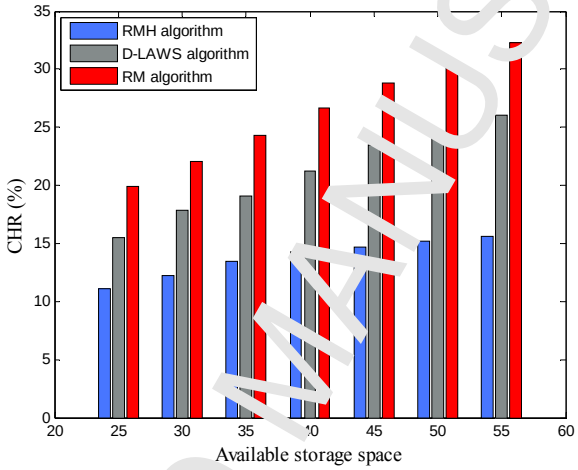

Fig. 9 CHR with different available storage space

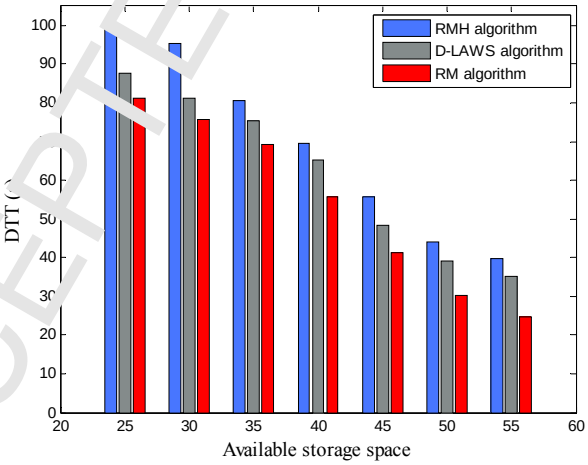

Fig. 10 DTT with different available storage space

Fig. 11 depicts CHR with different number of tasks. With larger number of tasks, CHR for each algorithm increases. For example, CHR for RM algorithm with 256 tasks can achieve up to 51.26% improvement over that with 16 tasks. CHR for RM algorithm is largest than that of other two algorithms. CHR for RMH algorithm is smallest than that of other two algorithms. For instance, with 256 tasks, CHR for RM algorithm is improved up to 98.03% comparing with that for RMH algorithm, up to 18.04%

comparing with that for D-LAWS algorithm. For each algorithm, with larger number of tasks, the number of tasks satisfied by local data resources becomes large so that CHR becomes larger. From 16 to 256 tasks, the change of CHR for RMH algorithm is not obvious. Because RMH algorithm focuses on disk locality, the resources have no change so that CHR for RMH has little change. From 16 to 256 tasks, CHR for RM has obvious change due to the joint function of disk locality and memory locality.

