Liu Q, Guo Y, Wu J *et al.* Effective query grouping strategy in clouds. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(6): 1231–1249 Nov. 2017. DOI 10.1007/s11390-017-1797-9

Effective Query Grouping Strategy in Clouds

Qin Liu[1,2], *Member*, *CCF*, Yuhong Guo[3], Jie Wu[4], *Fellow*, *IEEE*
and Guojun Wang[5,*], *Distinguished Member*, *CCF*, *Member*, *ACM*, *IEEE*

[1]*College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China*
[2]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications Beijing 100876, China*
[3]*School of Computer Science, Carleton University, Ottawa, ON K155B6, Canada*
[4]*Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, U.S.A.*
[5]*School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China*

E-mail: gracelq628@hnu.edu.cn; yuhong.guo@careton.ca; jiewu@temple.edu; csgjwang@gmail.com

Received May 18, 2016; revised January 12, 2017.

Abstract As the demand for the development of cloud computing grows, more and more organizations have outsourced their data and query services to the cloud for cost-saving and flexibility. Suppose an organization that has a great number of users querying the cloud-deployed multiple proxy servers to achieve cost efficiency and load balancing. Given n queries, each of which is expressed as several keywords, and k proxy servers, the problem to be solved is how to classify n queries into k groups, in order to minimize the difference between each group and the number of distinct keywords in all groups. Since this problem is NP-hard, it is solved in mathematic and heuristic ways. Mathematic grouping uses a local optimization method, and heuristic grouping is based on k-means. Specifically, two extensions are provided: the first one focuses on robustness, i.e., each user obtains search results even if some proxy servers fail; the second one focuses on benefit, i.e., each user can retrieve as many files as possible that may be of interest without increasing the sum. Extensive evaluations have been conducted on both a synthetic dataset and real query traces to verify the effectiveness of our strategies.

Keywords cloud computing, cost efficiency, load balancing, robustness, benefit

1 Introduction

Cloud computing has emerged as a new type of commercial paradigm due to its merits of fast deployment, scalability, and elasticity[1]. Organizations with limited budgets can achieve cost-saving and flexibility by outsourcing their data and query services to the cloud. In a typical cloud computing environment, users will query the cloud with certain keywords to retrieve the data of interest, and the cloud will evaluate the query on the whole dataset and return search results to appropriate users[2-3].

During the querying process, the cloud, which is outside the organization's trust domain, will know what kind of data a user is interested in by observing the query and the type of returned files. To preserve user privacy in the cloud, our previous work[4-5] proposed deploying a proxy server between the users and the cloud. In this case, the users will first send their queries to a proxy server, which will aggregate queries and query the cloud on the users' behalf with a combined query, i.e., the union of distinct keywords in the received queries. Therefore, every user query is blended in a crowd. In the combined query, the cloud cannot know which keywords an individual user is searching for or how many users are interested in a specific keyword.

Regular Paper

This research was supported in part by the National Science Foundation of USA under Grant Nos. CNS-1449860, CNS-1461932, CNS-460971, CNS-1439672, CNS-1301774, and ECCS-1231461, the National Natural Science Foundation of China under Grant Nos. 61632009, 61472451, 61402161, 61472131, 61272151, and 61272546, the Hunan Provincial Natural Science Foundation of China under Grant No. 2015JJ3046, and the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) under Grant No. SKLNST-2016-2-20.

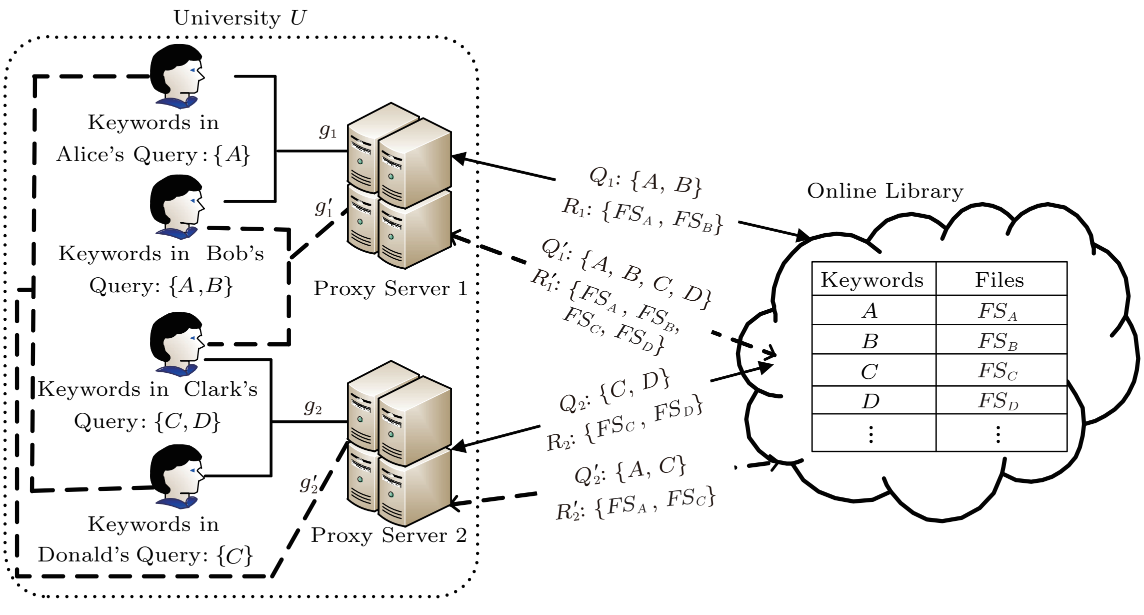
*Corresponding Author

The main drawback of [4-5] is the lack of scalability. A single proxy server with limited computation power and communication bandwidth will quickly become a performance bottleneck when it needs to process thousands of queries simultaneously. Furthermore, it will cause a single point of failure. If the single proxy server fails, all users will lose their search results. Therefore, a preferable solution is to deploy multiple proxy servers to mitigate the above problem. In this case, user queries will be classified in several groups, and a group of queries will be sent to one proxy server, which can help to preserve user privacy by hiding individual queries in a combined query. When each group is of size larger than k, k-anonymity[6-8] is in some sense achieved.

Apart from user privacy, an important problem is how to classify queries in different groups to reduce the cost incurred at the cloud. A naive grouping strategy would be getting each user to send their queries to a random proxy server. The main drawback of this simple solution is the waste of unnecessary bandwidth. To illustrate, let us consider the application as shown in Fig.1. University U outsources the online library resources to a cloud for easy access by its staff and students. The staff and students, as the authorized users, may either subscribe to the online edition of magazines and newspapers, or perform real-time information retrieval by querying the cloud with several keywords to acquire the desired data. Given two proxy servers deployed inside University U, there are two kinds of grouping strategies: 1) Alice and Bob are in group g_1 and Clark and Donald are in group g_2; 2) Alice and Donald are in group g'_1 and Bob and Clark are in group g'_2. Let Q_i and Q'_i denote the combined queries sent from g_i and g'_i, and let R_i and R'_i denote the responses returned to g_i and g'_i, respectively. We observe that when queries with no common keywords are randomly grouped together, as shown in grouping strategy 1, up to 50% of the bandwidth is wasted compared with grouping strategy 2, where queries with the most common keywords are grouped together.

This paper focuses on designing effective query grouping strategies in clouds (EQGC) to simultaneously achieve cost efficiency and load balancing. Cost efficiency refers to minimizing the bandwidth consumed in the cloud. Commercial clouds follow a pay-as-you-go model, where the customer is billed for the consumed bandwidth, CPU time, etc. For example, Amazon EC2 charges \$0.1 for running one large instance per hour, and Amazon S3 charges \$0.25/GB for transferring data from the cloud to the Internet. In other words, reducing the executing time and bandwidth incurred at the cloud is directly translated to monetary savings. Our solution is to group queries with the most common keywords together.

Load balancing refers to balancing bandwidth consumed among proxy servers. For each proxy server, the transfer-in bandwidth is mainly incurred by receiv-



Keywords	Files
A	FS_A
B	FS_B
C	FS_C
D	FS_D
⋮	⋮

Fig.1. Online library in the cloud. File sets FS_A, FS_B, FS_C, and FS_D are described with keywords A, B, C, and D, respectively. Users Alice, Bob, Clark, and Donald query with keywords $\{A\}$, $\{A, B\}$, $\{C, D\}$, and $\{C\}$, respectively, to retrieve desired information.

ing search results from the cloud, and the transfer-out bandwidth is mainly incurred by distributing search results to a group of users. Our solution is to make the number of queries in each group (group size) equal for load balancing. Meanwhile, given a combined query, the cloud cannot guess which keywords a user is interested in with a confidence higher than $1/k$ if each group size is larger than k. Therefore, k-anonymity is achieved.

Given n queries and k proxy servers, our objective is to classify n queries into k groups of equal size so that the sum number of distinct keywords in all groups is minimized. The grouping problem is similar to the clustering and graph cutting problems, which have been proven to be NP-hard. This paper solves it in mathematic and heuristic ways. Mathematic grouping solves the relaxed problem by using a local optimization method. Heuristic grouping is based on the classical heuristic clustering algorithm, k-means[9]. A basic grouping strategy called BasicEQGC which mainly addresses cost efficiency and load balancing is provided first. Then, two extensions are provided. The first extension, RobustEQGC, addresses robustness, where each user can obtain search results even if some machines fail. Our solution is to generate multiple copies for each query, where each copy will be classified in different groups and sent to different proxy servers. As long as one proxy server runs, the user will not lose search results. The second extension, BenefitEQGC, addresses benefit, where each user can retrieve as many files as possible that may be of interest without increasing the bandwidth consumed in the cloud. Our solution is to allow each user to choose several unsure keywords together with several sure keywords, where only sure keywords will take effect. The strategies are constructed in both mathematic and heuristic ways[①]. The key contributions of this paper are as follows.

1) To the best of our knowledge, it is the first attempt to devise effective query grouping strategies to reduce the bandwidth at the cloud.

2) The grouping problem is solved in both mathematical and heuristic ways to simultaneously achieve cost efficiency and load balancing.

3) The robustness and benefit extensions are provided so as to provide more convenient and more personalized cloud services.

4) Extensive experiments have been performed on both a synthetic dataset and real query traces to validate our grouping strategies.

