Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment

Li Chunlin, Tang Jianhang, Hengliang Tang, Youlong Luo

PII: S0167-739X(18)30943-9
DOI: https://doi.org/10.1016/j.future.2019.01.007
Reference: FUTURE 4700

To appear in: *Future Generation Computer Systems*

Received date : 20 April 2018
Revised date : 29 November 2018
Accepted date : 4 January 2019

Collaborative Cache Allocation and Task Scheduling for Data-Intensive Applications in Edge Computing Environment

Li Chunlin[1,2,3], Tang Jianhang[1] , Hengliang Tang[4] , Youlong Luo[1]

[1] Department of Computer Science, Wuhan University of Technology, Wuhan 430063, P.R.China

[2] Key Laboratory of Geographic Information Science of Ministry of Education, East China Normal University, Shanghai, 200241, China

[3] Key Laboratory of Urban Land Resources Monitoring and Simulation, Ministry of Land and Resources， Shenzhen, P.R.China

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

* Corresponding author:chunlin74@tom.com

Abstract—In the wake of the development of mobile devices, how to provide low-latency mobile services with the limited battery power is attracting more and more attention. A novel paradigm, edge computing, can make services closer to users, which can dramatically reduce the latency and improve battery life of UEs. However, inappropriate placement and utilization of caching can degrade the system performance. In this paper, a cache-aware task scheduling method in edge computing is proposed. First, an integrated utility function is derived with respect to the data chunk transmission cost, caching value and cache replacement penalty. Data chunks are cached at optimal edge servers to maximize the integrated utility value. After placing the caches, a cache locality-based task scheduling method is presented. We model the task scheduling problem as a weighted bipartite graph. Weights of edges of the graph are mainly influenced by the locations of the required data. During each heartbeat, maximal weighted matching between tasks and resources are obtained. All the proposed algorithms have polynomial time complexities which are acceptable in edge computing. Furthermore, extensive experiments show that the cache-aware task scheduling algorithm outperforms other baseline algorithms in terms of the cache hit ratio, data locality, data transmission time, task response time and energy consumption costs.

Index Terms—Edge computing, Cache placement, Task scheduling, Weighted bipartite graph

1 Introduction

Recently, with the advent of technological evolution of portable mobile devices, such as smartphones, laptops and sensors, the limitations of battery capacity and bandwidth have been serious obstacles for the quality of service (QoS). The traditional solution to cover these limitations is to offload applications with high resource requirements to a conventional core cloud [1], [2]. However, it is not efficient enough to transmit applications to the cloud due to limited bandwidths. The vision of edge computing that can make the service closer to users has led the path to a manner with low delays for mobile users. Edge computing mainly consists of following computing concepts, Fog Computing [3], [4], Cloudlet [3], [4], [5] and Mobile Edge Computing [3], [4]. Executing a portion of applications on the edge servers can reduce the amount of data transmitted in the network, which reduces both latency and energy consumption costs.

Many application scenarios benefit from edge computing including face recognition application [6], [7], IoT [6] and connected vehicles [6], [8]. All of these applications need low-latency services. The architecture of edge computing mainly consists of a core cloud, an edge orchestrator (EO) and several edge servers, as shown in Fig. 1. In order to overlook all edge servers, the edge orchestrator is connected to the same network with them. Edge servers are installed at WiFi access points or base stations for various scenarios [9]. Since the location of services is closer to users, the edge computing is receiving more and more attention.

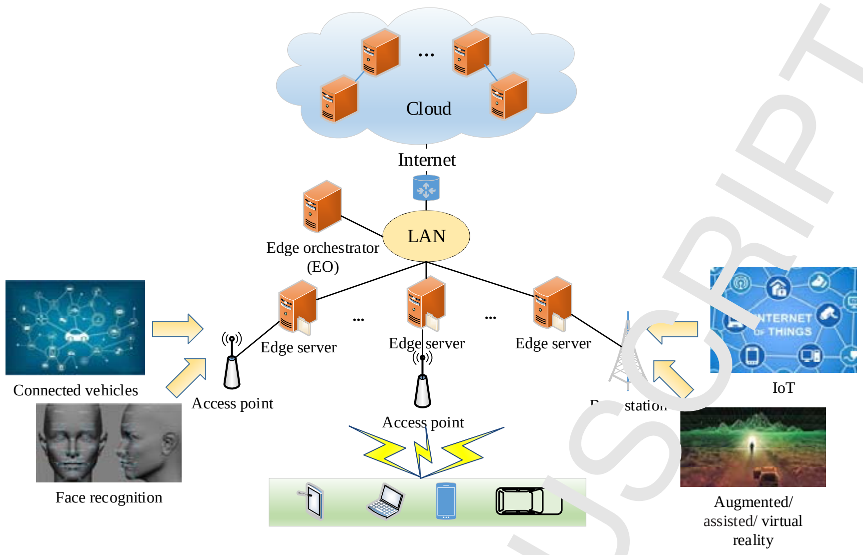


Fig. 1 Application scenarios of edge computing

In edge computing system, each edge server has a certain cache size in the memory. The popular contents will be dispatched and cached in the edge servers to further improve the system performance. Moreover, excessive data transmission will increase the latency and energy costs. In order to improve the quality of service (QoS), the cache-aware task scheduling is regarded as a valid manner. However, there are many challenges in cache-aware task scheduling in edge computing.

First, caching spaces of edge servers are usually smaller than common servers in cloud computing. Therefore, the cache replacement occurs frequently if contents are cached at edge servers with high cache replacement rates. The system should make a decision to cache contents on edge servers not only according to caching values but also according to the replacement rates of edge servers. Thus, an integrated utility-based cache placement strategy which jointly considers data transmission cost, caching values and cache replacement penalty is necessary.

Second, reading data from local cache is faster than reading data from local disk or remote edge servers. In addition, reading data from local caches or disks can reduce the amount of data transmitted in the network. Most of current cache-aware scheduling algorithms require lots of iterations, which may cause higher entire system latency. A heuristic algorithm with polynomial time complexity is necessary to take full advantage of computing and caching resources to reduce both the latency and energy consumption costs.

The main contributions of this paper are shown as follows:

• An integrated utility-based cache placement strategy to reasonably place caches in edge computing system is proposed by jointly considering data chunk transmission cost, caching value and cache replacement penalty. The data chunks are cached at optimal edge servers to maximize the integrated utility value of caching.

• A weighted bipartite graph model is applied to describe the relationships between tasks and edge servers. Weights of the graph are mainly derived by the locations of required data. The data transmission cost for task scheduling is measured by data reading time and energy consumption costs. A heuristic algorithm named cache locality-based task scheduling algorithm is proposed. The proposed task scheduling algorithm can obtain maximal weighted matching during each heartbeat, which can reduce both latency and energy consumption costs.

• Finally, we evaluate the performance of the cache-aware task scheduling method and previous method via extensive experiments. The results indicate that the proposed task scheduling method improves the cache hit ratio and data locality. And it also reduces the data transmission time, task response time and the energy consumption costs of the system significantly.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 proposes the cache-aware task scheduling method. Section 4 provides the analysis of experiment results. Conclusions are made in Section 5.

2 Related work

2.1 Cache placement in edge computing

Some applications in edge computing need low-latency services. The content caching is regarded as a promising technique to reduce the network delays. Since caching spaces of edge servers are limited, only a part of contents can be cached. Many researchers studied edge caching policies mainly according to popularity values of contents and cache storage constraints of edge servers. In previous references, contents with high popularity values were selected and cached in edge servers under various storage space constraints. Zeydan et al. [10] studied content caching in 5G wireless networks and presented a big-data-enabled architecture. This architecture can harness a vast amount of data to estimate the content popularity and cache strategic contents to improve the user satisfaction and backhaul offloading. The authors only considered the limited storage capacities of edge servers. In this work, the cache replacement penalty which is incurred by the limited caching spaces is proposed as a main factor to cache contents. Liu J et al. [11] proposed both centralized and distributed transmission aware cache placement approaches to reduce users' average download delay. They considered diverse content preferences of different users. Al-Turjman [12] considered four main parameters including age of the data, popularity of requests, delay to receive the information and data fidelity. However, the authors achieved the data popularity only by data request frequency. In this paper, the data access time and average time interval are also considered to obtain data popularity values. Tran et al [13] proposed a collaborative caching and processing method in Mobile-Edge Computing networks. They stored both the videos and their appropriate bitrate versions in the caches and considered the transcoding relationships among versions. Wang X et al. [14] proposed an edge caching architecture based on the content-centric networking. The authors evaluated content access delay and traffic load in experiments. In this paper, cache replacement number is studied as a vital parameter to indicate the system stability. Pellegrini F D et al. [15] proposed a caching policy derived by popularity of contents, caching strategies of competing content providers and spatial distribution of small cells. In this caching scheme, popular contents were cached in the intermediate servers. In previous works, only the storage spaces of edge servers were seen as constrains to cache contents with high popularity values. In this work, a dynamic caching process is analyzed. Both content caching and replacing are studied. The data transmission cost and cache replacement penalty are introduced. The significant differences of the proposed method are that the contents with high popularity values are cached in the edge servers with low cache replacement rates to avoid the eviction of cached data chunks frequently when the available cache size of edge servers is less than a certain threshold. Moreover, data chunks cached in edge servers need to be transmitted from the edge servers that store these data chunks. Thus, the data transmission cost is also considered.

Compared with servers in cloud datacenters, edge devices and servers are closer to users. Contents are cached in the edge devices and servers to reduce the data transmission delays. When applications arrive, contents are transmitted from edge devices or servers rather than cloud servers. There are many caching approaches to study the content placement in edge devices and servers. Drolia U et al. [16] presented an edge caching system called Cachier for recognition applications. They proposed to use edge servers as "caches with computing resources". Drolia U et al. [17] proposed a caching model that regarded edge servers as caches for compute-intensive recognition applications. However, the authors studied a coarse-grained cache placement problem. We consider a part of the memory in edge servers as caches, which can be seen as fine-grained caches. Lots of researchers studied the cache placement in edge devices (e.g. smartphones and tablets). Huang Y et al. [18] considered caching fairness issue among peer edge devices in edge computing. The path contention cost that was formulated as a linear transformation of the contention delay. We also applied the delay as cache replacement penalty. Moreover, cache replacement rate is proposed as a main factor to evaluate edge servers. Zhang X et al. [19] partitioned the entire wireless cell to avoid the interference in the edge computing network. They randomly distributed and cache the popular contents in the mobile devices. However, edge devices have limited computing resources, storage spaces and battery capacities. Moreover, the access speed of memory is faster than that of disk. In this work, we consider a part of the memory of each edge server as cache. Reading data from local memory of edge servers is faster than reading data from local disk or remote edge servers. The various configurations of edge servers including available caching space, CPU performance and memory speed are also considered, which can improve the utilization of caching and computing resources in the system.