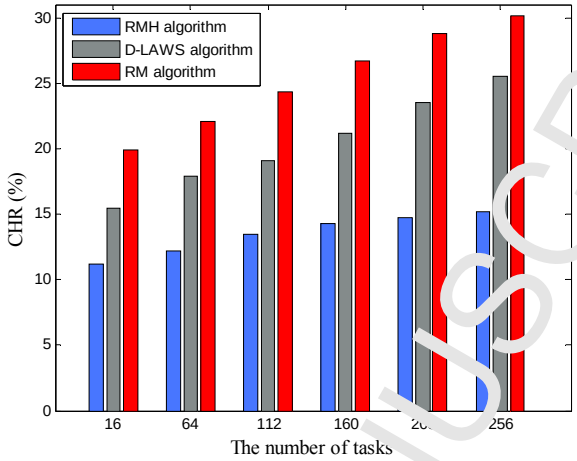

Fig. 11 CHR with different number of tasks

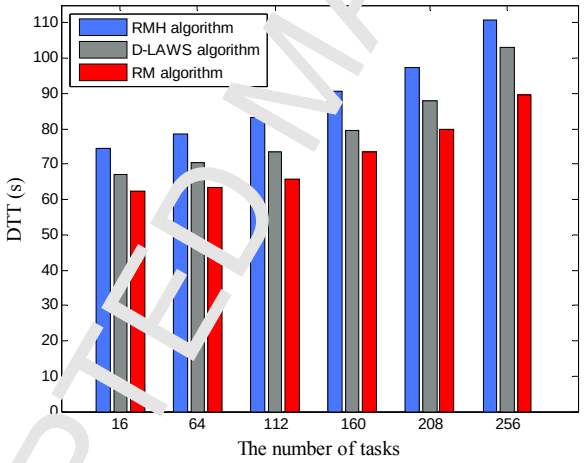

Fig. 12 DTT with different number of tasks

Fig. 12 describes DTT with different number of tasks. With larger number of tasks, DTT for each algorithm increases. For example, DTT for RM algorithm with 256 tasks can achieve up to 43.20% improvement over that with 16 tasks. DTT for RM algorithm is smallest than that of other two algorithms. DTT for RMH algorithm is largest than that of other two algorithms. For instance, with 256 tasks, DTT for RM algorithm is reduced up to 19.30% comparing with that for RMH algorithm, up to 13.17% comparing with that for D-LAWS algorithm. For each algorithm, given the available storage space and tasks, with the number of tasks increasing, the number of required resources without storing in local disk or memory becomes large so that DTT increases. RMH algorithm is an algorithm proposed based on the queue. It only focuses on the optimal allocation of resources for current tasks and neglects the optimal allocation of resources for global tasks. D-LAWS algorithm pays attention to the joint function of resource location and node computation capacity, but ignores the optimal allocation of resources for global tasks. However, RM algorithm considers not only the resource location and node computation capacity, but the

optimal allocation of resources for global tasks. Therefore, the performance on DTT for RM algorithm is better than that for other two algorithms.

Ⅵ Conclusion

In this paper, we propose the dynamic resource allocation algorithm for the cloud-edge environment. The dynamic resource allocation algorithm consists of the resource scheduling algorithm from cloud to edge and the resource matching algorithm on edge servers. The resource scheduling algorithm from cloud to edge is applied to improve the performance of the resource matching algorithm.

In the resource scheduling strategy from cloud to edge, a resource scheduling problem can be obtained according to the stored penalty of scheduling contents, the value of scheduling contents and the transmission cost of scheduling contents. The stored penalty of a scheduling content is determined by the stored space of ES and the size of scheduling content. When stored space of ES is enough for the scheduling data, there exists no stored penalty. Otherwise, ES needs to release some contents to make room for storing the scheduling content. In this case, the stored penalty is defined as the transmission cost of the released contents from DC to ES. The value of a scheduling content consists of the content popularity, the stored capability of ES and the replacement rate of ES so that the most popular contents are scheduled into the ES with larger stored capability and with smaller replacement rate, which can hold the contents for a while to improve QoS. The transmission cost of a scheduling content is computed by the distances between the ESs and DC. Then, tabu search (TS) is applied to the optimal solution of the optimal problem. Furthermore, the cache data are cached into ESs with the optimal solution.

In the resource matching strategy on edge servers, a resource matching model is proposed by considering the resource location, the task priorities and the network transmission cost. The resource location includes memory locality and disk locality. The memory locality implies that the tasks and corresponding resources are in the same ES and the resources are stored into the memory of the ES. The disk locality indicates that the tasks and corresponding resources are in the same ES. The resource location is used to build the relationship between tasks and containers in terms of similarity, where a container is taken as a set of the computation resources. The task priorities consist of job priorities and local resource priorities, which make sure the tasks with high job priorities and with corresponding local resources have the priority process chance. The network transmission cost is calculated by the size of the required resources, the distances and bandwidths between the ES and the locations that require the resources, such as local memory, local disk, neighbor ES or remote DC, to evaluate the minimum transmission cost of the required resources from the location that stores the corresponding resources to the location where the tasks wait for the resources. For addressing the resource matching problem, the optimal model is converted to an optimal matching problem of the weighted bipartite graph (OMWBG). Moreover, an optimal matching problem of the weighted complete bipartite graph (OMWCBG) is created by adding the spurious containers. Finally, the optimal resource matching strategy for tasks on the edge servers (ESs) is achieved.