Paper Organization. The preliminaries are provided in Section 2 before the overview in Section 3. The basic grouping strategy is presented in Section 4. In Section 5, the robust extension is presented. Next, the benefit extension is presented in Section 6. After providing an evaluation in Section 7, this paper introduces the related work in Section 8. Finally, this paper is concluded in Section 9.

2 Preliminaries

In this section, we will first provide our system model. Then, we will describe the design goals of this work and analyze which parameters affect the design goals.

2.1 System Model

The system consists of three kinds of entities: users, proxy servers, and the cloud, as shown in Fig.2. The most relevant notations are shown in Table 1. The proxy servers can be classified into r query routers (QRs) and k aggregation and distribution machines (ADMs). The QR will run our grouping strategies to classify a batch of n queries into k groups. The ADM, as the only entity interacting with the cloud, acts like a query aggregator and a result distributor. The main function of the cloud is to process the combined queries on the file collection and return search results to the ADMs.

The interaction process between above entities is as follows: 1) the user sends individual queries (MSG_1) to the QR; 2) the QR responds to each user with the grouping decision (MSG_2) and forwards a group of queries (MSG_3) to the ADM; 3) the ADM sends the cloud a combined query (MSG_4); 4) the cloud returns overall outcomes (MSG_5) to the ADM; 5) the ADM distributes results (MSG_6) to each user in a group.

Note that if a user cannot receive responses from the QR in a period of time, he/she will send the query (MSG'_1) to an ADM. There are k ADMs, each taking charge of a range of IP addresses. To decide which ADM to send the query to, the user first determines which range the IP address falls into and sends the query to the corresponding ADM. Once receiving

[①]M and H are used to denote mathematic and heuristic constructions respectively. Therefore, there are M-BasicEQGC and H-BasicEQGC for BasicEQGC, M-RobustEQGC and H-RobustEQGC for RobustEQGC, and M-BenefitEQGC and H-BenefitEQGC for BenefitEQGC.

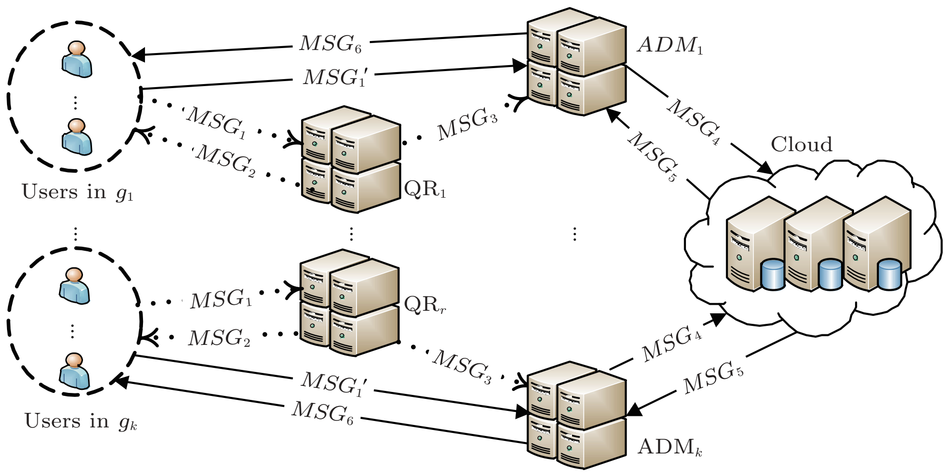


Fig.2. System model.

queries from the users directly, the ADM surmises that the current QR fails and proposes to elect a new QR.

Table 1. Summary of Notations

Notation	Description
n	Number of users/queries in a batch
d	Number of keywords in the dictionary
t	Number of files stored in the cloud
k	Number of groups
g_j	Group j
s_j	Seed of g_j
Q_i	User i's query
$\hat{Q}_j$	Combined query of group j
$\boldsymbol{Q}$	Query matrix
S_i	Number of keywords in Q_i
$\hat{S}_j$	Number of keywords in $\hat{Q}_j$

In the above process, the QR can efficiently output a reasonable grouping decision while there are a lot of users querying the cloud. Otherwise, the QR may wait for a period of time to aggregate sufficient queries. This will incur a certain querying delay. It is important to note that the degree of aggregation can be controlled through a time-out mechanism to meet a given querying delay requirement based on different policies. One policy is to let the organization set a parameter T during system setup to determine the time that the QR will wait before querying the cloud. When T is set to zero, it is degraded to a normal sequential search. An alternative policy is to allow each user to specify the tolerable delay personally. Let T_i denote the tolerable delay for user i. User i will send T_i together with the query to the QR, which will record T_i in an alarming table. If the time in the alarming table is overdue, the QR will run the proposed grouping strategies with all of the queries received so far.

2.2 Design Goals

Our design goals are to design effective grouping strategies in cloud computing by simultaneously achieving:

1) effectiveness: the QR can generate optimal grouping results within a polynomial time;

2) cost efficiency: the bandwidth consumed in the cloud is minimized;

3) load balancing: the bandwidth consumed among the ADMs is balanced;

4) robustness: users can obtain search results even if some ADMs fail;

5) benefits: users can retrieve as many files as possible that may be of interest without increasing the bandwidth consumed in the cloud.

The basic grouping strategy focuses on simultaneously achieving requirements 1∼3. The robust extension focuses on simultaneously achieving requirements 1∼4. The benefit extension focuses on simultaneously achieving requirements 1∼3, 5.

2.3 Parameter Analysis

Suppose that the number of universal keywords is d. To analyze which factors may impact our design goals, a simplest case is used as follows: each file is described by γ keywords and the probability of a keyword in a file is γ/d. Furthermore, the difference of the number of keywords in each user's query is assumed to be small. Cost efficiency refers to minimizing the number of files returned from the cloud. Thus, the expected value of the number of returned files can be calculated with (1):

$$e_1 = \sum\nolimits_{j=1}^{k} t \times (1 - (1 - \gamma/d)^{\hat{S}_j}), \qquad (1)$$

where t is the number of files stored in the cloud, k is the number of groups, and $\hat{S}_j$ is the number of keywords in the combined query $\hat{Q}_j$ generated by ADM_j for $1 \leqslant j \leqslant k$. Given that t, k, γ, and d are fixed, e_1 mainly depends on $\hat{S}_1, \ldots, \hat{S}_k$. Therefore, cost efficiency is equivalent to minimizing the number of keywords in each combined query.

Load balancing refers to balancing bandwidth consumed among ADMs. The transfer-in bandwidth is primarily incurred by receiving files from the cloud. (1) shows that for ADM_j, the expected value of the number of returned files mainly depends on $\hat{S}_j$. Suppose that the i-th user in g_j chooses S_i keywords. The average number of keywords for all of users in g_j is $\bar{S}_j = \sum_{i=1}^{|g_j|} S_i/|g_j|$, where $|g_j|$ is the size of g_j. The expected value of $\hat{S}_j$ can be calculated with (2):

$$e_{2j} = d \times (1 - (1 - \overline{S_j}/d)^{|g_j|}). \qquad (2)$$

The transfer-out bandwidth is primarily incurred by returning files to a group of users. For ADM_j, the estimated value of the number of files returned to g_j can be calculated with (3):

$$e_{3j} = \sum\nolimits_{i}^{|g_j|} t \times (1 - (1 - \gamma/d)^{S_i}). \qquad (3)$$

Given that t, γ, and d are fixed and S_i is basically the same, e_{2j} and e_{3j} mainly depend on $|g_j|$. Therefore, achieving load balancing is equivalent to balancing the number of users in each group.

3 Overview of the EQGC Schemes

In this section, we will overview the basic grouping strategy and its two extensions.

3.1 Basic Grouping Strategy

Given a public dictionary Dic of size d that consists of the universal keywords, our basic idea is to construct a user query or a combined query as a 0-1 bit string of size d. Let Q_i and $\hat{Q}_j$ represent bit strings constructed from user i's query and group j's combined query, respectively. The following definitions are provided.

Definition 1 (User Query in BasicEQGC). *Q_i is a* 0-1 *bit string, where $Q_i[l] = 1$ if the l-th keyword in the dictionary is chosen by user i; otherwise $Q_i[l] = 0$.*

Definition 2 (Combined Query in BasicEQGC). *$\hat{Q}_j$ is a* 0-1 *bit string, where $\hat{Q}_j[l] = 1$ if the l-th keyword in the dictionary is chosen by at least one user in group j; otherwise $\hat{Q}_j[l] = 0$.*

Therefore, minimizing the number of keywords in the combined queries is equal to minimizing the total number of 1s in the combined queries. Let cost denote the total number of 1s in the combined queries. Based on (1)∼(3), the problem to be solved in the basic version can be defined as follows.

Definition 3 (Problem Definition in BasicEQGC). *Given k ADMs and n queries, we derive a grouping strategy such that* 1) *each group is of size n/k and* 2) *the cost is minimized.*

The problem of generating a grouping strategy with the above properties is NP-hard. An intuitive example of the "tourist grouping problem" is as follows: suppose that each of n tourists has a set of interests for sightseeing places, and that for each attraction, its cost association is measured by the number of its appearances in different groups. The question is how to partition n tourists into k groups of equal sizes (to balance the group size for each tour guide) so that each interest of a tourist is covered and at the same time, the total number of places the k groups have to cover is minimized (to reduce the operation cost of the tourist company). In the "tourist grouping problem", BasicEQGC is formulated as a tourist assignment problem with n tourists divided into k groups of equal size m, i.e., $n = m \times k$. Each tourist is associated with a vector of interest for attractions: 1 stands for attractions with interest and 0 for attractions without interest. The optimal tourist assignment is to minimize the total number of attractions tour guides have to visit for all groups to reduce total time spent. When $k = 2$ (i.e., there are two groups), the graph bisection problem, a known NP-hard problem, can be reduced to the tourist assignment problem. As $k = 2$ is a special case, BasicEQGC is NP-hard.

As the queries with more common keywords are grouped together, there are fewer 1s in the combined query. To illustrate, let us assume that the dictionary consists of (A, B, C, D, E, F, G, H) and that the sample queries are as shown in Table 2.

For example, Q_1 denotes that keywords $\{A, B, C\}$ are chosen, and Q_2 denotes that keywords $\{A, B\}$ are chosen. Table 3 provides the sample grouping patterns, where P1 is an instance of random grouping and P2 is an instance of BasicEQGC. Table 3 shows that P1 randomly groups queries with no common keywords together, generating more 1s in the combined queries than P2. For the sample file collection in Table 4, the cloud needs to return 2 000 files in P1 and 1 600 files in P2. Thus, the bandwidth in P2 is saved by about 25% compared with that in P1.