2.2 Cache locality-based scheduling

In Hadoop-based systems, some data-intensive applications may cause high delays to slow down the system performance. Most literature studied the cache locality-based scheduling methods which can achieve high data availability and low data transmission cost. Tasks were dispatched to the nodes with required contents in caches to reduce data access costs. Lim B et al. [20] presented a cache-aware task scheduling method (CATS) that considered the data storage in memory layer to improve the system performance in Hadoop-based systems. We consider both cache locality and disk locality to make full use of system storage resources. Li G et al. [21] cached computing results for some complicated jobs to reduce the processing time of subsequent jobs with the same inputs and operations. Dai X et al. [22] proposed a Cache A Replica On Modification cloud file system to improve its efficiency. They applied a tripartite graph to present the relationships among computation nodes, data nodes and tasks by considering limitations of the cache sizes and task performance. In this paper, task scheduling problem is formulated as a weighted bipartite graph in which tasks are processed by logical bundles of computing resources bound to edge servers. Chen Q. et al [23] took full advantage of the file cache by leveraging the output data as soon as it was written to the file system in MapReduce. Tanaka M et al. [24] studied I/O-aware task scheduling problem to maximize the disk cache hit rate for data-intensive and many-task workflow. Only time consumption was considered as a main metric. We study the energy costs in both modelling part and experiments to discuss the energy utilization. Bryk P et al. [25] proposed a dynamic scheduling algorithm which took advantage of both file locality and data caching in clouds. This task scheduling algorithm can decrease the number of file transfers. In these references, contents were cached in one location, such as memory or disk. When tasks were scheduled, only the differences of reading data between caches and remote servers were studied. In this work, a part of memory is regarded as cache. We consider the diverse data locations including local caches, local disks and remote servers for task scheduling. Tasks are set to different priority values according to the locations of required data chunks. Data transmission costs including data transmission delays and energy costs are achieved according to the above three locations. As a result, all kinds of storage resources are fully utilized to improve the system throughput.

Caching can reduce the data transmission costs significantly. Thus, some researches applied the caching to decrease the backhaul cost of popular contents and improve the performance of the network. Zhou Y et al. [26] studied the information-centric virtualized heterogeneous networks with mobile edge computing and in-network caching. They proposed a virtual resource allocation strategy which benefited from not only virtualization but also caching and computing. In contrast to the energy consumption for task execution in [26], the energy cost for data access is studied and evaluated in this paper. He Y et al. [27] studied software-defined networks with caching and mobile edge computing for smart cities. They applied a deep Q-leaening method to improve the utilization rates of networking, caching and computing resources. However computation abilities of edge servers were integrated to guarantee the service quality for applications. In this work, various computing resource requirements including CPU and memory are considered to improve the system utilization. Wang C et al. [28] studied the computation offloading and content caching strategy. They applied the alleviated backhaul bandwidth as the caching reward to improve the total revenue of the network. The previous researches mainly focused on the backhaul time of popular contents and the computing capacities of edge servers. In edge computing, applications submitted by mobile users are usually heterogeneous. Thus, we consider the resource requirements of different tasks. A weighted bipartite graph is proposed to make full use of caching and computing resources.

3. Cache-aware task scheduling method in edge computing

3.1 Cache-aware task scheduling model

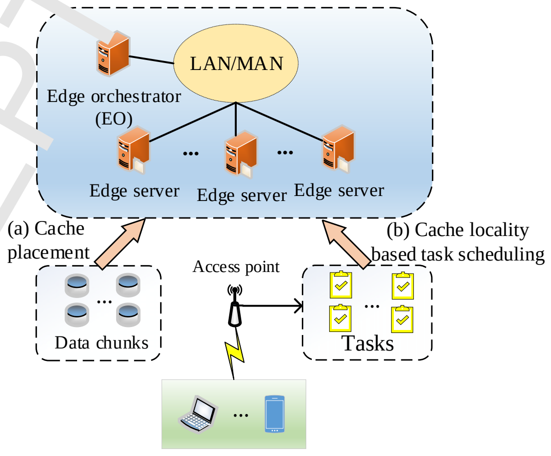


Fig. 2 The architecture of the cache-aware task scheduling

In edge computing architecture, an edge orchestrator and several edge servers are installed. The edge orchestrator manages all edge servers by Local Area Network (LAN) or Metropolitan Area Network (MAN). Mobile devices communicate with edge servers by wireless communications. Edge servers have some computing and caching resources. The edge orchestrator maintains overall view on both available computing resources and caching resources. Popular contents are dispatched and cached among all edge servers. Tasks submitted by users are scheduled and processed in these edge servers. The architecture of the cache-aware task scheduling is shown in Fig. 2.

In order to further reduce latency, popular data should be cached at optimal edge servers and tasks should be scheduled based on the cache locality. Therefore, the cache-aware task scheduling method proposed in this work mainly includes the following two components:

(1) An integrated utility function is derived according to data chunk transmission cost, caching value and cache replacement penalty. Then, data chunks are cached at optimal edge servers to maximize the integrated utility value of caching.

(2) After caching data chunks, tasks will be dispatched to appropriate computing resources. The task scheduling problem is modeled as a matching problem between tasks and computing resources in a weighted bipartite graph model. The weights are mainly influenced by the locations of required data.

3.2 The integrated utility-based cache placement strategy

In order to reasonably cache the data chunks, an integrated utility-based cache placement strategy is proposed. The integrated utility function is a function of the data chunk transmission cost, caching value and cache replacement penalty to evaluate the caching results. The tabu search is given to obtain the optimal cache placement with maximal cache placement integrated value.

3.2.1 Integrated utility function of caching

An integrated utility function is derived with respect to the data chunk transmission cost, caching value and cache replacement penalty to evaluate the caching results as shown in equation (1). The data chunk transmission cost is measured by the network distances among edge servers. The caching value is proposed according to cache capacities of edge servers, replacement rates of edge servers and data popularity values. The cache replacement penalty is given by the available cache size of edge servers and caching data size. The goal of the optimal cache placement strategy is to maximize the integrated utility value of caching (*CIUV*). Therefore, the cache placement problem can be formulated as the following programming problem,

$$\max CIUV = \sum_{i=1}^{N_d}\sum_{n=1}^{N} x_{i,n}\cdot\left(Value_i^n - Acq_i^n - Penalty_i^n\right). \tag{1}$$

$$\text{s.t.}\begin{cases} x_{i,n}\in\{0,1\}, \forall i\in\{1,2,\dots,N_d\}, n\in\{1,2,\dots,N\} \\ \sum_{n=1}^{N} x_{i,n}=1, \forall i\in\{1,2,\dots,N_d\} \\ 1\le\left|\left\{n\,\middle|\,x_{i,n}=1, \forall i\in\{1,2,\dots,N_d\}\right\}\right|\le N. \end{cases} \tag{2}$$

where $Value_i^n$ denotes the caching value, Acq_i^n is the data chunk transmission cost and $Penalty_i^n$ represents the cache replacement penalty.

A. Data chunk transmission cost

In the system, data is divided into equal size data chunks. Let $D=\{d_i\,|\,i=1,2,\dots,N_d\}$ be the data chunk set which consists of N_d data chunks. The size of each data chunk is a constant ds. The edge servers with different configurations are denoted by a set $S=\{s_n\,|\,n=1,2,\dots,N\}$. We assume that if a data chunk needs to be cached at a certain edge server, this edge server should acquire this data chunk from other edge servers. Then, the data chunk transmission occurs. The data chunk transmission cost Acq_i^n for edge server s_n to acquire data chunk d_i can be defined from (3),

$$Acq_i^n = a \cdot h\left(s_n \cdot s_{n'}\right),\ n, n' \in \{1, 2, \ldots, N\} \text{ and } n \neq n' \tag{3}$$

where a is a positive constant and $h\left(s_n, s_{n'}\right)$ represents the network distance between s_n and $s_{n'}$. Assume that an edge server at which data chunks are cached cannot be the one that stores data chunk replications. Therefore, $h\left(d_n, d_{n'}\right) > 0$.

B. Caching value

The caching value is proposed to evaluate the result that a data chunk is cached on a certain edge server. It is a function of cache capacities of edge servers, data popularity values and replacement rates of edge servers. The configurations of edge servers are usually different. This leads to the different cache space contentions on edge servers. Thus, it is significant to consider various cache capacities of edge servers to place caches appropriately. Let Cap_n denote the cache capacity of edge server s_n as defined in (4), which is influenced by the proportion of available caching space, CPU performance and memory speed.

$$Cap_n = \sqrt[3]{F_n^{(1)} \cdot F_n^{(2)} \cdot F_n^{(3)}}\,. \tag{4}$$

where $F_n^{(1)}$ denotes the proportion of available cache of edge server s_n, $F_n^{(2)}$ represents CPU performance of edge server s_n and $F_n^{(3)}$ denotes memory performance of edge server s_n.

In the following, how to achieve efficacy coefficients, $F_n^{(1)}$, $F_n^{(2)}$ and $F_n^{(3)}$, will be introduced. Let cs_n denote the cache size of edge server s_n and acs_n be the available cache size of edge server s_n, where $n \in \{1, 2, \ldots, N\}$. $F_n^{(1)}$ can be defined in (5),

$$F_n^{(1)} = \frac{p_n^{aca} - \min p_n^{aca} + a_1}{\max p_n^{aca} - \min p_n^{aca} + a_1} \tag{5}$$

where $p_n^{aca} = acs_n / cs_n$ is the percentage of the available cache size.

Let $MIPS_n$ represent Million Instructions per Second (MIPS) of the CPU of edge server s_n. Then, $F_n^{(2)}$ can be defined as follows,

$$F_n^{(2)} = \frac{MIPS_n - \min MIPS_n + a_2}{\max MIPS_n - \min MIPS_n + a_2}\,. \tag{6}$$

Let $AMAT_n$ represent average memory access time (AMAT) of the memory of edge server s_n. Then, $F_n^{(3)}$ can be defined in (7),

$$F_n^{(3)} = \frac{AMAT_n - \min AMAT_n + a_3}{\max AMAT_n - \min AMAT_n + a_3}\,. \tag{7}$$

a_1, a_2 and a_3 are constants in equations (5)-(8). Because any one of efficacy coefficients $F_n^{(1)} \sim F_n^{(3)}$ cannot be 0, a term a_i is in the numerator. If any one of efficacy coefficients is 0, Cap_n will be 0 no matter what the values of the other efficacy coefficients are. Cap_n will be 1 when all variables including p_n^{aca}, $MIPS_n$ and $AMAT_n$ achieve their minimum values. Hence, a term a_i is in the denominator.

The cache replacement will occur when the available cache size of an edge server is less than a certain threshold. Because of the heterogeneity of edge servers, a number of edge servers may replace data chunks frequently. If a cached data chunk is replaced, this data chunk should be transmitted from other edge servers when users request it. Frequent cache replacements may result in high extra costs. Requested multiple times by users, a data chunk should be cached at an edge server with low cache replacement rate. Let Rep_n denote the cache replacement rate of edge server s_n as depicted in (8),

$$Rep_n = \frac{1}{cs_n}\sum_{j=1}^{k_n} data_j^n \tag{8}$$

where $data_j^n$ represents the data size of caches to be replaced in jth cache replacement on edge server s_n.

The total number of cache replacement times is k_n on edge server s_n. Rep_n reflects the cache space contention of edge server s_n. It means that data chunks cached on edge servers with high cache replacement rates will be replaced frequently.

In order to improve the cache hit ratio and utilization of cache spaces, the data popularity is presented. Let Pop_i be the popularity of data chunk d_i which can be achieved as follow,

$$Pop_i = \left(RN_i \Big/ \sum_{i=1}^{N_d} RN_i \right) \cdot \left(\frac{1}{T^{now} - T_i^{last}} \right) \cdot \frac{1}{\left(T_i^{last} - T_i^{first}\right) \Big/ RN_i} \tag{9}$$

where RN_i denotes the number of requests for data chunk d_i, T_i^{last} denotes the time that data chunk d_i was last requested, T_i^{first} represents the time that data chunk d_i was first requested, T^{now} indicates the current time. In equation (9), $RN_i \Big/ \sum RN_i$ indicates the request frequency of data chunk d_i, $T^{now} - T_i^{last}$ reflects the recent request for data chunk d_i and $\left(T_i^{last} - T_i^{first}\right) \Big/ RN_i$ denotes the average time interval of requests for data chunk d_i.