The resources are scheduled into the edge servers by the resource scheduling algorithm. The tasks are scheduled by resource matching algorithm based on the location of the scheduling resources. Therefore, the resource matching algorithm considers not only the resource locations and the node computation capacities, but the optimal allocation of resources for global tasks. However, RMH algorithm is an algorithm proposed based on the queue. It only focuses on the optimal allocation of resources for current tasks and neglects the optimal allocation of resources for global tasks. D-LAWS algorithm pays attention

to the joint function of the resource locations and the node computation capacities. It schedules tasks based on round-robin scheduling and ignores the optimal allocation of resources for global tasks. So, in theory, the performance of the resource matching algorithm is better than that of RMH and D-LWAS algorithm. Moreover, the extensive experiments are conducted to evaluate the performance of the proposed resource matching algorithm, which indicates that the performance of the proposed resource matching algorithm is indeed better than that of RMH algorithm and D-LWAS algorithm with respect to CHR and DTT.

In this paper, the dynamic resource allocation algorithm is proposed based on the stably stored capacities of the edge servers for a duration of time. In an actual environment, the stored capacities change dynamically. Therefore, the dynamic resource allocation strategy combining with the dynamiclly stored capacities of the edge servers is a promising future research direction. Due to the limitation of the dataset in our experiment, the experiment with more larger scale dataset is also a significant extension of our work.

Acknowledgment

The work was supported by the National Natural Science Foundation (NSF) under grants (No.61672397, No. 61873341, No.61472294), Application Foundation Frontier Project of WuHan (No. 2018010401011290). Beijing Intelligent Logistics System Collaborative Innovation Center Open Project (No.BILSCIC-2018KF-02), Beijing Youth Top-notch Talent Plan of High-Creation Plan (No.2017000026833ZK25), Canal Plan-Leading Talent Project of Beijing Tongzhou District (No. YHLB2017038), and Beijing Key Laboratory of Intelligent Logistics System (No.BZ0211), Open Foundation of Hebei Engineering Technology Research Center for IOT Data acquisition & Processing, North China Institute of Science and Technology. Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the above agencies.

Reference

[1] Freeman H, Zhang T. The emerging era of fog computing and networking [The President's Page]. IEEE Communications Magazine, 2016, 54(6): 4-5.

[2] Chang H, Hari A, Mukherjee S, et al. Bringing the cloud to the edge. Computer Communications Workshops. IEEE, 2014: 346-351.

[3] Sun X, Ansari N. Latency aware workload offloading in the cloudlet network. IEEE Communications Letters, 2017, 99: 1-4.

[4] Kaur K, Dhand T, Kumar N, et al. Container-as-a-service at the edge: trade-off between energy efficiency and service availability at fog Nano data centers. IEEE Wireless Communications, 2017, 24(3): 48-56.

[5] Mao Y, You C, Zhang J, et al. A survey on mobile edge computing: the communication perspective. IEEE Communications Surveys & Tutorials, 2017, 99: 1-37.

[6] Wikitude. http://www.wikitude.com/.

[7] Apple Siri. http://www.apple.com/ios/siri/.

[8] George Pavlou, Ioannis Psaras. The troubled journey of QoS: from atm to content networking, edge computing and distributed internet governance. Computer Communications. 2018.07: 1-5.

[9] Li G, Li X, Yang F, et al. Traffic at-a-glance: time-bounded analytics on large visual traffic data.

IEEE Transactions on Parallel & Distributed Systems, 2017, 99: 1-9.

[10] He Y, Yu F R, Zhao N, et al. Software-defined networks with mobile edge computing and caching for smart cities: a big data deep reinforcement learning approach. IEEE Communications Magazine, 2017, 55(12): 31-37.