Table 2. Sample User Queries in the Basic Version

Sample Query
$Q_1 = (11100000)$
$Q_2 = (11000000)$
$Q_3 = (11000000)$
$Q_4 = (00010000)$
$Q_5 = (00000111)$
$Q_6 = (00000011)$
$Q_7 = (00000011)$
$Q_8 = (00001000)$

Table 3. Sample Grouping Patterns in the Basic Version

Grouping Pattern	Group Result	Cost
P1	$g_1 = \{Q_1, Q_5\}$, $\hat{Q}_1 = 11100111$	16
	$g_2 = \{Q_2, Q_6\}$, $\hat{Q}_2 = 11000011$	
	$g_3 = \{Q_3, Q_7\}$, $\hat{Q}_3 = 11000011$	
	$g_4 = \{Q_4, Q_8\}$, $\hat{Q}_4 = 00011000$	
P2	$g_1 = \{Q_2, Q_3\}$, $\hat{Q}_1 = 11000000$	12
	$g_2 = \{Q_1, Q_4\}$, $\hat{Q}_2 = 11110000$	
	$g_3 = \{Q_6, Q_7\}$, $\hat{Q}_3 = 00000011$	
	$g_4 = \{Q_5, Q_8\}$, $\hat{Q}_4 = 00001111$	

Table 4. Sample File Collection

File	Keyword
$F_1, \ldots, F_{100}$	$\{A, B\}$
$F_{101}, \ldots, F_{200}$	$\{B, C\}$
$F_{201}, \ldots, F_{300}$	$\{C, D\}$
$F_{301}, \ldots, F_{400}$	$\{D, E\}$
$F_{401}, \ldots, F_{500}$	$\{E, F\}$
$F_{501}, \ldots, F_{600}$	$\{F, G\}$
$F_{601}, \ldots, F_{700}$	$\{G, H\}$
$F_{701}, \ldots, F_{800}$	$\{H, A\}$

3.2 Robust Version

In the basic version, a group of users send their queries to a single ADM. In this case, if the ADM fails, at least one group of users cannot obtain search results. A breakdown of machinery is common for a large-scale system. To ensure that each user obtains search results even if some ADMs fail, the robust version generates multiple copies for each query with the constraint that each copy will be classified into different groups. The queries used in the robust version are defined in the same way as those in the basic version. The problem to be solved in the robust version can be defined as follows.

Definition 4 (Problem Definition in RobustEQGC). *Given k ADMs and n queries, each with $\alpha \leqslant k$ copies, we derive a grouping strategy such that* 1) *each group is of size* $(\alpha \times n)/k$, 2) *the cost is minimized, and* 3) *α copies of each query are classified into different groups.*

BasicEQGC is a special case of RobustEQGC. Therefore, RobustEQGC is NP-hard. The robust version still groups queries with the most common keywords together and consequently, and the performance is better in the robust version than in random grouping. To illustrate, let us assume that the dictionary, the file collection, and the sample queries are the same as those in the basic version. Table 5 provides the sample grouping patterns, where each query has two copies. P3 is an instance of the robust version of random grouping, which requires the cloud to return 2 800 files, and P4 is an instance of RobustEQGC, which requires the cloud to return 2 000 files. Thus, the bandwidth in P4 is saved by about 28.5% compared with that in P3.

Table 5. Sample Grouping Patterns in the Robust Version

Group Pattern	Group Result	Cost
P3	$g_1 = \{Q_1, Q_5, Q_2, Q_6\}$, $\hat{Q}_1 = 11100111$	24
	$g_2 = \{Q_1, Q_5, Q_2, Q_6\}$, $\hat{Q}_2 = 11100111$	
	$g_3 = \{Q_3, Q_7, Q_4, Q_8\}$, $\hat{Q}_3 = 11011011$	
	$g_4 = \{Q_3, Q_7, Q_4, Q_8\}$, $\hat{Q}_4 = 11011011$	
P4	$g_1 = \{Q_1, Q_4, Q_2, Q_3\}$, $\hat{Q}_1 = 11110000$	16
	$g_2 = \{Q_1, Q_4, Q_2, Q_3\}$, $\hat{Q}_2 = 11110000$	
	$g_3 = \{Q_5, Q_8, Q_6, Q_7\}$, $\hat{Q}_3 = 00001111$	
	$g_4 = \{Q_5, Q_8, Q_6, Q_7\}$, $\hat{Q}_4 = 00001111$	

3.3 Benefit Version

In the previous versions, a user only has two options for a keyword: chosen and unchosen. A user is sure that the chosen keywords are useful and the unchosen keywords are useless. The chosen keywords and the unchosen keywords are called sure keywords. Now, let us consider an application in which a user is not sure whether a keyword is useful or not. These kinds of keywords are called unsure keywords. For example, Alice intends to retrieve files that contain the keyword "security". While performing a search, she finds that files that contain the keyword "privacy" may also be of interest. To verify whether the keyword "privacy" is useful or not, she should retrieve related files to read. Suppose that each user is authorized to download only a certain amount of documents from the cloud. Alice faces the risk of wasting the bandwidth on the cloud if the keyword "privacy" is actually useless. In this situation, "privacy" is an unsure keyword to Alice. Although Alice also wishes to retrieve relevant files that contain unsure keywords, she is more concerned about the bandwidth consumed on the cloud. That is to say, Alice wishes to retrieve as many files as possible without increasing the bandwidth consumed on the cloud.

Our solution allows each user to choose several unsure keywords in addition to several sure keywords, where only the sure keywords take effect. Let $Dic[l]$ denote the l-th keyword in the dictionary. The queries used in the benefit version can be defined as follows.

Definition 5 (User Query in BenefitEQGC). *Q_i is a 0-1-$*$ string, where $Q_i[l]=1$ if $Dic[l]$ is a sure chosen keyword to user i; $Q_i[l]=0$ if $Dic[l]$ is a sure unchosen keyword to user i; $Q_i[l]=*$ if $Dic[l]$ is an unsure keyword to user i.*

Definition 6 (Combined Query in BenefitEQGC). *$\hat{Q}_j$ is a 0-1 bit string, where $\hat{Q}_j[l]=1$ if $Dic[l]$ is a sure chosen keyword for at least one user in group j; otherwise $\hat{Q}_j[l]=0$.*

Let benefit denote the number of "$*$" in user queries being turned to 1 in the combined queries. The higher the benefit is, the more files that may be of interest to the users can be returned. The problem to be solved in the benefit version can be defined as follows.

Definition 7 (Problem Definition of BenefitEQGC). *Given n queries and k ADMs, we derive a grouping strategy such that* 1) *each group is of size n/k,* 2) *the cost is minimized, and* 3) *the benefit is maximized under requirements* 1) *and* 2).

BasicEQGC is a special case of BenefitEQGC. Thus, BenefitEQGC is NP-hard. The grouping objective is classified into two layers: the first-layer objective is to minimize the grouping cost; the second-layer objective is to maximize the benefit. To illustrate our layered objective, we consider the following example. Assume that the dictionary consists of (A, B, C, D, E, F, G, H) and that the sample queries are as shown in Table 6. For example, Q_1 means that A, B, and C are sure chosen keywords, and D and E are unsure keywords. Table 7 provides the sample grouping patterns in the benefit version, where P5 and P6 are instances of random grouping and P7 is an instance of BenefitEQGC. Note that P6 and P7, which will incur fewer costs, are better than P5, and P7 generates more benefits and is better than P6.

Table 6. Sample User Queries in the Benefit Version

Sample Query
$Q_1=(111**000)$
$Q_2=(110000**)$
$Q_3=(110000**)$
$Q_4=(00010**0)$
$Q_5=(**000111)$
$Q_6=(00**0011)$
$Q_7=(**000011)$
$Q_8=(0**01000)$

Table 7. Sample Grouping Patterns in the Benefit Version

Grouping Pattern	Group Result	Cost	Benefit
P5	$g_1=\{Q_1,Q_5\}$, $\hat{Q}_1=11100111$	16	6
	$g_2=\{Q_2,Q_6\}$, $\hat{Q}_2=11000011$		
	$g_3=\{Q_3,Q_7\}$, $\hat{Q}_3=11000011$		
	$g_4=\{Q_4,Q_8\}$, $\hat{Q}_4=00011000$		
P6	$g_1=\{Q_2,Q_3\}$, $\hat{Q}_1=11000000$	12	0
	$g_2=\{Q_1,Q_4\}$, $\hat{Q}_2=11110000$		
	$g_3=\{Q_6,Q_7\}$, $\hat{Q}_3=00000011$		
	$g_4=\{Q_5,Q_8\}$, $\hat{Q}_4=00001111$		
P7	$g_1=\{Q_2,Q_3\}$, $\hat{Q}_1=11000000$	12	5
	$g_2=\{Q_1,Q_8\}$, $\hat{Q}_2=11110000$		
	$g_3=\{Q_6,Q_7\}$, $\hat{Q}_3=00000011$		
	$g_4=\{Q_5,Q_4\}$, $\hat{Q}_4=00001111$		

4 BasicEQGC

In this section, the basic grouping strategies will be constructed in both mathematic and heuristic ways.