Therefore, the value of caching a data chunk d_i on edge server s_n can be derived as depicted in (10),

$$Value_i^n = \frac{Pop_i \cdot Cap_n}{Rep_n}. \tag{10}$$

Then, the total caching value of N_d data chunks can be achieved in (11),

$$Value = \sum_{i=1}^{N_d}\sum_{n=1}^{N} Value_i^n = \sum_{i=1}^{N_d}\sum_{n=1}^{N} x_{i,n} \cdot \frac{Pop_i \cdot Cap_n}{Rep_n}. \tag{11}$$

where the binary variable $x_{i,n}$ is defined to denote whether a data chunk d_i is cached on edge server s_n as follows,

$$x_{i,n} = \begin{cases} 1, \text{ if data chunk } d_i \text{ is cached at edge server } s_n, \\ 0, \text{ otherwise.} \end{cases} \tag{12}$$

C. Cache replacement penalty

New data chunks will incur the eviction of certain cached data chunks at an edge server when the available cache size of this edge server is less than a certain threshold. Because future requests for the evicted data chunks cannot be served at this edge server, such eviction leads to replacement penalty. Let $Penalty_i^n$ be the replacement penalty of caching data chunk d_i on edge server s_n. $Penalty_i^n$ can be defined as (13),

$$Penalty_i^n = \begin{cases} 0, bs \le acs_n \\ val = \dfrac{bs}{band_n}, \text{otherwise} \end{cases} \tag{13}$$

As mentioned above, an integrated utility function of caching $CIUV$ is proposed according to data chunk transmission cost, caching value and cache replacement penalty. Then, $CIUV$ can be defined as depicted in (14),

$$CIUV = \sum_{i=1}^{N_d} \sum_{n=1}^{N} x_{i,n} \cdot \left(Value_i^n - Acq_i^n - Penalty_i^n \right) \tag{14}$$

3.2.2 Optimal cache placement

The goal of optimal cache placement strategy is to maximize integrated utility value of caching. The problem formulated in (1) can be reduced to a knapsack problem which is an NP-complete problem. Thus, it is an NP-hard problem. Tabu search (TS) is an efficient method that employs local search methods to solve combinatorial optimization problems [29]. The tabu search starts with an initial solution which is generated according to data popularity values and cache replacement rates. The calculation of the initial solution mainly includes three steps.

First, data chunks are sorted by data popularity values in descending order. Edge servers are sorted by cache replacement rates in ascending order.

Then, a data chunk d_i will be cached at an edge server s_n with the maximal ratio of Pop_i to Rep_n.

Finally, when all data chunks are cached, the initial solution is generated. New solutions are generated based on the initial solution until the stopping criterion (e.g. the maximum number of iterations) and tabu search can return the best one which is achieved during execution period.

3.2.3 The integrated utility-based cache placement algorithm

Algorithm 1: Integrated utility-based cache placement algorithm

```
Input: Data chunk set $D = \{d_i | i = 1,2,\ldots,N_d\}$, edge server set $S = \{s_n | n = 1,2,\ldots,N\}$
Output: Optimal cache placement result $\text{HashMap}\langle D,S \rangle$
 1:  $\text{HashMap}\langle D,S \rangle \leftarrow \varnothing$, $\text{iniHashMap}\langle D,S \rangle \leftarrow \varnothing$ // Initialization
 2:  for each $d_i \in D$ do
 3:      Calculate $Pop_i$ // The popularity of data chunk $d_i$
 4:      $TR \leftarrow 0$
 5:      for each $s_n \in S$ do
 6:          Calculate $Rep_n$ // The cache replacement rate of edge server $s_n$
 7:          $R \leftarrow Pop_i / Rep_n$ // The ratio of $Pop_i$ to $Rep_n$
 8:          if $TR < R$ then
 9:              $TR \leftarrow R$, $\text{iniHashMap}\langle D,S \rangle \leftarrow \text{HashMap}\langle d_i, s_n \rangle$ //Cache data chunk $d_i$ on
                 edge server $s_n$ and record the mapping
10:          end if
11:      end for each
12:  end for each
```

```
13:   $S_{initial} \leftarrow CIUV\left(\text{iniHashMap}\langle D,S\rangle\right)$ // Obtain the initial solution of cache placement
14:   $\max CIUV\left(\text{HashMap}\langle D,S\rangle\right) \leftarrow TS\left(S_{initial}\right)$ // Derive the optimal cache placement result
                                                                      by tabu search algorithm
15:   return  $\text{HashMap}\langle D,S\rangle$
```

Algorithm 1 represents the pseudo-code of the integrated utility-based cache placement algorithm.

First, the popularity value of each data chunk is calculated according to equation (9) (Algorithm 1 line 3).

Secondly, the cache placement rate of each edge server is calculated according to equation (8) (Algorithm 1 line 6). Thirdly, the data chunk d_i is cached on edge server s_n with maximal ratio of Pop_i to Rep_n. And the initial mapping is recorded (Algorithm 1 line 5-10).

Then, an initial solution of cache placement is obtained (Algorithm 1 line 13).

Finally, the optimal cache placement result is obtained according to the initial solution by tabu search algorithm (Algorithm 1 line 14).

In Algorithm 1, the time expense of deriving an initial solution is $O\left(N_d \cdot N\right)$ and the time expense of tabu search algorithm to achieve the optimal cache placement result is $O\left(N_{iter} \cdot N_d^2\right)$ where N_{iter} denotes the iteration number of tabu search algorithm, N_d is the number of data chunks and N is the number of edge servers. Hence, the time complexity of the integrated utility-based cache placement algorithm is $O\left(N_{iter} \cdot N_d^2\right)$.

The core of the proposed cache-placement algorithm is tabu search algorithm. Tabu search algorithm is a well-known example of meta-heuristic scheduling techniques [33]. An advantage of tabu search algorithm is that its time complexity is not exponential but polynomial [33], [34]. Thus, the program overhead is relatively negligible [33], [34].

3.3 The cache locality-based task scheduling method

In this paper, both cache locality and disk locality are considered. It is noteworthy that reading data from local cache is faster than reading data from local disk. Reading data from local disk is faster than reading data from remote edge servers. Thus, the cache locality-based task scheduling method is proposed to further reduce the latency. We model the task scheduling problem as a weighted bipartite graph of which weights are mainly derived by locations of required data chunks. During each heartbeat, a maximal weighted matching between tasks and resources are obtained.

3.3.1 Task scheduling model based on weighted bipartite graph

Tasks are handled by containers that are logical bundles of computing resources (such as <1 CPU, 3GB RAM>) bound to edge servers. N_t tasks are scheduled to N_c containers. Let $T=\left\{t_\lambda \middle| \lambda=1,2,\ldots,N_t\right\}$ and $C=\left\{c_\theta \middle| \theta=1,2,\ldots,N_c\right\}$ demote the task set and container set respectively. In addition, we have $N_t \geq N_c$.

A simple graph $G=\left(U,V,e\right)$, whose vertices are divided into two independent sets U and V, is a bipartite graph. e denotes an edge set. Each edge connects a vertex of U with one of V in the graph. If every edge has an associated weight in a bipartite graph, this bipartite graph is a weighted bipartite graph. The task scheduling problem can be modeled as a weighted bipartite graph $G=\left(T,C,E\right)$ where the task set T and the container set C represent vertex sets U and V respectively as shown in Fig. 3. If a task can be handled by a container, there is an edge with a weight between them. Weights are derived according to the similarity between tasks and resources, task scheduling priority and data transmission cost.

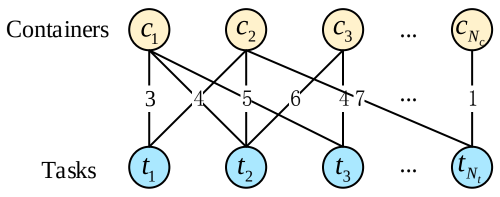


Fig. 3 Task scheduling model

A. Similarity between tasks and resources

Tasks have different requirements of computing resources, such as CPU and memory. A task can be defined as a row vector $t_\lambda = [tc_\lambda, tm_\lambda]$ where tc_λ and tm_λ denote CPU requirement and memory requirement respectively. In the same way, a container can be defined as a row vector $c_\theta = [cpu_\theta, mem_\theta]$ where cpu_β denotes the CPU of container c_θ and mem_θ denotes the memory of container c_θ. Therefore, the similarity between task t_λ and container c_θ can be achieved from (15),

$$sim(t_\lambda, c_\theta) = \frac{t_\lambda (c_\theta)^T}{|t_\lambda| \cdot |c_\theta|} \tag{15}$$

B. Task scheduling priority

The priority value of scheduling task t_λ to container c_θ is determined by the priority of the job including task t_λ and the location of data required by task t_λ .In this paper, a lower priority value means the higher priority. Let jp_λ indicate the priority value of the job which includes task t_λ Let $loc_{\lambda,\theta}$ represent the data location priority value which is measured according to the location of data required by task t_λ as shown in Table 1. For example, if the data required by task t_λ is cached by a server that provides container c_θ, the $loc_{\lambda,\theta}$ is set to b_1

Table 1 Data location priority values

Data locations	Value
Local caches of edge servers	b_1
Local disks of edge servers	b_2
Remote edge servers	b_3

where $b_1 < b_2 < b_3$. Then, the task scheduling priority value $tp_{\lambda,\theta}$ can be achieved from (16),

$$tp_{\lambda,\theta} = jp_\lambda + loc_{\lambda,\theta}. \tag{16}$$

C. Data transmission cost

If the data required by task t_λ is not in the cache or disk of computation node which provides container c_θ when task t_λ is dispatched to container c_θ, the data transmission occurs. The data transmission overhead consists of reading time and energy costs. The energy consumption costs are measured by the power consumed by accessing and transmitting data, such as KWH. The energy consumption of reading a required data chunk for task t_λ includes the local access energy cost, remote access energy cost and data move energy cost [36], [37], [38], which can be achieved as follows,

$$Ec_{\lambda,\theta} = EL_{\lambda,\theta} + ER_{\lambda,\theta} + EM_{\lambda,\theta} \tag{17}$$

where $EL_{\lambda,\theta}$ denotes the energy consumption cost for local access of edge server s_n which provides container c_θ to execute task t_λ, $ER_{\lambda,\theta}$ denotes the average energy consumption cost for remote access of other edge servers and $EM_{\lambda,\theta}$ denotes the minimum data move energy cost to container c_θ to execute task t_λ. If the required data chunk in the local cache or disk of an edge server, we have $ER_{\lambda,\theta} = 0$ and $EM_{\lambda,\theta} = 0$.

The data transmission cost that task t_λ is executed by container c_θ can be defined as follows,

$$tc_{\lambda,\theta} = \begin{cases} 0, \text{ if required data is in local cache or disk} \\ \dfrac{N_r \cdot bs}{band_n} \cdot dis, \text{ otherwise} \end{cases} \tag{18}$$

where N_r denotes the number of required data chunks, dis indicates the minimum distance to get required data chunks and $band_n$ is the network bandwidth of edge server s_n which provides container c_θ. Then, the total data transmission cost can be achieved as follows,

$$cost_{\lambda,\theta} = \varepsilon_1 Ec_{\lambda,\theta} + \varepsilon_2 tc_{\lambda,\theta} \tag{19}$$

where ε_1 and ε_2 denote the adjustment coefficients Moreover, we have $\varepsilon_1 + \varepsilon_2 = 1$.