[11] Mao Y, You C, Zhang J, et al. Mobile edge computing: survey and research outlook. IEEE Communications Surveys and Tutorials. 2017: 1-30.

[12] Aral A, Ovatman T. A decentralized replica placement algorithm for edge computing. IEEE Transactions on Network & Service Management, 2018,15(2): 516-530

[13] Zeydan E, Bastug E, Bennis M, et al. Big data caching for networking: moving from cloud to edge. IEEE Communications Magazine, 2016, 54(9): 36-42.

[14] Maggi L, Leguay J. A closed/open-loop cache update strategy by peeking into the future. Computer Communications, 2017, 107: 49-59.

[15] Pellegrini F D, Massaro A, Goratti L, et al. Competitive caching of contents in 5G edge cloud networks. International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks. IEEE, 2017: 1-8.

[16] Al-Turjman F M. Cognitive caching for the future sensor in fog networking. Pervasive & Mobile Computing, 2017, 42: 317-334.

[17] Darwish, S.M. and S.A.G. El. An intelligent database proactive cache replacement policy for mobile communication system based on genetic programming. International Journal of Communication Systems, 2018, 31(8): 3536-3549.

[18] Huang Y, Song X, Ye F, et al. Fair caching algorithms for peer data sharing in pervasive edge computing environments. 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017: 605-614.

[19] F., M., C. D. and J. L. Multi-level partial cache algorithm for content delivery networks. 2017 10th International Symposium on Computational Intelligence and Design (ISCID). 2017, 12: 320-323.

[20] Liu J, Bai B, Zhang J, et al. Cache placement in fog-rans: from centralized to distributed algorithms. IEEE Transactions on Wireless Communications, 2017, 16(11): 7039-7051.

[21] Tran T X, Pandey P, Hajisami A, et al. Collaborative multi-bitrate video caching and processing in mobile-edge computing networks. 2017 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS). IEEE, 2017: 165-172.

[22] X., Z. and Z. Q. Distributed mobile devices caching over edge computing wireless networks. in 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). 2017: 127-132.

[23] Maddah-Ali, M.A. and U. Niesen. Decentralized coded caching attains order-optimal memory-rate tradeoff. IEEE/ACM Transactions on Networking, 2015. 23(4): 1029-1040.

[24] Li W, Chan E, Feng G, et al. Analysis and performance study for coordinated hierarchical cache placement strategies. Computer Communications, 2010, 33(15): 1834-1842.

[25] Thilakarathna K, Jiang F Z, Mrabet S, et al. Crowd-cache: leveraging on spatio-temporal correlation in content popularity for mobile networking in proximity. Computer Communications, 2017, 100(100): 104-117.

[26] Zhang Z, Lung C H, Lambadaris I, et al. When 5G meets ICN: an ICN-based caching approach for mobile video in 5G networks. Computer Communications, 2018, 118: 81-92.

[27] Drolia U, Guo K, Tan J, et al. Cachier: edge-caching for recognition applications. 2017 IEEE 37th

International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017: 276-286.

[28] Drolia U, Guo K, Tan J, et al. Towards edge-caching for image recognition. IEEE International Conference on Pervasive Computing and Communications Workshops. IEEE, 2017: 593-598.

[29] Zhang X, Zhu Q. Spectrum efficiency maximization using primal-dual adaptive algorithm for distributed mobile devices caching over edge computing networks. Information Sciences and Systems. IEEE, 2017.

[30] Dai X, Wang X, Liu N. Optimal scheduling of data-intensive applications in cloud-based video distribution services. IEEE Transactions on Circuits & Systems for Video Technology, 2017, 99.

[31] Lim B, Kim J W, Chung Y D. CATS: cache-aware task scheduling for Hadoop-based systems. Cluster Computing, 2017, 20(1): 1-15.

[32] Wang C, Liang C, Yu F R, et al. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. IEEE Transactions on Wireless Communications, 2017, 16(8): 4924-4938.