4.1 M-BasicEQGC

Let $\boldsymbol{Q}$ be an $n\times d$ matrix, representing n query strings with length d. M-BasicEQGC considers classifying these query strings into k groups such that each group has the same number of query strings and the total number of 1s in the sum query string from each group can be minimized. Let $\boldsymbol{Y}\in\{0,1\}^{n\times k}$ denote the grouping setting, then the problem can be formulated into the following optimization:

$$\min_{\boldsymbol{Y}} \quad tr(\boldsymbol{E}^{\mathrm{T}}\delta(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q}))$$
$$\boldsymbol{Y}\in\{0,1\}^{n\times k},\quad \boldsymbol{Y}^{\mathrm{T}}\mathbf{1}=\frac{n}{k}\mathbf{1},\quad \boldsymbol{Y}\mathbf{1}=\mathbf{1}, \quad (4)$$

where tr denotes the trace operator that takes the sum of the diagonal values of the input square matrix, $\mathbf{1}$ denotes a vector of all 1s (assuming that its length can be determined from the context), $\boldsymbol{E}$ denotes a $k\times d$

matrix of all 1s, and $\delta(\cdot)$ denotes an indicator function such that $\delta(c) = 1$ if c is not zero and $\delta(c) = 0$ otherwise. Hence $\delta(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})$ will transform the nonzero values of $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q}$ into 1s and indicate whether there are 1s for each bit of the sum query string from each group. The problem formulated above, however, is NP-hard, and it is difficult to conduct optimization directly over it. M-BasicEQGC thus solves a relaxation problem instead. First, it approximates the indicator matrix, $\delta(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})$, with a smooth function, $1 - \exp(-\beta\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})$, where β is a large constant number, e.g., $\beta = 10, 20, 30, 40$. The differentiable smooth function $1 - \exp(-\beta\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})$ is used to approximate the indicator matrix $\delta(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})$ and to facilitate the optimization procedure. Let $\boldsymbol{C} = \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q}$. Then each (i, j)-th entry of C takes either a zero value such that $C_{ij} = 0$ or a nonzero value such that $1 \leqslant C_{ij} \leqslant n/k$ (based on (4)). For $C_{ij} = 0$, it is observed that $1 - \exp(-\beta C_{ij}) = 0 = \delta(C_{ij})$. For $1 \leqslant C_{ij} \leqslant n/k$ where $\delta(C_{ij}) = 1$, we have $1-\exp(-\beta) \leqslant 1 - \exp(-\beta C_{ij}) \leqslant 1$. With a large constant β value, $1 - \exp(-\beta)$ will be very close to 1. For example, if $\beta = 10$, we have $1 - \exp(-\beta) = 1 - 4.54 \times 10^{-5} \approx 1$. Hence $1 - \exp(-\beta\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})$ can be a good approximation for $\delta(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})$ with a reasonably large β value. Then, it relaxes the integer matrix $\boldsymbol{Y}$ into a continuous matrix. After relaxation, the following optimization problem is obtained:

$$\min_{\boldsymbol{Y}} \; tr(\boldsymbol{E}^{\mathrm{T}}(1 - \exp(-\beta\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q})))$$
$$0 \leqslant \boldsymbol{Y} \leqslant 1, \quad \boldsymbol{Y}^{\mathrm{T}}\mathbf{1} = \frac{n}{k}\mathbf{1}, \quad \boldsymbol{Y}\mathbf{1} = \mathbf{1},$$

which is equivalent to:

$$\max_{\boldsymbol{Y}} \; tr(\boldsymbol{E}^{\mathrm{T}}\exp(-\beta\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Q}))$$
$$0 \leqslant \boldsymbol{Y} \leqslant 1, \quad \boldsymbol{Y}^{\mathrm{T}}\mathbf{1} = \frac{n}{k}\mathbf{1}, \quad \boldsymbol{Y}\mathbf{1} = \mathbf{1}. \quad (5)$$

Let $f(\boldsymbol{Y})$ denote the objective function in (5). Then $f(\boldsymbol{Y})$ is a convex function of $\boldsymbol{Y}$. However, the maximization over a convex function is a non-convex optimization problem. Note that $f(\boldsymbol{Y})$ is a composition of an exponential function with linear functions. Linear functions are affine functions and the exponential function is convex. Hence $f(\boldsymbol{Y})$ is still convex. A first-order local optimization method is used to conduct optimization, which is similar to the existing work [10]. The gradient can be computed as $\nabla_{\boldsymbol{Y}} f(\boldsymbol{Y}) = -\beta\boldsymbol{Q}\exp(-\beta\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Y})$. The first-order local optimization algorithm iteratively updates the variable matrix $\boldsymbol{Y}$ to maximize $f(\boldsymbol{Y})$. In each iteration, it approximates the objective using a first-order Taylor expansion and solves this intermediate convex optimization problem to obtain an updating direction for $\boldsymbol{Y}$. The update of $\boldsymbol{Y}$ is then conducted to maximize $f(\boldsymbol{Y})$ with a simple line search procedure.

After obtaining a continuous solution to (5), $\boldsymbol{Y}^*$, M-BasicEQGC can then round it back to a feasible integer matrix using a heuristic greedy procedure: in each iteration, it finds the $\boldsymbol{Y}$ entry, Y_{ij}, with the largest value among all of the entries in consideration. If there are currently less than n/k 1s in the j-th column, it sets $Y_{ij} = 1$, and sets all the other entries on the same row to 0. Then it removes this row from further consideration. If there are already n/k 1s in the j-th column, it sets $Y_{ij} = 0$ and gets to the next iteration. When the maximum Y_{ij} returned is 0, it completes the procedure. The solution obtained after using the proposed optimization method is a local optimal solution. To overcome the drawback of local optima, it uses random restarts to produce multiple local optimal solutions and picks the best one.

4.2 H-BasicEQGC

The heuristic grouping strategy is based on the classical heuristic clustering algorithm k-means. In statistics and data mining, k-means clustering aims to partition n members into k clusters so that each member belongs to the cluster with the nearest mean. Although this problem is computationally difficult (NP-hard), there are efficient heuristic algorithms that converge quickly to a local optimum[11]. Before describing H-BasicEQGC, the related definitions are provided as the background knowledge.

Definition 8 (Group Seed). *For $1 \leqslant j \leqslant k$, the group seed, denoted as s_j, is the center and the first member of group j.*

Definition 9 (Distance). *The distance between query Q_i and query Q_j, denoted as $Dist(Q_i, Q_j)$, is the number of increased 1s for Q_i after combining with Q_j.*

For example, if $Q_1 = (11100000)$ and $Q_6 = (00000011)$, the combined query is $\hat{Q} = 11100011$. The number of increased 1s is 2 for Q_1 and 3 for Q_6. Therefore, $Dist(Q_1, Q_6) = 2$ and $Dist(Q_6, Q_1) = 3$.

Definition 10 (Nearest Neighbor). *Given a query Q_j, query Q_i with the minimal distance from Q_j is Q_j's nearest neighbor.*

Take the queries in Table 2 as an example: Q_1's nearest neighbors include Q_2, Q_3 with the minimal distance 0, and Q_2's nearest neighbor is Q_3 with the minimal distance 0.

As shown in Algorithm 1, H-BasicEQGC largely consists of two steps and will be run within multiple rounds. The first step is to find k group seeds. In the first round, since no group has been formed yet, k seeds are randomly chosen from n queries; in the following rounds, given k groups, each seed is chosen randomly from n/k queries in each group. The second step is to classify $n-k$ queries that are closest to a seed into a group. Specifically, a set $CandiQ$ that consists of universal queries with the exception of k seeds is first constructed. Then, given seed s_j, a set $Neighbor_j$ that accommodates the nearest neighbors of s_j is constructed and a random element in $Neighbor_j$ is removed from $CandiQ$ after being classified into g_j. The groups are processed in ascending order of their IDs, where the group with a lower ID has a higher priority for choosing the next member. Then, the IDs of the groups are reordered at the end of each round. In each round, the grouping result will be recorded. At the end of this algorithm, the optimal result will be the output.

Algorithm 1. H-BasicEQGC

```
1: Initialize $CandiQ$ with the universal user queries
2: Randomly choose $k$ distinct queries from $CandiQ$ as
   seeds $s_1, \dots, s_k$ for groups $g_1, \dots, g_k$, and remove them
   from $CandiQ$
3: while $CandiQ$ is not empty do
4:    for $j = 1$ to $k$ do
5:       $Neighbor_j$ is a subset of $CandiQ$ that accommo-
         dates $s_j$'s nearest neighbors
6:       Choose a random element $Q_i \in Neighbor_j$ into $g_j$
         and remove it from $CandiQ$
7:    end for
8: end while
9: Reorder the IDs of the groups
10: Initialize $CandiQ$ with the universal user queries
11: for $j = 1$ to $k$ do
12:    Randomly choose a query from $g_j$ as the seed $s_j$ for
       the next round and remove it from $CandiQ$
13: end for
```

Table 8 shows how H-BasicEQGC classifies the sample queries, as shown in Table 2, into four groups. In the first step, Q_2, Q_1, Q_6, and Q_5 are chosen as the seeds of g_1, g_2, g_3, g_4, respectively. In the second step, the nearest neighbor set is calculated for each group seed, where a random element will be chosen as the next group member. For example, after choosing group seeds, $CandiQ$ includes $\{Q_3, Q_4, Q_7, Q_8\}$. For s_1, the nearest neighbor set $Neighor_1 = \{Q_3\}$ with the minimal distance equaling 0 ($Dist(s_1, Q_3) = 0$). Thus, Q_3 is chosen as the next member of g_1, and will be removed from $CandiQ$. Now, $CandiQ$ includes $\{Q_4, Q_7, Q_8\}$. For s_2, the nearest neighbor set $Neighor_2 = \{Q_4, Q_8\}$ with minimal distance equaling 1 ($Dist(s_2, Q_4) = Dist(s_2, Q_8) = 1$). A random element is chosen from $Neighor_2$, e.g., Q_4 as the next member of g_2, and will be removed from $CandiQ$. The members of g_3 and g_4 are chosen in the same way.

Table 8. Working Process of H-RobustEQGC

Grouping Seed	Step 1	Step 2
$s_1 = \{Q_2\}$	$g_1 = \{Q_2\}$	$CandiQ = \{Q_3, Q_4, Q_7, Q_8\}$ $Neighor_1 = \{Q_3\}$ $g_1 = \{Q_2, Q_3\}$
$s_2 = \{Q_1\}$	$g_2 = \{Q_1\}$	$CandiQ = \{Q_4, Q_7, Q_8\}$ $Neighor_2 = \{Q_4, Q_8\}$ $g_2 = \{Q_1, Q_4\}$
$s_3 = \{Q_6\}$	$g_3 = \{Q_6\}$	$CandiQ = \{Q_7, Q_8\}$ $Neighor_3 = \{Q_7\}$ $g_3 = \{Q_6, Q_7\}$
$s_4 = \{Q_5\}$	$g_4 = \{Q_5\}$	$CandiQ = \{Q_8\}$ $Neighor_4 = \{Q_8\}$ $g_4 = \{Q_5, Q_8\}$

5 Robust Extension

In the basic version, a group of users send their queries to only one ADM and will lose search results when this ADM fails. To ensure that a user has a higher success rate of retrieving files back compared with BasicEQGC, we consider a robust extension, which allows a user to send multiple query copies to different ADMs in this section. As in the basic version, we construct the robust version in both mathematic and heuristic ways.