As mentioned above, edge weights of the weighted bipartite graph can be calculated from (20),

$$ew_{\lambda,\theta} = a_1 \cdot \frac{sim(t_\lambda, c_\theta)}{\overline{sim(t_\lambda, c_\theta)}} - a_2 \cdot \frac{tp_{\lambda,\theta}}{\overline{tp_{\lambda,\theta}}} - a_3 \frac{cost_{\lambda,\theta}}{\overline{cost_{\lambda,\theta}}}, \lambda \in \{1, 2, \ldots, N_t\} \text{ and } \theta \in \{1, 2, \ldots, N_c\} \tag{20}$$

where

$$\begin{cases} \overline{sim(t_\lambda, c_\theta)} = \dfrac{\sum\limits_\lambda \sum\limits_\theta sim(t_\lambda, c_\theta)}{\lambda \cdot \theta} \\ \overline{tp_{\lambda,\theta}} = \dfrac{\sum\limits_\lambda \sum\limits_\theta tp_{\lambda,\theta}}{(\lambda \cdot \theta)} \\ \overline{cost_{\lambda,\theta}} = \dfrac{\sum\limits_\lambda \sum\limits_\theta cost_{\lambda,\theta}}{(\lambda \cdot \theta)} \end{cases} \tag{21}$$

a_1, a_2 and a_3 are the weight coefficients of three influenced factors that can be achieved by a weight coefficient matrix W as shown in (22). The three influenced factors f_1, f_2 and f_3 indicate the similarity between tasks and resources, task scheduling priority and data transmission cost respectively.

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & a_{32} & w_{33} \end{pmatrix} \tag{22}$$

where w_{rc} $(r = 1, 2, 3 \text{ and } c = 1, 2, 3)$ can be calculated as follows,

$$w_{rc} = \begin{cases} 1, \text{ if } f_r \text{ is more important than } f_c \\ 0.5, \text{ if } f_r \text{ is as important as } f_c \\ 0, \text{ otherwise.} \end{cases} \tag{23}$$

Moreover, a_ω $(\omega = 1, 2, 3)$ can be calculated from (24),

$$a_\omega = \frac{\sum\limits_{c=1}^{3} w_{rc}}{\sum\limits_{r=1}^{3} \sum\limits_{c=1}^{3} w_{rc}}. \tag{24}$$

3.3.2 Optimal matching between tasks and computing resources

In order to schedule tasks, the task scheduling problem is formulated as an optimal matching problem in the weighted bipartite graph as follows,

$$\max \sum_{\lambda=1}^{N_t}\sum_{\theta=1}^{N_c} y_{\lambda,\theta} ew_{\lambda,\theta} \tag{25}$$

$$\text{s.t.} \begin{cases} y_{\lambda,\theta} \in \{0,1\}, \forall\lambda \in \{1,2,\ldots,N_t\}, \forall\beta \in \{1,2,\ldots,N_c\} \\ \sum_{\theta=1}^{N_c} y_{\lambda,\theta} = 1, \forall\lambda \in \{1,2,\ldots,N_t\} \end{cases} \tag{26}$$

where

$$y_{\lambda,\theta} = \begin{cases} 1, \text{if task } t_\lambda \text{ is dispatched to container } c_\theta \\ 0, \text{otherwise.} \end{cases} \tag{27}$$

If $N_t > N_c$, $N_t - N_c$ hypothetic containers should be added. The added hypothetic containers are connected with all tasks by edges whose weights are 0. That is to say, the weighted bipartite graph can be regard as a complete weighted bipartite graph $G=(T',C,E)$ where T' denotes the task set which consists of N_c tasks after adding hypothetic tasks.

Definition 1: A real function $l(v)$ is a feasible vertex labeling in $G=(T',C,E)$, for $\forall t_{\lambda'} \in T', c_\theta \in C$ where $\lambda' \in \{1,2,\ldots,N_t,N_{t+1},\ldots,N_c\}$, if

$$l(t_{\lambda'}) + l(c_\theta) \geq ew_{\lambda',c_\theta}. \tag{28}$$

Definition 2: A spanning subgraph $G'=(T',C,E')$ of graph $G=(T',C,E)$ is an equality subgraph, for $\forall e'_{\lambda',\theta} \in E'$, if

$$l(t_{\lambda'}) + l(c_\beta) = ew'_{\lambda',c_\theta}. \tag{29}$$

Theorem 1: The above task scheduling problem shown in (23) has at least one feasible solution.

Proof: Normally, the number of tasks is larger than the number of containers. After adding $N_t - N_c$ hypothetic containers, the task scheduling problem can be regarded as a complete weighted bipartite graph $G=(T',C,E)$. Let $G'=(T',C,E')$ be an equality subgraph of G. Let v' denote a vertex of T', e indicate an edge of E and e' represent an edge of E'. Suppose that G' has a matching M' named perfect matching that matches all vertices. Therefore, we have

$$w(M') = \sum_{e' \in M'} w(e') = \sum_{v' \in T'} l(v).$$

where $w(e)$ is the weight of edge e and $w(M')$ is the sum of all edge weights of perfect matching M'. Suppose that M is a perfect matching of G. We have

$$w(M) = \sum_{e \in M} w(e) \leq \sum_{v' \in T'} l(v') = w(M').$$

Hence, the above task scheduling problem has at least one feasible solution which is a perfect matching of equality subgraphs during each heartbeat.

3.3.3 The cache locality-based task scheduling algorithm

Algorithm 2: Cache locality-based task scheduling algorithm

```
Input: Task set to be scheduled $T=\{t_\lambda|\lambda=1,2,\ldots,N_t\}$, Container set $C=\{c_\theta|\theta=1,2,\ldots,N_c\}$
Output: Task scheduling result Matching$(T,C)$
 1:  for each $t_\lambda \in T$, $c_\theta \in C$ do
 2:      Calculate $sim(t_\lambda,c_\theta)$ // The similarity between task $t_\lambda$ and container $c_\theta$
 3:      Calculate $tp_{\lambda,\theta}$ // The task scheduling priority
 4:      Calculate $Ec_{\lambda,\theta}$ and $tc_{\lambda,\theta}$ // The time and energy consumption costs
 5:      for each $0 \le r \le 3$, $0 \le c \le 3$ do
 6:          Calculate $W$ // The weight coefficient matrix
 7:      end for each
 8:      for each $0 \le \omega \le 3$ do
 9:          Calculate $a_r$ // The weight coefficient
10:      end for each
11:  end for each
12:  for each heartbeat
13:      if $N_t > N_c$ then
14:          Add $N_t - N_c$ hypothetic containers and calculate $ew_{\lambda,\theta}$ // The complete
             weighted bipartite graph during each heartbeat
15:          Repeat
16:              Randomly pick an equality subgraph $G'$ of the complete weighted bipartite
                 graph
17:          Until $G'$ has a perfect matching $(T,C)$
18:      else
19:          Add $N_c - N_t$ hypothetic tasks and record the perfect matching $(T,C)$ of the
             equality subgraph of the complete weighted bipartite graph
20:      end if
21:      Update Task scheduling result $(T,C)$
22:  end for each
23:  return task scheduling result Matching$(T,C)$
```

Algorithm 2 shows the pseudo-code of the cache locality-based task scheduling algorithm.

Firstly, similarities, task scheduling priority values and data transmission costs between tasks and containers are calculated according to equations (15), (16) and (17) respectively (Algorithm 2 line 2-4).

Secondly, weight coefficients are calculated according to the weight coefficient matrix (Algorithm 2 line 5-10).

Then, a complete weighted bipartite graph is constructed and a maximum weighted bipartite matching is obtained during each heartbeat (Algorithm 2 line 12-22).

In Algorithm 2, the time expense of calculating edge weights of a bipartite graph is $O(N_t \cdot N_c)$ where N_t is the number of tasks to be scheduled and N_c is the number of containers. After adding $N_t - N_c$ hypothetic containers in a heartbeat interval, the time expense to achieve a perfect matching of equality subgraphs of a complete weighted bipartite graph is $O(N_t^3)$. Therefore, the time complexity of Algorithm 2 is $O(\lceil (N_t / N_c) \rceil \cdot N_t^3)$.

Compared with the time expense of task scheduling, the time spent on constructing a weighted bipartite graph can be ignored. The time complexity of the task scheduling algorithm is with the polynomial time $O\left(\lceil (N_t / N_c) \rceil \cdot N_t^3\right)$. The polynomial time complexity makes the algorithm suitable option for the task scheduling [22], [35]. Since the time complexities have no exponential terms, the time overhead of the proposed algorithms is relatively negligible.

4 Performance evaluation

4.1 Experimental environment: smart campus

With the rapid development of edge computing, computer vision and IoT, the traditional education is changing. Modern campus is emerging by applying these new techniques, which improves the utilization of available campus resources, heighten the work efficiency of both students and teachers [39], [40], [41]. Smart campus is a novel and significant application of edge computing, which mainly includes smart class, augmented reality assisted mobile campus and smart lab [42], [43]. The smart class changes the way students learn.

Compared with traditional classes, smart classes make students main and active roles. First, teachers and students log in a smart class system by face recognition techniques. Second, students select the teaching contents they favor. When they finish the classes, they can submit their speeches or presentations which are stored, retrieved and analyzed at edge computing servers. Finally, teachers don’t need to record the presentations for specific students who use the smart class system. The contents used in the smart class mainly consist of videos and audios. Moreover, these contents are requested by students repeatedly in particular buildings. Reading contents from cloud is costly and time-consuming. Therefore, it is significant to deploy edge servers in the buildings of the campus and cache popular contents in these edge servers.

The smart campus system mainly consists of an edge orchestrator, edge servers and campus network, as shown in Fig. 4. Popular teaching contents are cached in the edge servers which are deployed in the buildings, such as library, teaching buildings and dormitories. Students and teachers connect the edge servers by access points. The edge orchestrator that manages and controls all edge servers acts as a master node. The edge orchestrator and all edge servers communicate with each other by campus network. And the entire campus network can also be connected to the Internet.

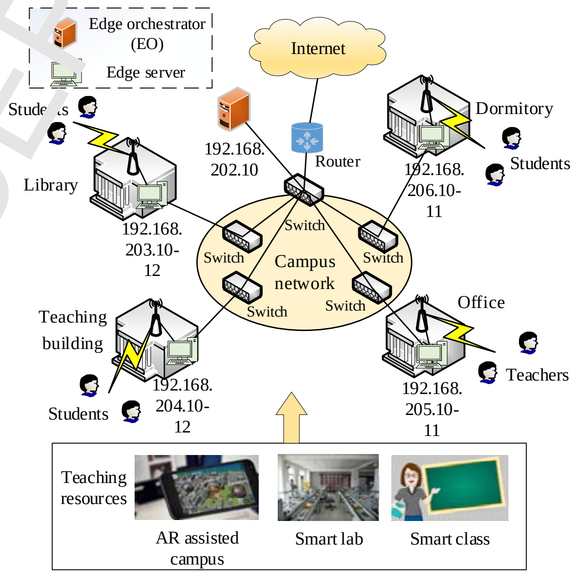


Fig. 4 Experimental environment

The cache-aware task scheduling method is experimentally verified. The experimental environment is shown in Fig. 4. 1 edge orchestrator and 10 edge servers with different configurations are installed. These 10 edge servers are deployed in 10 different buildings in our campus. The edge orchestrator is composed of an Intel Core i5-4590 CPU running at 3.30 GHz and 4GB RAM. Each edge server that acts as a slave node is empowered with an Intel Core i5-4590 CPU running at 3.30 GHz. Slave 1-Slave 3 are configured with 4GB RAM. Slave 4-Slave 6 are composed of 6GB RAM. And Slave 7-Slave 10 are configured with 8GB RAM.