[33] Zhou Y, Yu F R, Chen J, et al. Resource Allocation for information-centric virtualized heterogeneous networks with in-network caching and mobile edge computing. IEEE Transactions on Vehicular Technology, 2017, 66(12): 11339-11351.

[34] Stavrinides G L, Duro F R, Karatza H D, et al. Different aspects of workflow scheduling in large-scale distributed systems. Simulation Modelling Practice & Theory, 2017, 70: 120-134.

[35] Stavrinides G L, Karatza H D. The impact of data locality on the performance of a SaaS cloud with real-time data-intensive applications. IEEE/ACM, International Symposium on Distributed Simulation and Real Time Applications. IEEE Computer Society, 2017: 1-8.

[36] Bryk P, Malawski M, Juve G, et al. Storage-aware algorithms for scheduling of workflow ensembles in clouds. Journal of Grid Computing, 2016, 14(2): 359-378.

[37] Zaharia M, Borthakur D, Sarma J S, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. European Conference on Computer Systems. ACM, 2010: 265-278.

[38] Guo Z, Fox G, Zhou M. Investigation of data locality and fairness in MapReduce. International Workshop on Mapreduce & Its Applications DATE. ACM, 2012: 25-32.

[39] Liu J, Mao Y, Zhang J, et al. Delay-optimal computation task scheduling for mobile-edge computing systems. IEEE International Symposium on Information Theory. IEEE, 2016: 1451-1455.

[40] Lyu X, Ni W, Tian H, et al. Optimal schedule of mobile edge computing for internet of things using partial information. IEEE Journal on Selected Areas in Communications, 2017, 35(11): 2606-2615.

[41] Chen Q, Yao J, Li P, et al. PISCES: Optimizing multi-job application execution in MapReduce. IEEE Transactions on Cloud Computing, 2016, 99: 1-14.

[42] Zhang X, Zhu Q. Distributed mobile devices caching over edge computing wireless networks. Computer Communications Workshops. IEEE, 2017.

[43] Larumbe F, Sansò B. A Tabu Search Algorithm for the Location of Data Centers and Software Components in Green Cloud Computing Networks. IEEE Transactions on Cloud Computing, 2013, 1(1): 22-35.

[44] SNAP website: http://snap.stanford.edu/, 2017.

[45] Leskovec J, Backstrom L, Kleinberg J. Meme-tracking and the dynamics of the news cycle. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2009: 497-506.

[46] Wikipedia website: https://en.wikipedia.org/wiki/Memetracker#External_links.

[47] Xu Y, Ci S, Li Y, et al. Design and evaluation of coordinated in-network caching model for content centric networking. Computer Networks, 2016, 110: 266-283.

[48] Choi J, Adufu T, Kim Y. Data-locality aware scientific workflow scheduling methods in hpc cloud environments. International Journal of Parallel Programming, 2017, 45(5): 1-14

[49] Li Chunlin, Liu Yanpei, Luo Youlong, Zhou Min, Collaborative content dissemination based on game theory in multimedia cloud, Knowledge-Based Systems, Elsevier, Volume 124, 15 May 2017, Pages 1–15

[50] Chunlin Li, Jing Zhang, Youlong Luo, Real-Time Scheduling Based on Optimized Topology and Communication Traffic in Distributed Real-Time Computation Platform of Storm, Journal of Network and Computer, Elsevier, Volume 87, 1 June 2017, Pages 100–115

[51] Li Chunlin, Liu Yanpei, Luo Youlong, Efficient Service Selection Approach for Mobile Devices in Mobile Cloud, Journal of Supercomputing, Springer-Verlag, 2016, 72(6) : 2197-2220

[52] Li Chunlin, Yan Xin, Li LaYuan, Flexible Service Provisioning Based On Context Constraint for Enhancing User Experience in Service Oriented Mobile Cloud, Journal of Network and Computer Applications, Elsevier, Volume 66, May 2016, Pages 250–261