5.1 M-RobustEQGC

M-RobustEQG only provides the construction for two query copies. However, our grouping strategy can be easily adapted to more than two query copies. Let $\boldsymbol{Q}$ be an $n \times d$ matrix, representing n query strings with length d. Given two copies of $\boldsymbol{Q}$, assigning the copies of each query string to different groups is required. Let $\boldsymbol{Z} = (\boldsymbol{Q}; \boldsymbol{Q})$ be a $2n \times d$ matrix, it then has

$$\begin{aligned} \min_{\boldsymbol{Y}} \quad & tr(\boldsymbol{E}^{\mathrm{T}} \delta(\boldsymbol{Y}^{\mathrm{T}} \boldsymbol{Z})) \\ & \boldsymbol{Y} \in \{0,1\}^{2n \times k}, \boldsymbol{Y}^{\mathrm{T}} \boldsymbol{1} = \frac{2n}{k} \boldsymbol{1}, \\ & \boldsymbol{Y1} = \boldsymbol{1}, \boldsymbol{AY} \leqslant 1, \end{aligned} \tag{6}$$

where $\boldsymbol{A} = (\boldsymbol{I}, \boldsymbol{I})$, and $\boldsymbol{I}$ is an $n \times n$ identity matrix. The relaxed optimization problem is

$$\max_{\boldsymbol{Y}} \quad tr(\boldsymbol{E}^{\mathrm{T}} \exp(-\beta \boldsymbol{Y}^{\mathrm{T}} \boldsymbol{Z}))$$

$$\begin{aligned} &0 \leqslant \boldsymbol{Y} \leqslant 1, \quad \boldsymbol{Y}^{\mathrm{T}}\mathbf{1} = \frac{2n}{k}\mathbf{1}, \\ &\boldsymbol{Y}\mathbf{1} = \mathbf{1}, \quad \boldsymbol{A}\boldsymbol{Y} \leqslant 1. \end{aligned} \tag{7}$$

The heuristic greedy rounding procedure can be updated correspondingly: in each iteration, it finds the $\boldsymbol{Y}$ entry, Y_{ij}, with the largest value among all the entries in consideration. If there are currently less than $2n/k$ 1s in the j-th column, it sets $Y_{ij} = 1$, $Y_{\hat{i}j} = 0$ (if $i > n$, then $\hat{i} = i - n$, otherwise $\hat{i} = i + n$), sets all of the other entries on the i-th row to 0 and removes this row from further consideration. If there are already $2n/k$ 1s in the j-th column, it sets $Y_{ij} = 0$ and gets to the next iteration. When the maximum Y_{ij} returned is 0, it completes the procedure.

5.2 H-RobustEQGC

In H-RobustEQGC, each user generates $2 \leqslant \alpha \leqslant k$ copies for each query. Our objective is to classify all query copies in k groups of equal size so that the grouping cost is minimized under the constraint that all query copies are classified in the different groups. H-RobustEQGC continues to use the definitions (group seed, distance, and nearest neighbor) defined in H-BasicEQGC. As shown in Algorithm 2, the main differences between H-RobustEQGC and H-BasicEQGC lie in the constructions of the sets $CandiQ$ and $Neighbor_j$ for $1 \leqslant j \leqslant k$. In H-RobustEQGC, $CandiQ$ consists of α copies of each n queries, and $Neighbor_j$ consists of the nearest neighbors of s_j but not in g_j.

To illustrate, let us consider the example shown in Table 9. The example takes the queries shown in Table 2 as samples and sets $\alpha = 2$. Therefore $CandiQ$ is initialized with $\{Q_1, \ldots, Q_8, Q_1, \ldots, Q_8\}$. In the first step, queries Q_2, Q_1, Q_6, and Q_5 are chosen as the seeds of g_1, g_2, g_3, g_4, respectively. In the second step, H-RobustEQGC classifies the queries that are the closest to the seed of a group g_i ($i \in [1, 2, 3, 4]$) and are not in g_i into g_i. For example, for s_1, the nearest neighbors include Q_3 and Q_2. Based on the constraint that all copies should be classified into different groups, it sets $Neighbor_1 = \{Q_3\}$ instead of $Neighbor_1 = \{Q_2, Q_3\}$.

Algorithm 2. H-RobustEQGC

```
1: Initialize $CandiQ$ with $2 \leqslant \alpha \leqslant k$ query copies
2: Randomly choose $k$ distinct queries from $CandiQ$ as
   seeds $s_1, \ldots, s_k$ for groups $g_1, \ldots, g_k$ respectively, and
   remove them from $CandiQ$
3: while $CandiQ$ is not empty do
4:     for $j = 1$ to $k$ do
5:         $Neighbor_j$ is a subset of $CandiQ - g_j$ that accom-
           modates $s_j$'s nearest neighbors
6:         Choose a random element $Q_i \in Neighbor_j$ into $g_j$
           and remove it from $CandiQ$
7:     end for
8: end while
9: Reorder the IDs of the groups
10: Initialize $CandiQ$ with $2 \leqslant \alpha \leqslant k$ query copies
11: for $j = 1$ to $k$ do
12:     Randomly choose a query from $g_j$ as the seed $s_j$ for
        the next round and remove it from $CandiQ$
13: end for
```

6 Benefit Extension

In the previous versions, we assume that a user is sure of whether a keyword is useful or not. In this section, we will consider an extension where a user is allowed to choose some unsure keywords to retrieve files that may be of interest. As in the previous versions, we provide both mathematic and heuristic constructions.

Table 9. Working Process of H-RobustEQGC

Grouping Seed	Step 1	Step 2	Step 3	Step 4
$s_1 = \{Q_2\}$	$g_1 = \{Q_2\}$	$CandiQ = \{Q_1, Q_2, Q_6, Q_5, Q_3, Q_3, Q_4, Q_4, Q_7, Q_8, Q_7, Q_8\}$ $Neighor_1 = \{Q_3\}$ $g_1 = \{Q_2, Q_3\}$	$CandiQ = \{Q_1, Q_5, Q_3, Q_4, Q_4, Q_8, Q_7, Q_8\}$ $Neighor_1 = \{Q_1, Q_4, Q_8\}$ $g_1 = \{Q_2, Q_3, Q_1\}$	$CandiQ = \{Q_5, Q_4, Q_4, Q_8\}$ $Neighor_1 = \{Q_4, Q_8\}$ $g_1 = \{Q_2, Q_3, Q_1, Q_4\}$
$s_2 = \{Q_1\}$	$g_2 = \{Q_1\}$	$CandiQ = \{Q_1, Q_2, Q_6, Q_5, Q_3, Q_4, Q_4, Q_7, Q_8, Q_7, Q_8\}$ $Neighor_2 = \{Q_2, Q_3\}$ $g_2 = \{Q_1, Q_2\}$	$CandiQ = \{Q_5, Q_3, Q_4, Q_4, Q_8, Q_7, Q_8\}$ $Neighor_2 = \{Q_3\}$ $g_2 = \{Q_1, Q_2, Q_3\}$	$CandiQ = \{Q_5, Q_4, Q_8\}$ $Neighor_2 = \{Q_4, Q_8\}$ $g_2 = \{Q_1, Q_2, Q_3, Q_4\}$
$s_3 = \{Q_6\}$	$g_3 = \{Q_6\}$	$CandiQ = \{Q_1, Q_6, Q_5, Q_3, Q_4, Q_4, Q_7, Q_8, Q_7, Q_8\}$ $Neighor_3 = \{Q_7\}$ $g_3 = \{Q_6, Q_7\}$	$CandiQ = \{Q_5, Q_4, Q_4, Q_8, Q_7, Q_8\}$ $Neighor_3 = \{Q_4, Q_5, Q_8\}$ $g_3 = \{Q_6, Q_7, Q_8\}$	$CandiQ = \{Q_5, Q_8\}$ $Neighor_3 = \{Q_5, Q_8\}$ $g_3 = \{Q_6, Q_7, Q_8, Q_5\}$
$s_4 = \{Q_5\}$	$g_4 = \{Q_5\}$	$CandiQ = \{Q_1, Q_6, Q_5, Q_3, Q_4, Q_4, Q_8, Q_7, Q_8\}$ $Neighor_4 = \{Q_6, Q_7\}$ $g_4 = \{Q_5, Q_6\}$	$CandiQ = \{Q_5, Q_4, Q_4, Q_7, Q_8\}$ $Neighor_4 = \{Q_7\}$ $g_4 = \{Q_5, Q_6, Q_7\}$	$CandiQ = \{Q_8\}$ $Neighor_4 = \{Q_8\}$ $g_4 = \{Q_5, Q_6, Q_7, Q_8\}$

6.1 M-BenefitEQGC

Let $\boldsymbol{Q} \in \{0,1,*\}^{n\times d}$ represent n strings with length d. M-BenefitEQGC aims to classify these strings into k groups such that each group has the same number of strings while in the process of grouping, the "$*$" entries of $\boldsymbol{Q}$ can be either turned into 1 or 0. The goal is to classify n queries into k groups of equal size, so that the total number of nonzeros in the sum strings from each cluster can be minimized while as many "$*$" entries as possible can be turned into 1s. This extended grouping problem can be formulated as a new optimization problem:

$$\begin{aligned}
\max_{\boldsymbol{Z},\boldsymbol{Y}} \quad & tr(\boldsymbol{E}^{\mathrm{T}}\exp(-\beta\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Z}))+\alpha\mathbf{1}^{\mathrm{T}}\boldsymbol{Z}\mathbf{1}\\
\text{s.t.} \quad & \boldsymbol{Y}\in\{0,1\}^{n\times k},\quad \boldsymbol{Y}^{\mathrm{T}}\mathbf{1}=\frac{n}{k}\mathbf{1},\quad \boldsymbol{Y}\mathbf{1}=\mathbf{1}\\
& \boldsymbol{Z}\in\{0,1\}^{n\times d},\quad Z_{ij}=0,\forall(i,j)\in A_0,\\
& Z_{ij}=1,\forall(i,j)\in A_1, \qquad (8)
\end{aligned}$$

where $\mathbf{1}$ denotes a vector of all 1s, assuming its length can be determined from the context. $\boldsymbol{E}$ denotes a $k\times d$ matrix of all 1s. $\boldsymbol{Y}\in\{0,1\}^{n\times k}$ denotes the grouping setting. $\boldsymbol{Z}\in\{0,1\}^{n\times d}$ denotes the resulting 0-1 matrix from $\boldsymbol{Q}$. A_0 and A_1 are defined as $A_0=\{(i,j): Q(i,j)=0\}$ and $A_1=\{(i,j): Q(i,j)=1\}$. The above optimization problem is obviously an integer optimization problem and is non-convex. Let $f(\boldsymbol{Y},\boldsymbol{Z})$ denote the objective function in (8). A relaxed optimization problem is solved as follows:

$$\begin{aligned}
\max_{\boldsymbol{Z}} g(\boldsymbol{Z}) \quad & \text{s.t. } \mathbf{0}\leqslant\boldsymbol{Z}\leqslant 1,\\
& Z_{i,j}=0,\quad \forall(i,j)\in A_0,\\
& Z_{i,j}=1,\forall(i,j)\in A_1,\\
\text{where} \quad & g(\boldsymbol{Z})=\max_{\boldsymbol{Y}} f(\boldsymbol{Y},\boldsymbol{Z})\\
& \text{s.t. } \boldsymbol{Y}\in\{0,1\}^{n\times k}, \boldsymbol{Y}^{\mathrm{T}}\mathbf{1}=\frac{n}{k}\mathbf{1},\\
& \boldsymbol{Y}\mathbf{1}=\mathbf{1}.
\end{aligned}$$