The network bandwidth of the campus network is 30Mbps. The model number of network switches is ALCATEL Omnicore5022. The operating system is Linux Ubuntu 14.04.1 LTS, The Hadoop 2.7.1 is adopted to perform the experiments, the version of the Java Development Kit is JDK 1.7.0_45 and the development environment is MyEclipse 10. The Huawei AP3010DN-AGN is adopted as the WiFi access point in each building. End devices use IEEE 802.11 radios to connect WIFI access points which are connected to local edge servers.

The configurations of edge orchestrator and edge servers are shown as follows.

Table2 Configurations of edge servers

Host name	Configuration	IP
Master (EO)	Intel(R)Core(TM) i5-4590CPU 3.30GHz RAM: 4G	192.168.202.10
Slave1-Slve3 (Edge servers)	Intel(R)Core(TM) i5-4590CPU 3.30GHz RAM: 4G	192.168.203.10 192.168.203.11 192.168.203.12
Slave4-Slve6 (Edge servers)	Intel(R)Core(TM) i5-4590CPU 3.30GHz RAM: 6G	192.168.204.10 192.168.204.11 192.168.204.12
Slave7-Slve8 (Edge servers)	Intel(R)Core(TM) i5-4590CPU 3.30GHz RAM: 8G	192.168.205.10 192.168.205.11
Slave9-Slve10 (Edge servers)	Intel(R)Core(TM) i5-4590CPU 3.30GHz RAM: 8G	192.168.206.10 192.168.206.11

The experimental data set including text data about 96 million blog posts of *Memetracker* comes from Stanford Network Analysis Project (SNAP) [30]. The data set named *memetracker9* includes *Memetracker* phrases and hyperlinks between 96 million blog posts from Aug 2008 to Apr 2009 [30]. This data set consists of the real-world text data. Thus, it is general enough to conclude. This data set contains 9 files. The total size of these files is 13.36GB. In the Hadoop Distributed File System (HDFS), the default setting of the data chunk size is 64 MB. Thus, the size of each data chunk is set as 64MB. In order to evaluate the performance of the proposed task scheduling algorithm and other task scheduling algorithms, we adopted WordCount as a benchmark job, which counts the characters from above data set.

The detailed testing parameters are listed in Table 3.

Table 3 Testing parameters

Parameter name	Value
The data location priority value b_1	10
The data location priority value b_2	20

The data location priority value b_3	30
The number of edge orchestrators	1
The number of edge servers	10
The data size of each data chunk	64MB
The average cache size of edge servers	$[25,55]$
The number of data chunks to be read	$[16,256]$
The number of data chunks to be cached	100
Popularity of data chunks	$[0.5,0.9]$
The number of tasks	$[16,256]$
The size of each task	64MB

In table 3, the amounts of some parameters are determined according to the settings of the software and hardware. In the Hadoop Distributed File System (HDFS), the default setting of the data chunk size is 64 MB. The default setting of the task size is same as the data chunk size. Thus, the sizes of both data chunks and tasks were set to 64MB.

The numbers of edge orchestrators and servers are determined according to hardware requirements and real-life environment. In edge computing framework, the mode including 1 edge orchestrator and *N* edge servers is generally applied at system level [5], [6], [8], [47]. We also apply this mode. Thus, 1 edge orchestrator is required. In the smart campus scene which is a real-word experiment environment, 10 edge servers are deployed to cover most of students on campus. These edge servers are deployed in library, teaching buildings and dormitories. It is available that 10 edge servers are deployed.

According to [45] and [46], the number of tasks is set to the multiples of 10 or 20 to evaluate the performance of algorithms. And the maximum number of tasks is set to the square of these numbers. And the maximum number of tasks is set to the square of these numbers. We follow these settings. The numbers of both tasks and data chunks are set to the multiples of 16. The maximum value is set to 256.

The average cache size of each edge server is set to 55 at most. The number of data chunks to be cached is set to 100. These data chunks should be cached in 2 edge servers at least and may incur the eviction in edge servers. In addition, too many cached data chunks will weaken the impacts of different cache placement algorithms. The data location priority values and popularity values of data chunks are defined by equations (9) and (16). Others are obtained according to the relevant references [10], [13].

There are two main parts in experiments of verifying the effectiveness of cache-aware task scheduling algorithm. First, the integrated utility-based cache placement algorithm is compared with the centralized cache management mechanism Hadoop provides and *CP-Dynamic* algorithm [31] in terms of cache service ratio, data reading time and cache replacement number. Then, the cache locality-based task scheduling algorithm is compared with First In First Out (FIFO) scheduling algorithm, *D-LAWS* algorithm [32] and *CATS* algorithm [20] in terms of cache hit ratio, data locality ratio, data transmission time, task response time and energy cost. Since FIFO scheduling algorithm, *D-LAWS* algorithm and *CATS* algorithm are the algorithms with same objectives as the proposed cache locality-based task scheduling algorithm, these three algorithms are selected as comparison algorithms.

In verification experiments of the integrated utility-based cache placement algorithm, cache service ratio, data reading time and average replacement number are given to evaluate the algorithm performance. The cache service ratio CSR represents the ratio of CRN that denotes the number of requested data chunks that are cached on edge servers to RN which represents the total number of requested data chunks as shown in (30),

$$CSR = \frac{CRN}{RN} \tag{30}$$

The data reading time denotes the overall time for reading data chunks. The cache replacement number denotes the total replacement number of cached data chunks.

In verification experiments of the cache locality-based task scheduling algorithm, cache hit ratio, data locality, data transmission time, task response time and energy cost are given to evaluate the algorithm performance. The cache hit ratio CHR denotes the ratio of LCN that denotes the number of tasks whose required data chunks are in local cache to N_t which indicates the total number of tasks. The cache hit ratio can be calculated as depicted in (31)

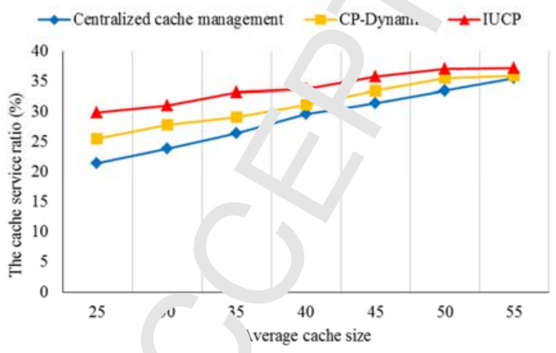
$$CHR = \frac{LCN}{N_t} \tag{31}$$

The data locality represents the ratio of the number of tasks whose required data chunks are stored in local disk or in local cache to the total number of tasks. When the required data chunks for tasks are stored in remote edge servers, the data transmission will occur. The data transmission time denotes the total time for transmitting the data chunks which must be transmitted. The task response time is the time interval from the task submission time to the finish time of the last task. The energy cost is the energy consumption for task execution.

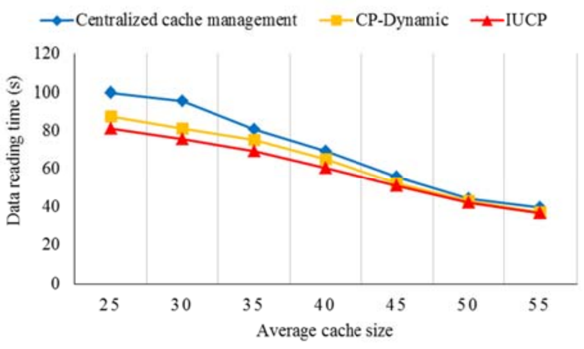
4.2 Comparison and analysis

4.2.1 Performance evaluation of integrated utility-based cache placement algorithm

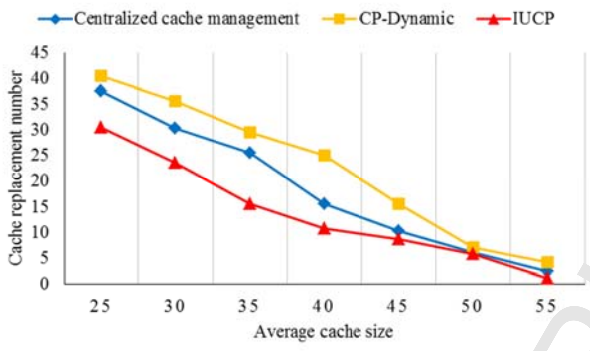
In this section, the performance evaluation results of the integrated utility-based cache placement (*IUCP*) algorithm and the other two benchmark algorithms, the centralized cache management mechanism Hadoop provides and *CP-Dynamic* algorithm [31], are evaluated in terms of the cache service ratio, data reading time and cache replacement number. In order to validate the effectiveness of cache placement algorithms, each experiment has been executed for 12 times and the average values of evaluation metrics are given to improve the reliability of experiment results. How the performance of each algorithm is affected by various parameters is analyzed by setting different parameter values.



(a)



(b)



(c)

Fig. 5 Impacts of different average cache sizes on (a) the cache service ratio, (b) the data reading time, (c) the cache replacement number

In this subpart, the size of each data chunk is 64MB. The maximum number of data chunks that can be cached on an edge server denotes the cache size of this edge server. After caching 100 data chunks according to historical running data by mentioned three algorithms, 160 data chunks will be read randomly to validate the algorithm performances. The historical running data includes the numbers of requests for data chunks, the time that data chunks are first requested, the current time and request frequencies of data chunks. The mentioned historical running data is used to achieve popularity values of data chunks according to equation (9).

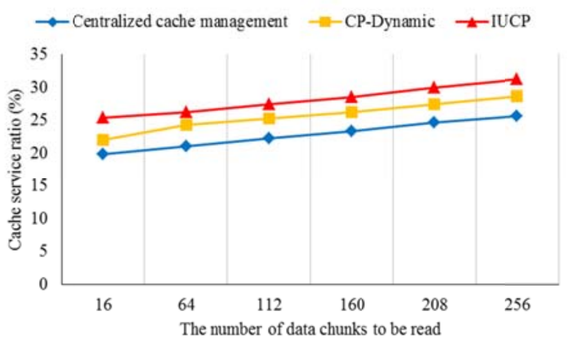
Fig. 5 depicts how the average cache size of all edge servers affects the cache service ratio, data reading time and cache placement number by varying the value from 25 to 55.

As shown in Fig. 5 (a), the cache service ratios of *IUCP* algorithm, centralized cache management mechanism and *CP-Dynamic* algorithm all increase when the average cache size of all edge servers increases from 25 to 55. This is because, with the increase of average cache size, more and more data chunks are cached on edge servers. When the average cache size is set to 25, compared with centralized cache management mechanism and *CP-Dynamic* algorithm, *IUCP* algorithm increases cache service ratio up to 39.07% and 17.25% respectively.