Note the maximization over $\boldsymbol{Y}$ given $\boldsymbol{Z}$ is the same optimization problem formulated in (5) and can be solved correspondingly. The outer maximization over $\boldsymbol{Z}$ can be conducted using a gradient ascend method with line search. Random restarts are used to pick the best solution from all local optimal solutions. Finally, given the yielded solution $\boldsymbol{Y}^*$, $\boldsymbol{Z}$ can be recovered heuristically by maximally turning the "$*$"s in each location of each resulting cluster string into 1s without reducing the objective function value in (8).

6.2 H-BenefitEQGC

The objective of H-BenefitEQGC is classified into two layers: the first-layer objective is to minimize the grouping cost, while the second-layer objective is to maximize the benefit. H-BenefitEQGC continues to use the definitions (group seed, distance, and nearest neighbor) in H-BasicEQGC. Furthermore, the additional definitions are provided as follows.

Definition 11 (Intimacy). *The intimacy between query Q_i and query Q_j, denoted as $Inti(Q_i,Q_j)$, is the sum number of "$*$"s that are turned into* 1 *in the combined query of Q_i and Q_j.*

For example, if $Q_1=(111**000)$ and $Q_5=(**000111)$, then the combined query is 11100111. For Q_1, the number of "$*$" being turned into 1 is 0; for Q_5, the number of "$*$" being turned into 1 is 2. Thus, $Inti(Q_1,Q_5)=0+2=2$.

Definition 12 (Friendliest Neighbor). *Given a query Q_j, query Q_i with the maximal intimacy from Q_j is Q_j's friendliest neighbor.*

Take the queries in Table 6 as an example, Q_1's friendliest neighbor is Q_8 with the maximal intimacy 3, and Q_2's friendliest neighbors include Q_5 and Q_7 with the maximal intimacy 4.

The general idea of H-BenefitEQGC is similar to that of H-BasicEQGC, excepting that a new condition is added for choosing the next member. Specifically, the next member, selected from all of the seed's nearest neighbors, has the maximal intimacy from the seed. As shown in Algorithm 3, given seed s_j, a set

Algorithm 3. H-BenefitEQGC

```
1: Initialize $CandiQ$ with the universal user queries
2: Randomly choose $k$ distinct queries from $CandiQ$ as
   seeds $s_1,\dots,s_k$ for groups $g_1,\dots,g_k$, and remove them
   from $CandiQ$
3: while $CandiQ$ is not empty do
4:   for $j=1$ to $k$ do
5:     $Neighbor_j$ is a subset of $CandiQ$ that accommo-
       dates $s_j$'s nearest neighbors
6:     $Friend_j$ is a subset of $Neighbor_j$ that accommo-
       dates $s_j$'s friendliest neighbors
7:     Choose a random element $Q_i \in Friend_j$ into $g_j$
       and remove it from $CandiQ$
8:   end for
9: end while
10: Reorder the IDs of the groups
11: Initialize $CandiQ$ with the universal user queries
12: for $j=1$ to $k$ do
13:   Randomly choose a query from $g_j$ as the seed $s_j$ for
      the next round and remove it from $CandiQ$
14: end for
```

$Neighbor_j \subseteq CandQ$ that accommodates the nearest neighbors of s_j is first constructed. For each member in $Neighbor_j$, H-BenefitEQGC calculates its intimacy from s_j and then constructs $Friend_j \subseteq Neighbor_j$, which accommodates the friendliest neighbors of s_j. Finally, a random element in $Friend_j$ is removed from $CandiQ$ after being classified into g_j. The above process will be run within multiple rounds, and in each round, the grouping cost and the grouping benefit will be recorded. At the end of this algorithm, the grouping patterns that generate the minimal grouping cost are set as the candidate outputs from which the output with the maximal benefit will be chosen as the optimal result.

Table 10 shows how H-BenefitEQGC classifies the sample queries (shown in Table 6) into four groups. In the first step, Q_2, Q_1, Q_6, and Q_5 are chosen as the seeds of g_1, g_2, g_3, g_4, respectively. In the second step, the friendliest neighbor set is calculated for each group seed where a random element will be chosen as the next group member. For example, for s_2, the nearest neighbor set is $Neighor_2 = \{Q_4, Q_8\}$ where we have $Inti(s_2, Q_4) = 1$ and $Inti(s_2, Q_8) = 3$. Thus, the friendliest neighbor set is $Friend_2 = \{Q_8\}$.

Table 10. Working Process of H-BenefitEQGC

Grouping Seed	Step 1	Step 2
$s_1 = \{Q_2\}$	$g_1 = \{Q_2\}$	$CandiQ = \{Q_3, Q_4, Q_7, Q_8\}$, $Neighor_1 = \{Q_3\}$, $Friend_1 = \{Q_3\}$, $g_1 = \{Q_2, Q_3\}$
$s_2 = \{Q_1\}$	$g_2 = \{Q_1\}$	$CandiQ = \{Q_4, Q_7, Q_8\}$, $Neighor_2 = \{Q_4, Q_8\}$, $Friend_2 = \{Q_8\}$, $g_2 = \{Q_1, Q_8\}$
$s_3 = \{Q_6\}$	$g_3 = \{Q_6\}$	$CandiQ = \{Q_7, Q_4\}$, $Neighor_3 = \{Q_7\}$, $Friend_3 = \{Q_7\}$, $g_3 = \{Q_6, Q_7\}$
$s_4 = \{Q_5\}$	$g_4 = \{Q_5\}$	$CandiQ = \{Q_4\}$, $Neighor_4 = \{Q_4\}$, $Friend_4 = \{Q_4\}$, $g_4 = \{Q_5, Q_4\}$

7 Evaluation

This section will compare the proposed strategies with random strategies in terms of performance, load balancing, and benefit. To verify the effectiveness of our proposed strategies, experiments are conducted on both synthetic and real datasets. The mathematic grouping strategies are implemented with MATLAB R2010a, and the heuristic grouping strategies are implemented with C++, running on a local machine with an Intel Core 2 Duo E8400 3.0 GHz CPU and 8 GB RAM. The whole file set is deployed in TCloud[2]. The m1.xlarge type instances, which have two CPUs, 2 GB RAM, and 20 GB of storage while running on a 64-bit Linux platform are subscribed. The average bandwidth between the instances is 945.5 MB/s.

7.1 Synthetic Data

The parameters for producing synthetic data are summarized in Table 11. The dictionary consists of 1 000 or 2 000 keywords. There are 20∼200 users needed to be classified into 5 or 10 groups. Each user randomly chooses 1∼10 keywords from the dictionary, and the query keywords follow the Power-Law distribution. For ease of illustration, Random, Random-Robust, and Random-Benefit are used to denote the basic version, the robust version, and the benefit version of random grouping, respectively. Given n users, random strategies will be run multiple times and the optimal value will be output as the final result.

Table 11. Parameter Setting in Synthetic Data

Parameter	Description	Value
$\|F\|$	File size	0.1 MB
n	Number of users in a batch	20∼200
d	Dictionary size	$\{1\,000, 2\,000\}$
t	Number of files in the cloud	10 000
r	Number of QRs	1
k	Number of groups (ADMs)	$\{5, 10\}$
S_i	Number of keywords in Q_i	1∼10
α	Number of query copies	2
γ	Number of keywords in each file	1∼5
δ	Probability of generating "*"	0.01

7.1.1 *Performance*

This paper first compares the percentage of the reduced number of 1s from the total number of "1" in all combined queries under different settings of n, k, and d. Fig.3 shows that both the proposed strategies and the random strategies work better as the number of users, n, increases. For example, under the setting of $d = 1\,000$ and $k = 5$, while n increases from 20 to 200, the percentage of reduced 1s increases from 21.7% to 60.5% in M-BasicEQGC, from 26.7% to 56.3% in H-BasicEQGC, and from 21.7% to 52.7% in Random. The

[2]https://sites.google.com/a/temple.edu/tcloud/home, July 2017. This is the Temple Elastic High Performance Computing(HPC) Cloud project site. The hardware is purchased under the National Science Foundation of USA under Grant No. CNS-0958854.

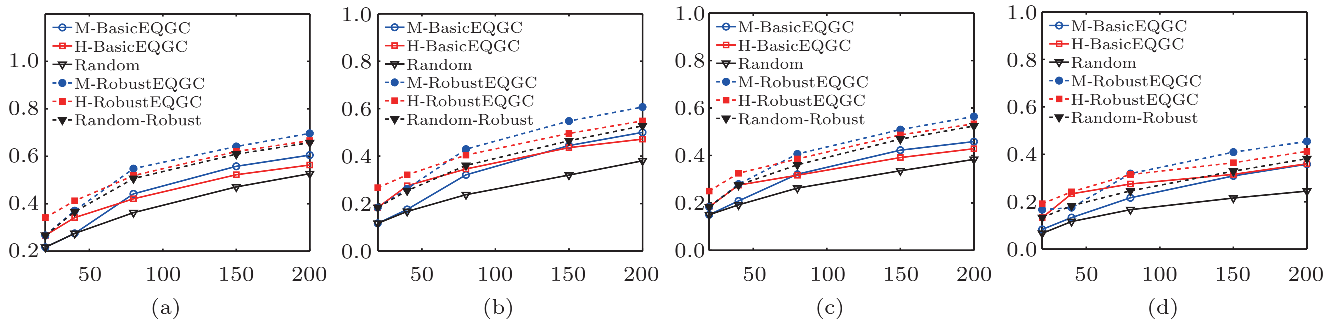


Fig.3. Comparison of the percentage of the reduced 1s in the synthetic dataset. The X-axis denotes the number of users and the Y-axis denotes the corresponding percentage. (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

reason is that in a dictionary of fixed size, the probability of a keyword being chosen by multiple users increases as the number of users in a group increases. This tendency conforms with the analysis results in Subsection 2.3.

Furthermore, all of the strategies work more poorly as either the number of groups k or the size of dictionary d increases. For example, under the setting of $n = 200$ and $d = 2\,000$, the percentage of reduced 1s decreases from 45.8% to 35.8% in M-BasicEQGC, from 42.8% to 36% in H-BasicEQGC, and from 38.3% to 24.5% in Random as k increases from 5 to 10; under the setting of $k = 10$ and $n = 100$, the percentage of reduced 1s decreases from 42.9% to 31.9% in M-RobustEQGC, from 40.4% to 31.5% in H-RobustEQGC, and from 36% to 24.6% in Random-Robust as d increases from $1\,000$ to $2\,000$. The reason is that with a fixed number of users, the probability of a keyword being chosen by multiple users decreases as the number of keywords in the dictionary increases. Similarly, in a group of five users, there is a higher probability that the users will choose the same keywords in a group of 25 in size.

It is also observed that our strategies generate fewer 1s than random strategies in both the basic and the robust versions. When the number of users exceeds a threshold value (e.g., $n = 100$ under the setting of $d = 2\,000$ and $k = 5$), mathematic strategies work better than heuristic strategies. This is because heuristic grouping is apt to obtain the local optimal result when the number of users is sufficiently large. Mathematic strategies can relieve this problem by setting multiple random restarting points.

To compare the bandwidth incurred at the cloud, it is assumed that the cloud maintains $10\,000$ files, where each file of size 0.1 MB is described by 1$\sim$5 keywords. It is also assumed that keywords are uniformly distributed in a file set. Fig.4 shows that our strategies can save more bandwidth than random strategies in both the basic and the robust versions. For example, under the setting of $d = 2\,000$ and $k = 10$, the bandwidth incurred at the cloud is reduced by 1%$\sim$15% in M-BasicEQGC and by 7%$\sim$16% in H-BasicEQGC compared with Random. Furthermore, the bandwidth incurred at the cloud increases either as the number of users or the number of groups increases, or as the dictionary size decreases.

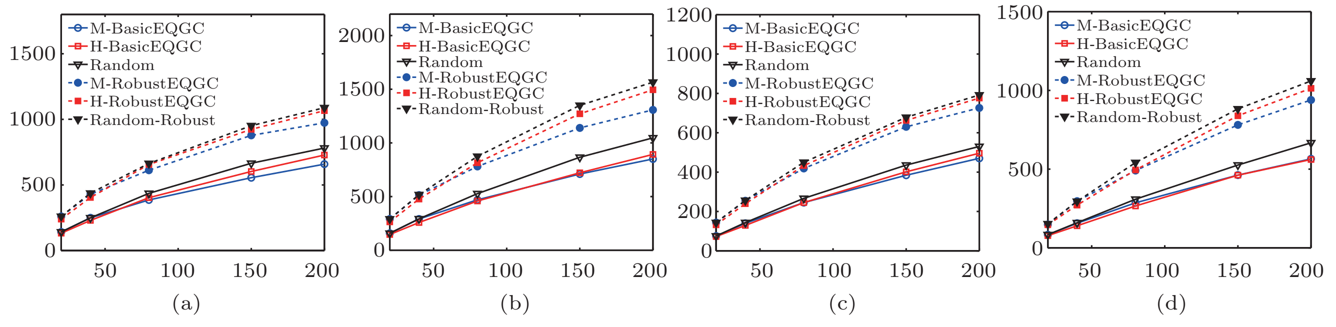


Fig.4. Comparison of the bandwidth at the cloud in the synthetic dataset. The X-axis denotes the number of users and the Y-axis denotes the corresponding bandwidth (MB). (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

7.1.2 *Load Balancing*

The imbalanced bandwidth imb is an aggregate value of the discrepancies of the consumed bandwidth at the ADMs from the average consumed bandwidth. This can be calculated with $imb = \sum_{i=1}^{k} |B_i - ave|$, where B_i is the bandwidth at ADM_i and $ave = (\sum_{i=1}^{k} B_i)/k$ is the average bandwidth among k ADMs. The transfer-in bandwidth at each ADM is mainly affected by the amount of 1s in the combined query, $\hat{S}_j$, and the transfer-out bandwidth at each ADM is mainly affected by the sum of 1s, $\sum S_i$, in a group of queries. As described in (2) and (3), the parameters that affect $\hat{S}_j$ are almost the same for each group, but the parameters that affect $\sum S_i$, e.g., the number of keywords in each query, are different for each user. Therefore, as shown in Fig.5 and Fig.6, the imbalanced transfer-out bandwidth is larger than the imbalanced transfer-in bandwidth. Furthermore, heuristic strategies keep load balancing more effectively than the mathematic ones. The reason is that heuristic strategies group queries with the minimal distance from the group seed together, minimizing the difference among a group of queries. Note that the group size is set to be the same and that no significant differences exist. For example, even in the worst case (M-RobustEQGC under the setting of $d = 1\,000$ and $k = 10$), the maximal imbalanced transfer-in and transfer-out bandwidths are 231.6 MB and 561.2 MB, respectively, and the average values of them are just about 143.124 MB and 264.52 MB, respectively. Since imb is an aggregate value, the average difference among all ADMs is about 20 MB.

7.1.3 *Benefit*

The gained benefit will be compared among different strategies. The probability for generating "$*$" is set as $\delta = 0.01$. Therefore, the number of "$*$" in each query for a dictionary of size $d = 1\,000$ is about 10, and for that of size $2\,000$ is about 20. Fig.7 shows that the heuristic strategies generally sacrifice more "$*$"s to 1 than the mathematic strategies in order to achieve a better performance. Random strategies, although gain more benefits than the proposed strategies in certain cases, always incur more costs (as indicated in Fig.4). Furthermore, it is observed that the gained benefit increases as the number of users, n, increases, as the dictionary size, d, increases, or as the number of groups, k, decreases.

7.2 Real Query Traces

To validate the effectiveness of the proposed strategies, simulations are conducted on real query traces, AOL[12]. This collection consists of 10 154 742 unique (normalized) queries collected from 650 000 users over three months from March 01, 2006, to May 31, 2006.

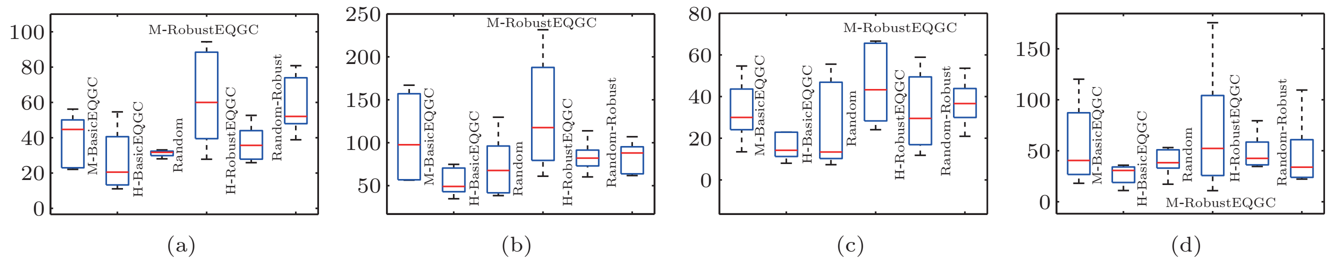


Fig.5. Comparison of the imbalanced transfer-in bandwidth at ADMs in the synthetic dataset. The X-axis denotes the number of users and the Y-axis denotes the corresponding imbalanced bandwidth (MB). (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

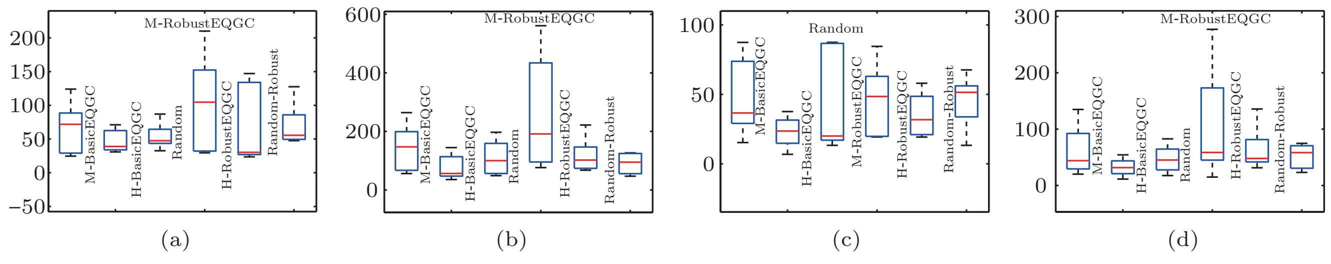


Fig.6. Comparison of the imbalanced transfer-out bandwidth at ADMs in the synthetic dataset. The X-axis denotes the number of users and the Y-axis denotes the corresponding imbalanced bandwidth (MB). (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

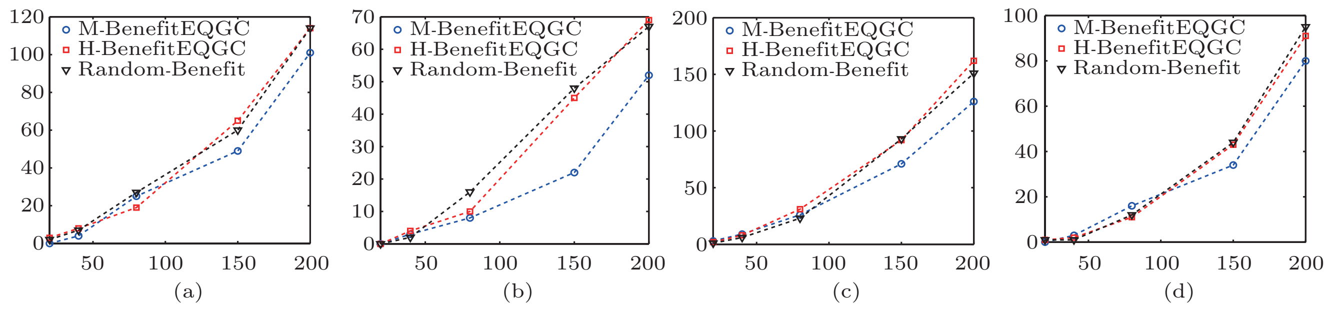


Fig.7. Comparison of the gained benefit in the synthetic dataset. The X-axis denotes the number of users and the Y-axis denotes the corresponding benefit. (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

AOL contains 10 data collections. Our experiments are conducted on the first dataset, which collects 65 535 queries from 400 users. After splitting and extracting keywords from each query, the number of distinct keywords in 65 535 queries is 177 989, each consisting of at most 57 keywords. Obviously, 177 989 is too large to be the dictionary size; in such a setting, few common keywords will exist within a group of queries. Therefore, the dictionary size is set as $d = \{1\,000, 2\,000\}$, where each keyword is mapped to a position in the dictionary by using a hash function. After mapping, each query consists of about 1~10 keywords. Meanwhile, the number of users n is set to $\{100, 200, 300, 400\}$, where a random query is chosen for each user for experiments.