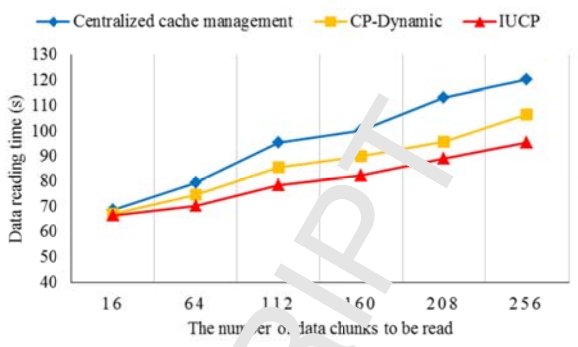
Fig. 5 (b) depicts that the data reading time acts as a decreasing function of average cache size in all algorithms. The reason is that reading data from local cache is faster than reading data from local disk or remote edge servers. When the average cache size is 25, *IUCP* algorithm compared with centralized cache management mechanism and *CP-Dynamic* algorithm can reduce the data reading time up to 18.46% and 7.19% respectively.

Fig. 5 (c) shows the cache replacement number as a decreasing function of average cache size for these three algorithms. Under the same condition of the average cache size which is set to 25, *IUCP* algorithm leads to a reduction of 18.62% over the centralized cache management mechanism and a 24.63% reduction over the *CP-Dynamic* algorithm in terms of the cache replacement number. This is because the cache placement penalty is a main factor of *IUCP* algorithm.

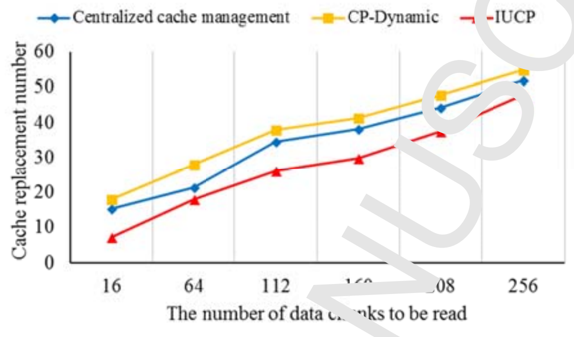
It is obvious that there are not a lot of differences in terms of the cache service ratio, data reading time and cache replacement number among the three algorithms when the average cache size is 55. The large average cache size of edge servers leads to caching more data chunks. Hence, the data chunks to be read are in the caches with more probability.



(a)



(b)



(c)

Fig. 6 Impacts of different numbers of data chunks to be read on (a) the cache service ratio, (b) the data reading time, (c) the cache replacement number

In this subpart, the default settings are as follows. The average cache size of all edge servers is set to 25, the size of each data chunk is 64MB and the number of data chunks which are cached on edge servers by *IUCP* algorithm, centralized cache management mechanism and *CP-Dynamic* algorithm is set to 100. Then, how the number of data chunks to be read affects cache service ratio and data reading time by varying the value from 16 to 256 is analyzed. All of data chunks are read randomly.

Fig. 6 (a) illustrates how the number of data chunks to be read affects the cache service ratio by varying the value from 16 to 256. It is obvious that the cache service ratio acts as an increasing function of the number of data chunks to be read among three algorithms. Because the number of data chunks that are read randomly increases, the probability that these data chunks are cached on edge servers also increases. When the number of data chunks to be read is 256, *IUCP* algorithm compared with the centralized cache management mechanism and *CP-Dynamic* algorithm increases the cache service ratio up to 21.88% and 8.71% respectively.

Then, we investigate how different numbers of data chunks to be read affects the data reading time as shown in Fig. 6 (b). As the number of data chunks which are read randomly increases, the data reading time increases significantly among all three algorithms. The differences among three algorithms are not so great when the number of data chunks to be read is 16. When the number of data chunks to be read increase to 256, *IUCP* algorithm leads to a 20.76% reduction over centralized cache management mechanism and a 10.59% reduction over *CP-Dynamic* algorithm in terms of data reading time. *IUCP* algorithm considers both data popularity and cache replacement rate of edge servers, which leads to a longer time to cache popular data chunks.

As shown in Fig. 6 (c), the cache replacement number shows as an increasing function of the number of data chunks to be read from 64 to 256. The reason is that data chunks cached on edge servers will be replaced when some new data chunks are read. The performance of *IUCP* algorithm is always better than other two algorithms. This is because the replacement penalty is a main factor of *IUCP* algorithm to cache new data chunks.

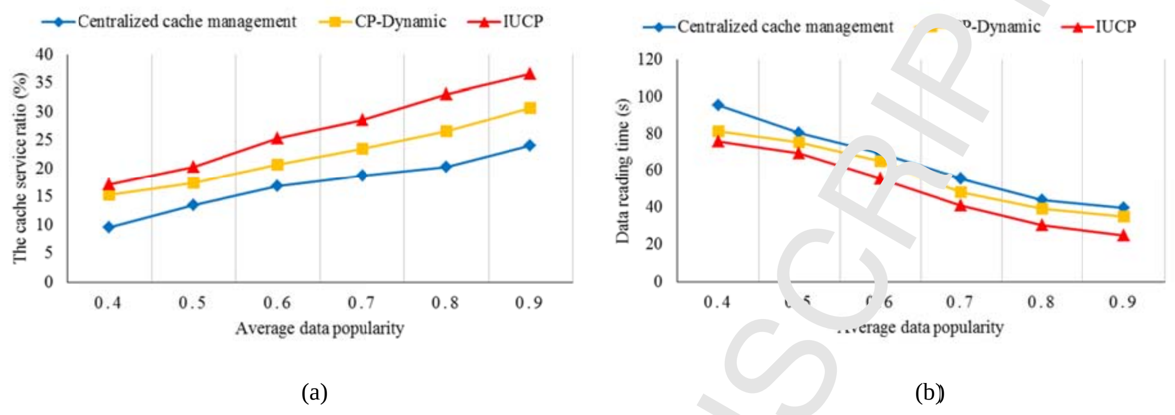


Fig. 7 Impacts of different average data popularity on (a) the cache service ratio, (b) the data reading time

The default settings of this part are shown as follows: the average cache size of all edge servers is 25, the number of data chunks cached by *IUCP* algorithm, centralized cache management mechanism and *CP-Dynamic* algorithm is 100, and the size of each data chunk is 64MB. Then, 160 data chunks are read randomly. How the average data popularity of all data chunks to be read affects the cache service ratio and data reading time via varying the value from 0.4 to 0.9.

Fig. 7 (a) depicts the cache service ratio as an increasing function of the average data popularity. It is obvious that the performance of *IUCP* algorithm is better than others all the way. This is because, *IUCP* algorithm considers both the data popularity and edge server cache placement rate, which leads to caching data chunks with the high data popularity for a longer time. When the average data popularity is set to 0.9, *IUCP* algorithm compared with the centralized cache management mechanism and *CP-Dynamic* algorithm improves the cache service ratio up to 52.28% and 19.16% respectively.

As observed from Fig. 7 (b), when the average data popularity increases from 0.4 to 0.9, the data reading time reduces dramatically among all three algorithms. Compared with the centralized cache management mechanism and *CP-Dynamic* algorithm, *IUCP* algorithm reduces the data reading time up to 37.78% and 30.03% when the average data popularity is set to 0.9 respectively. The reason is that the *IUCP* algorithm can keep caching data chunks with high data popularity values for a longer time.

4.2.2 Performance evaluation of cache locality-based task scheduling algorithm

In this section, the performance evaluation results of the cache locality-based task scheduling (*CLTS*) algorithm and the other three benchmark algorithms, First in First out (FIFO) scheduling algorithm, *D-LAWS* algorithm [32] and *CATS* algorithm [20], are investigated in terms of cache hit ratio, data locality, data transmission time, task response time and energy cost. When the system keeps stable after a period of operation, lots of data chunks are cached in the system. For the purpose of guaranteeing authenticity of all experiments, each experiment has been executed for 12 times and average values of evaluation metrics are calculated. How the performance of each scheduling algorithm is affected by various parameter settings is studied.

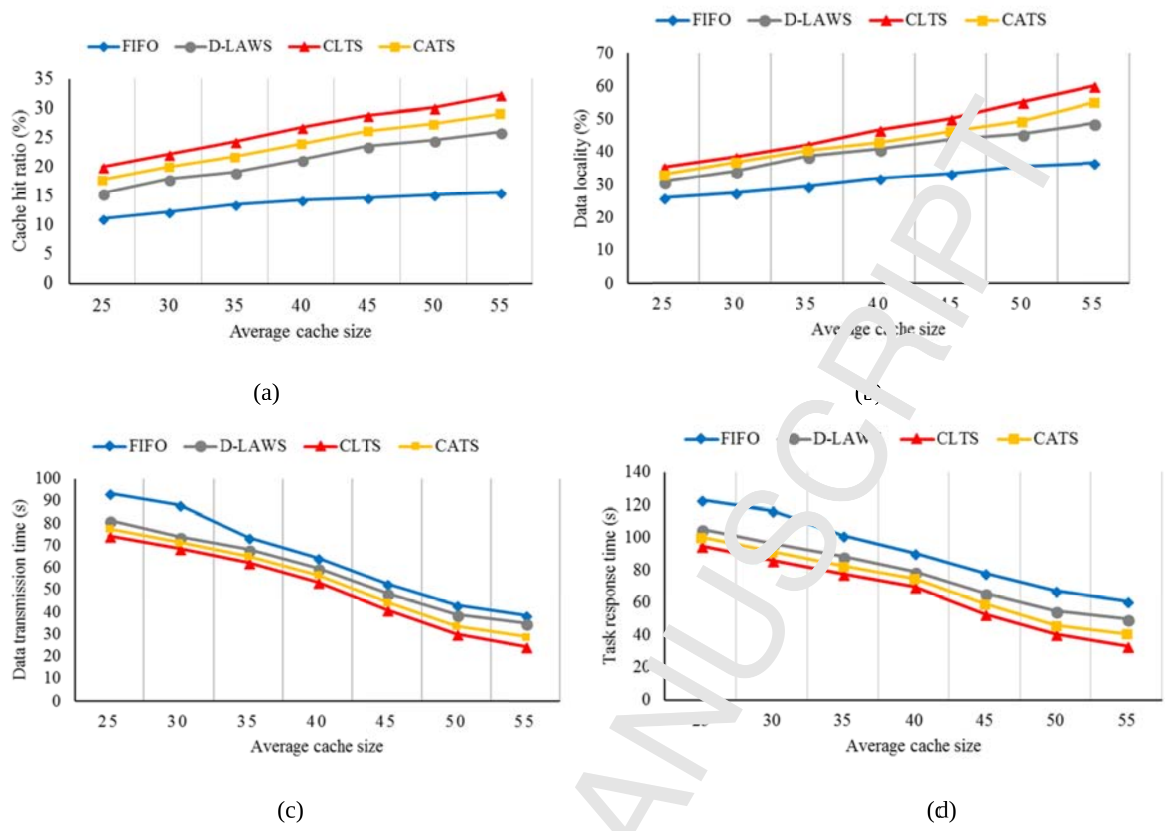


Fig. 8 Impacts of different average cache sizes on (a) the cache hit ratio, (b) the data locality, (c) the data transmission time, (d) the task response time

In this subpart of task scheduling experiments, we randomly choose 160 tasks to be executed. There are lots of cached data chunks when the system keeps stable after a period of operation. The maximum number of data chunks that can be cached on an edge server denotes the cache size of this edge server. Then, how the average cache size affects the mentioned evaluation metrics via varying the value from 25 to 55 is investigated.

Fig. 8 (a) shows the cache hit ratio as an increasing function of the average cache size from 25 to 55 for all four algorithms. This is because more data chunks are cached when the average cache size increases. The performance of *CLTS* algorithm is better than that of the other three scheduling algorithms all the time. The reason is that *CLTS* algorithm can keep data chunks with high popularity values for a longer time.

Then, we study how the average cache size affects the data locality by varying the value from 25 to 55. Fig. 8 (b) depicts the experimental results. It is obvious that the data locality acts as an increasing function of the average cache size in all algorithms. When the average cache size of edge servers is set to 55, compared with FIFO scheduling algorithm, *D-LAWS* algorithm and *CATS* algorithm, the *CLTS* algorithm can increase the data locality up to 63.76%, 16.59% and 10.34% respectively.

As observed from Fig. 8 (c), the data transmission time shows a decreasing function of the average cache size in all three algorithms when the average cache size increases from 25 to 55. More and more data chunks that are required for executing tasks frequently are cached, which leads to few data transfers. When the average cache size is 55, *CLTS* algorithm, compared with FIFO scheduling algorithm, *D-LAWS* algorithm and *CATS* algorithm, reduces the data transmission time up to 36.18%, 30.02% and 16.33% respectively.

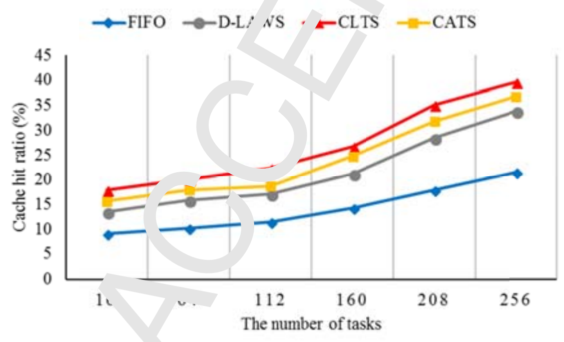
Finally, how the average cache size affects the task response time is analyzed, as shown in Fig. 8 (d). It is obvious that the task response time of all three algorithms shows a decreasing function of the average cache size. When the average cache size is set to 55, *CLTS* algorithm leads to a 45.57% reduction over FIFO scheduling algorithm, a 33.47% reduction over *D-LAWS* algorithm and a 18.55% reduction over *CATS* algorithm. The reason is that the *CLTS* algorithm makes full use of the cached data chunks and dispatch tasks to optimal resources.

In this part of experiments, the default settings of task scheduling experiments are as follows. According to references [10], [13] and [14], the ratio of cache size to total storage size is set from 25% to 85%. Based on the hardware configurations and above ratios, the average cache size of all edge servers is set to integral multiples of data chunk size, which is from 25 to 55. Moreover, 40 is the median of the average cache size interval. Thus, the average cache size of all edge servers is set to 40. Then, we study how the number of tasks affects the cache hit ratio, the data locality, the data transmission time and the task response time by varying the value from 16 to 256.

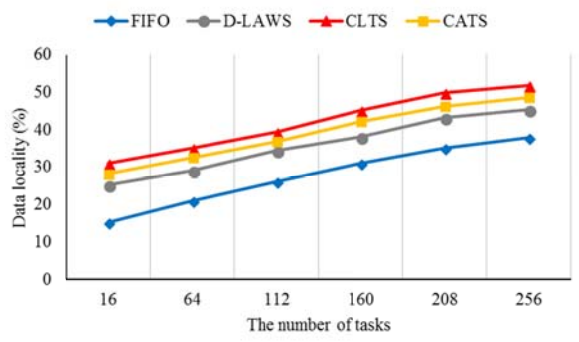
Fig. 9 (a) depicts the cache hit ratio under the different number of tasks. The cache hit ratio acts as an increasing of the number of tasks in all algorithms. The reason is that more cached data chunks are utilized when the number of tasks increases. As observed from Fig. 9 (b), the data locality shows as an increasing function of the number of tasks. When the number of tasks is 160, compared with FIFO scheduling algorithm, *D-LAWS* algorithm and *CATS* algorithm, *CLTS* algorithm increases the data locality up to 45.34%, 18.64% and 7.62% respectively.

Fig. 9 (c) shows that the data transmission time acts as an increasing function of the number of tasks. When the number of tasks is 16, there are few performance differences among three algorithms. This is because the data transmission seldom occurs when the number of tasks is small. When the number of tasks is 256, *CLTS* algorithm leads to a 21.43% reduction over FIFO scheduling algorithm, a 14.24% reduction over *D-LAWS* algorithm and a 7.52% reduction over *CATS* scheduling algorithm in terms of the data transmission time.

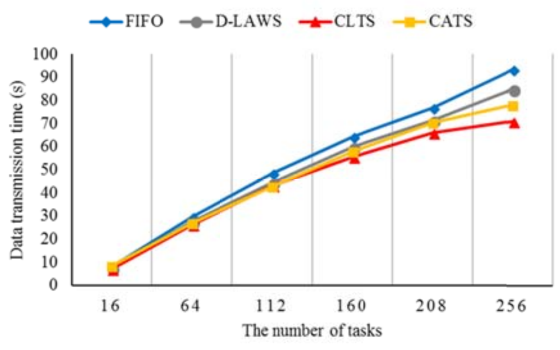
Finally, we investigate how the number of tasks affects the task response time by varying its value from 16 to 256 in Fig. 9 (d). The task response time acts as an increasing function of the number of tasks for all algorithms. The reason is that more time is needed to process more tasks. There are few performance differences to deal with 16 tasks among all algorithms. Compared with FIFO scheduling algorithm, *D-LAWS* algorithm and *CATS* algorithm, *CLTS* algorithm reduces the task response time up to 29.73%, 17.59% and 11.24% respectively when the number of tasks is 256.



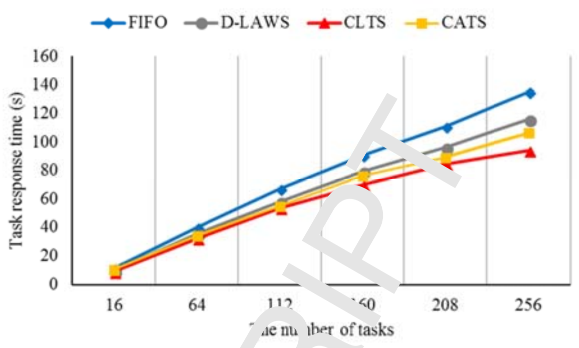
(a)



(b)

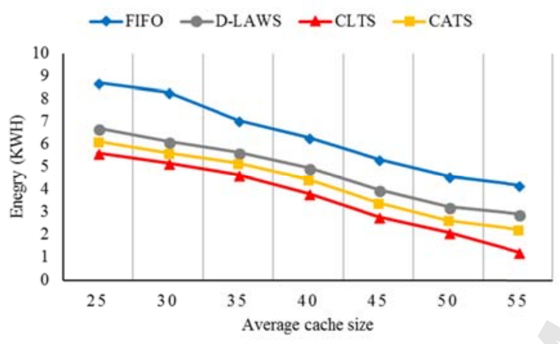


(c)

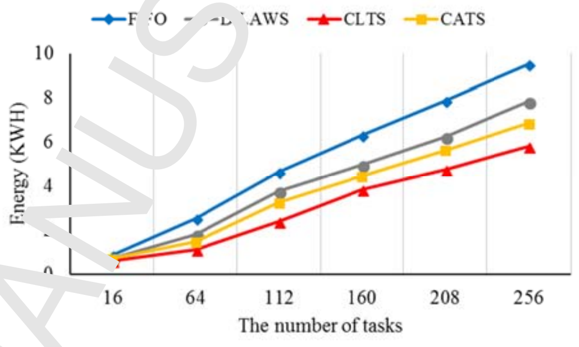


(d)

Fig. 9 Impacts of different numbers of tasks on (a) the cache hit ratio, (b) the data locality, (c) the data transmission time, (d) the task response time



(a)



(b)

Fig. 10 The energy costs of the whole system

Fig. 10 shows the energy costs of whole system with various average cache sizes and numbers of tasks. In Fig. 10 (a), the energy costs act as decreasing functions of average cache sizes. The reason is that more data chunks are cached in edge servers with larger average cache sizes, which reduces the data transmission costs. Compared with FIFO scheduling algorithm, *D-LAWS* algorithm and *CATS* algorithm, *CLTS* algorithm reduces the energy consumption up to 35.51%, 16.37% and 8.11% when the average cache size is set to 25.

As observed from Fig. 10 (b), the energy costs of all algorithms increase dramatically when the number of tasks increases. This is because more data chunks are required to process these tasks. When the number of tasks is 256, the *CLTS* algorithm leads to a 39.19% reduction over FIFO scheduling algorithm, a 26.22% reduction over *D-LAWS* algorithm and a 15.39 reduction over *CATS* algorithm

4.2.3 Sensitivity analysis

The sensitivity analysis can provide the details about how the proposed algorithms behave with varying parameters. The correlation analysis is a well-known technique to analyze the sensitivity [44]. The Pearson correlation analysis is applied to indicate the relationships among the parameters. The correlation coefficient can be achieved as follows,

$$r_{xy} = \frac{n\left(\sum xy\right)-\left(\sum x\right)\left(\sum y\right)}{\sqrt{\left[n\sum x^2-\left(\sum x\right)^2\right]}\sqrt{\left[n\sum y^2-\left(\sum y\right)^2\right]}} \tag{32}$$

In equation (32), x and y denote the experimental data. The number of data is n. The value of Pearson correlation coefficient is between -1 and 1, where the positive value indicates positive correlation and negative value means negative correlation. In addition, there is no correlation when the value is 0. The correlation coefficients among the mentioned parameters are shown in the following tables.

Table 4 Pearson correlation coefficients for the integrated utility-based cache placement algorithm

Parameters	a_1	a_2	a_3
b_1	0.936	-0.815	-0.837
b_2	0.947	0.743	0.891
b_3	-0.822	0.828	N/A

Table 4 shows the Pearson correlation coefficients of the parameters for the integrated utility-based cache placement algorithm. a_1, a_2 and a_3 denote the cache service ratio, data reading time and cache replacement number respectively. b_1, b_2 and b_3 denote the average cache size, the number of data chunks to be read and average data popularity respectively. Strong correlations among parameter pairs are shown in Table 4. Compared with the correlation between a_2 and b_2, the correlation between a_1 and b_2 is stronger. More data chunks are required when the number of data chunks to be read increases. Thus, the cache service ratio increases. The data reading time is mainly affected by not only the number of data chunks to be read but also data locations. The relationship between a_3 and b_1 is the strongest negative correlation among all parameter pairs. The reason is that the number of data chunks cached in edge servers is greater when the average cache size is larger, which leads to the smaller cache replacement number.

Table 5 Pearson correlation coefficients for the cache locality-based task scheduling algorithm

Parameters	c_1	c_2	c_3	c_4	c_5
d_1	0.958	0.857	-0.813	-0.687	-0.744
d_2	0.972	0.894	0.821	0.719	0.772

Table 5 shows the Pearson correlation coefficients of the parameters for the integrated utility-based cache placement algorithm. c_1, c_2, c_3, c_4 and c_5 denote the cache hit ratio, data locality, data transmission time, task response time and energy consumption respectively. d_1 and d_2 are average cache size and the number of tasks. All the absolute values of Pearson correlation coefficients are greater than 0.6 in Table 5. Thus, the Pearson correlation coefficients indicate that the parameter pairs selected in experiments are strongly correlated. Compared with the positive correlation between c_1 and d_2, the positive correlation between c_4 and d_2 is weaker. The relationship between c_1 and d_2 is the strongest positive correlation. This is because task response time is achieved by not only data reading time but also task execution time. In addition, more data chunks are required to process more tasks, which increases the cache hit ratio significantly. The correlation of -0.687 between c_4 and d_1 is the weakest negative correlation. The reason is that the average cache size mainly affects data reading time which is a portion of task response time.

4.3 Experiment Summary

IUCP algorithm is compared with the centralized cache management mechanism Hadoop provides and *CP-Dynamic* algorithm. Extensive experiments show that *IUCP* algorithm can improve the cache service ratio and reduce the data reading time and cache replacement number effectively. Moreover, *CLTS* algorithm is evaluated by comparing it with FIFO scheduling algorithm, *D-LAWS* algorithm and *CATS* algorithm. Many experiments indicate that *CLTS* algorithm can improve the cache hit ratio and data locality and reduce the data transmission time, task response time and energy consumption costs significantly.

5 Conclusion

In this paper, we propose a cache-aware task scheduling method in edge computing. First, the integrated utility-based cache placement strategy is presented. The data chunks are cached at optimal edge servers to maximize the integrated utility value of caching. Then, tasks are scheduled according to the cache placement results. We model the task scheduling problem as a weighted bipartite graph of which weights are mainly derived by the location of required data, such as local cache, local disk or remote one. During each heartbeat, maximal weighted matching between tasks and resources are obtained. The proposed algorithms with polynomial time complexities are appropriate in edge computing. Furthermore, extensive experiments show that the cache-aware task scheduling algorithm outperforms other baseline algorithms in terms of cache hit ratio, data locality, data transmission time, task response time and energy consumption costs.

Acknowledgment

The work was supported by the National Natural Science Foundation (NSF) under grants (No.61672397, No. 61873341, No.61472294, No.61771354), Application Foundation Frontier Project of WuHan (No. 2018010401011290). Fundamental Research Funds for the Central Universities (WUT No.2017-YB-029), Open Fund of Key Laboratory of Geographic Information Science (Ministry of Education), East China Normal University (Grant No. KLGIS2017A01), The Project Supported by the Open Fund of Key Laboratory of Urban Land Resources Monitoring and Simulation，Ministry of Land and Resources (Grant No. KF-2018-03-005), Beijing Intelligent Logistics System Collaborative Innovation Center Open Project (No.BILSCIC-2018KF-02), Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the above agencies.

References

[1] Huang G, Liu X, Lu X, et al. Programming Situational Mobile Web Applications with Cloud-Mobile Convergence: An Internetware-Oriented Approach. IEEE Transactions on Services Computing, 2016, PP(99), 1-1.

[2] Wang W, Xu P, Yang L T, et al. Cloud-Assisted Key Distribution in Batch for Secure Real-time Mobile Services. IEEE Transactions on Services Computing, 2016, 11(5): 850 - 863.

[3] Abbas N, Zhang Y, Taherkordi A, et al. Mobile Edge Computing: A Survey. IEEE Internet of Things Journal, 2018, 5(1): 450 - 465.

[4] Baktir A C, Ozgovde A, Ersoy C. How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases and Future Directions. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2359 - 2391.

[5] Dolui K , Datta S K . Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. Global Internet of Things Summit. IEEE, 2017.

[6] Mach P, Becvar Z. Mobile edge computing: A survey on architecture and computation offloading. IEEE Communications Surveys & Tutorials, 2017, 19(3): 1628 - 1656.

[7] Shi W, Cao J, Zhang Q, et al. Edge computing: Vision and challenges. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[8] Taleb T, Dutta S, Ksentini A, et al. Mobile edge computing potential in making cities smarter. IEEE Communications Magazine, 2017, 55(3): 38-43.

[9] Rimal B P, Van D P, Maier M. Mobile edge computing empowered fiber-wireless access networks in the 5G era. IEEE Communications Magazine, 2017, 55(2): 192-200.

[10] 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.

[11] Liu J, Bai B, Zhang J, et al. Cache Placement in Fog-RANs: From Centralized to Distributed Algorithms. IEEE Transactions on Wireless Communications, 2017, to be publilised

[12] Al-Turjman F M. Cognitive Caching for the future sensors in Fog networking. Pervasive and Mobile Computing, 2017, 42: 317-334.

[13] 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.

[14] Wang X, Chen M, Taleb T, et al. Cache in the air: exploiting content caching and delivery techniques for 5G systems. IEEE Communications Magazine, 2014, 52(2):131-139.

[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.

[16] 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.

[17] 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.

[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] 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.

[20] Lim B, Kim J W, Chung Y D. CATS: cache-aware task scheduling for Hadoop-based systems. Cluster Computing, 2017, 20(4): 3691–3705.

[21] Li G, Li X, Yang F, et al. Traffic at-a-glance: Time-bounded analytics on large visual traffic data. IEEE Transactions on Parallel and Distributed Systems, 2017.

[22] Dai X, Wang X, Liu N. Optimal scheduling of data-intensive applications in cloud-based video distribution services. IEEE Transactions on Circuits and Systems for Video Technology, 2017, 27(1): 73-83.

[23] Chen Q, Yao J, Li B, et al. PISCES: Optimizing Multi-job Application Execution in MapReduce. IEEE Transactions on Cloud Computing, 2016, PP(99):1-1.

[24] Tanaka M, Tatebe O. Disk cache-aware task scheduling for data-intensive and many-task workflow. 2014 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2014: 167-175.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Larumbe, Federico, and Brunilde Sanso. "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.

[30] SNAP website: http://snap.stanford.edu/data/index, Accessed 24 June 2017

[31] 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.

[32] Choi J, Adufu T, Kim Y. Data-locality aware scientific workflow scheduling methods in HPC cloud environments. International Journal of Parallel Programming, 2017: 1-14.

[33] Zhan Z H, Liu X F, Gong Y J, et al. Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Computing Surveys (CSUR), 2015, 47(4): 63.

[34] Youssef H, Sait S M, Adiche H. Evolutionary algorithms, simulated annealing and tabu search: a comparative study. Engineering Applications of Artificial Intelligence, 2001, 14(2): 167-181.

[35] Abrishami S, Naghibzadeh M, Epema D H J. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Generation Computer Systems, 2013, 29(1): 158-169.

[36] Qiu M, Chen Z, Ming Z, et al. Energy-aware data allocation with hybrid memory for mobile cloud systems. IEEE Systems Journal, 2017, 11(2): 813-822.

[37] Baker T, Al-Dawsari B, Tawfik H, et al. GreeDi: An energy efficient routing algorithm for big data on cloud. Ad Hoc Networks, 2015, 35: 83-96.

[38] Jalali F, Hinton K, Ayre R, et al. Fog computing may help to save energy in cloud computing. IEEE Journal on Selected Areas in Communications, 2016, 34(5): 1728-1739.

[39] Trinh H, Chemodanov D, Yao S, et al. Energy-Aware Mobile Edge Computing for Low-Latency Visual Data Processing. IEEE International Conference on Future Internet of Things and Cloud. IEEE Computer Society, 2017:128-133.

[40] Qi B, Kang L, Banerjee S. A vehicle-based edge computing platform for transit and human mobility analytics. ACM/IEEE Symposium. ACM, 2017:1-14.

[41] Shahidehpour M, Li Z, Ganji M. Smart Cities for a Sustainable Urbanization: Illuminating the Need for Establishing Smart Urban Infrastructures. IEEE Electrification Magazine, 2018, 6(2):16-33.

[42] Liu Y, Shou G, Hu Y, et al. Towards a smart campus: Innovative applications with WiCloud platform based on mobile edge computing. 2017 12th International Conference on Computer Science and Education (ICCSE). IEEE, 2017: 133-138.

[43] Hentschel K, Jacob D, Singer J, et al. Supersensors: Raspberry Pi devices for smart campus infrastructure. 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud). IEEE, 2016: 58-62.

[44] Matos R, Araujo J, Oliveira D, et al. Sensitivity analysis of a hierarchical model of mobile cloud

computing. Simulation Modelling Practice and Theory, 2015, 50: 151-164.

[45] Lin X, Wang Y, Xie Q, et al. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. IEEE Transactions on Services Computing, 2015, 8(2):175-186.

[46] Tang Z, Qi L, Cheng Z, et al. An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. Journal of Grid Computing, 2016, 14(1): 55-74.

[47] Farris I , Taleb T , Flinck H , et al. Providing ultra‐short latency to user‐centric 5G applications at the mobile network edge. Transactions on Emerging Telecommunications Technologies, 2017, 29(4): e3169.

[48]Li Chunlin, Yan Xin, Zhang Yang, Luo Youlong, Multiple Context Based Service Scheduling for Balancing Cost and Benefits of Mobile Users and Cloud Datacenter Supplier in Mobile Cloud, Computer Networks, Elsevier, 2017, July, Volume 122, Pages 138–152

[49] Li Chunlin, Zhu Liye, Luo Youlong, Location-aware interest-related Micro-cloud Topology Construction and Bacteria Foraging-based Offloading Strategy, Ad Hoc Networks, Elsevier, September, Volume 64, 1–21, 2017

[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, Zhou Min, Luo Youlong, Elastic resource provisioning in hybrid mobile cloud for computationally intensive mobile applications, Journal of Supercomputing, Springer-Verlag, 73(9), 3683-3714, Sep. 2017

Biographical notes:

Chunlin Li is a Professor of Computer Science in Wuhan University of Technology. She received the ME in Computer Science from Wuhan Transportation University in 2000, and PhD in Computer Software and Theory from Huazhong University of Science and Technology in 2003. Her research interests include cloud computing and distributed computing.

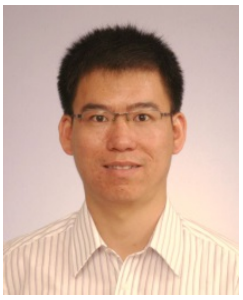
Jianhang Tang received his BS degree in Applied Mathematics from South-central University for Nationalities in 2013 and MS degree in Applied Statistics from Lanzhou University in 2015. He is a PhD student in School of Computer Science and Technology from Wuhan University of Technology. His research interests include cloud computing and big data.

Hengliang Tang is the associate professor, School of Information, Beijing Wuzi University, Beijing, China. He received Ph.D. degree from Beijing University of Technology in 2011. His research interest covers internet of things, logistics informatization.

Youlong Luo is a vice Professor of Management at Wuhan University of Technology. He received his M.S. in Telecommunication and System from Wuhan University of Technology in 2003 and his Ph.D. in Finance from Wuhan University of Technology in 2012. His research interests include cloud computing and electronic commerce.

Chunlin Li

Jianhang Tang



Hengliang Tang

Youlong Luo

Highlights

- An integrated utility-based cache placement strategy in edge computing is proposed by jointly considering data chunk transmission cost, caching value and replacement penalty.
- A weighted bipartite graph model is applied to describe the relationships between tasks and edge servers.
- A heuristic algorithm named cache locality-based task scheduling algorithm is proposed. Our proposed algorithm can obtain maximal weighted matching during each heartbeat.
- We evaluate the performance of our proposed method and previous method via extensive experiments. The results indicate that our proposed method can improve the cache hit ratio and data locality.