Performance. This paper first compares the percentage of reduced "1" among different strategies. Fig.8 shows that under all parameter settings, mathematic strategies work best in both basic and robust versions. For example, under the setting of $d = 2\,000$ and $k = 10$, the percentage of the reduced number of 1s achieves 19.5% in M-BasicEQGC, 15.8% in H-BasicEQGC, 11% in Random, 24.3% in M-RobustEQGC, 19.1% in H-RobustEQGC, and 17.1% in Random-Robust. To compare the bandwidth incurred at the cloud, parameters t and γ as well as the keyword distribution in file sets are set the same as those in the synthetic dataset. Fig.9 shows the comparison results. It is observed that mathematic strategies produce the least bandwidth in both basic and robust versions. Furthermore, the variation trends of the results shown in Fig.8 and Fig.9 are consistent with those in Fig.3 and Fig.4.

Load Balancing. The comparisons of imbalanced transfer-in bandwidth and transfer-out bandwidth are shown in Fig.10 and Fig.11, respectively. It is observed that in all settings, heuristic strategies maintain better load balancing than mathematic strategies in both basic and robust versions. Furthermore, the transfer-out imbalance bandwidth is larger than the transfer-in imbalanced bandwidth. This observation is consistent with the synthetic data. And even in the worst case (while under the parameter setting $d = 1\,000$ and $k = 10$), the aggregated imbalanced transfer-in bandwidth is about 237.26 MB in M-BasicEQGC and 265 MB in M-RobustEQGC and the aggregated imbalanced transfer-out bandwidth is about 279.96 MB in M-BasicEQGC and 497 MB in M-RobustEQGC. That is to say, the difference among all ADMs is about 20 MB~50 MB, which is a very small value.

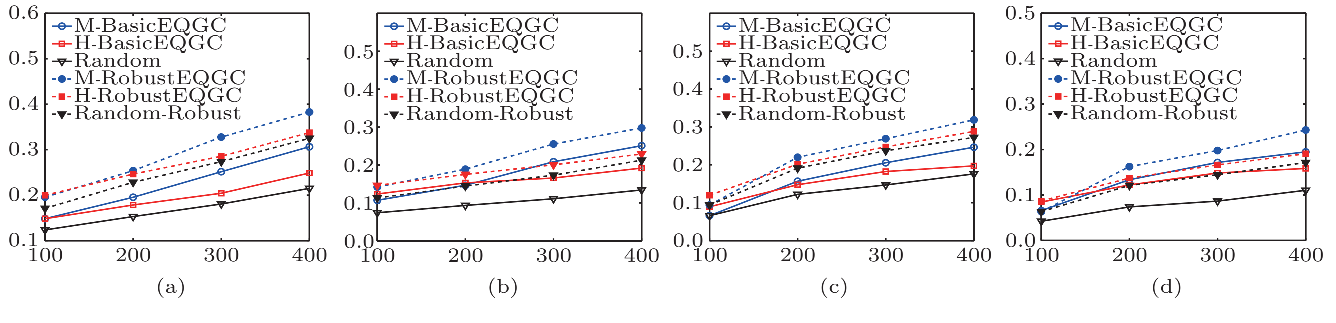


Fig.8. Comparison of the percentage of reduced 1s in the real query traces. The X-axis denotes the number of users and the Y-axis denotes the corresponding percentage. (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

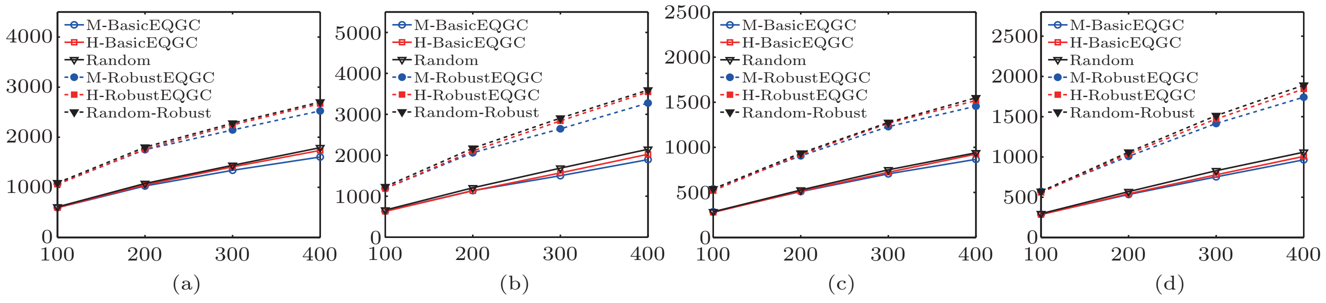


Fig.9. Comparison of the bandwidth at the cloud in the real query traces. The X-axis denotes the number of users and the Y-axis denotes the corresponding bandwidth (MB). (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

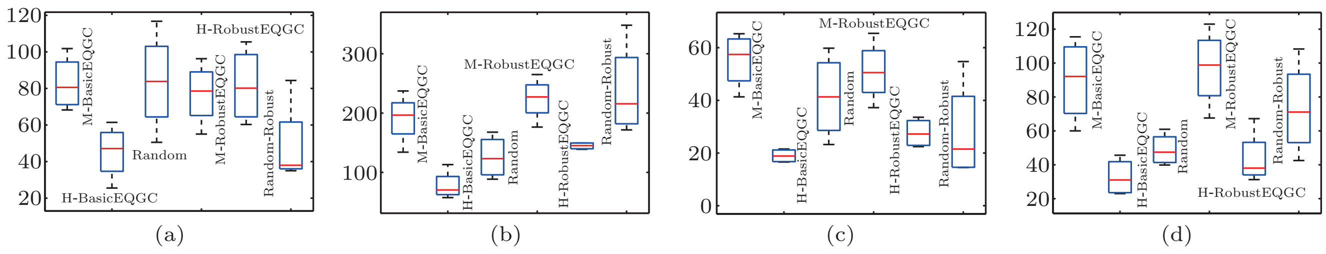


Fig.10. Comparison of the imbalanced transfer-in bandwidth at ADMs in the real query traces. The X-axis denotes the number of users and the Y-axis denotes the corresponding imbalanced bandwidth (MB). (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

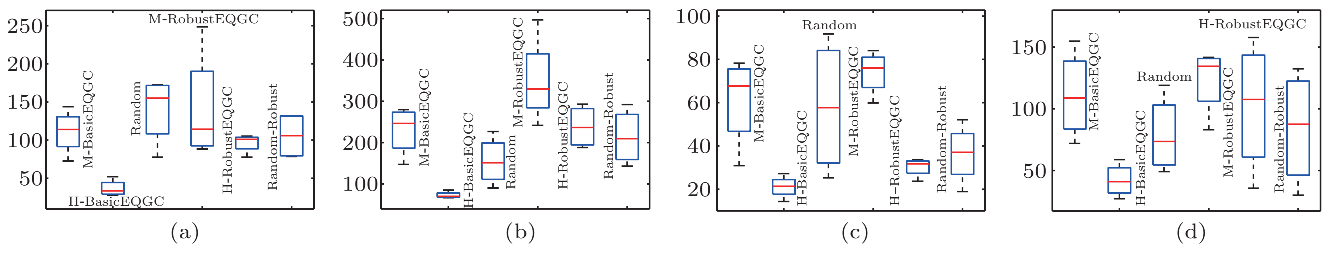


Fig.11. Comparison of the imbalanced transfer-out bandwidth at ADMs in the real query traces. The X-axis denotes the number of users and the Y-axis denotes the corresponding imbalanced bandwidth (MB). (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

Benefits. As in the setting of the synthetic data, the probability for generating "$*$" is set as $\delta = 0.01$. Given $n = \{100, 200, 300, 400\}$, the sum number of "$*$" in all queries is about $\{1\,000, 2\,000, 3\,000, 4\,000\}$ and $\{2\,000, 4\,000, 6\,000, 8\,000\}$ under the setting of $d = 1\,000$ and $d = 2\,000$, respectively. The gained benefits are mainly affected by the number of groups k. As k increases, the gained benefit decreases. Fig.12 shows that the heuristic strategy gains the most benefits and the mathematic strategy gains the least. For example, under the setting $d = 1\,000$, $k = 5$ and $n = 200$, H-BenefitEQGC gains about 175 benefits while incurring 1 049.8 MB bandwidth at the cloud, but M-BenefitEQGC only gains 144 benefits while incurring 1 026.8 MB bandwidth at the cloud.

7.3 Summary

In terms of efficiency, the computational complexity of heuristic grouping is $O(c \times k \times n)$ for the basic version, $O(c \times \alpha \times k \times n)$ for the robust version, and $O(c \times k \times n \times 2)$ for the benefit version where c is the number of iterations, k is the number of groups, n is the number of queries, and α is the number of query copies. Mathematic grouping strategies use the first-order local optimization method to solve the non-convex optimization problem, the computational complexity of which has not been proven. In our experiments, heuristic grouping strategies are very efficient and can output the final grouping results in several milliseconds, but mathematic grouping strategies take more time to terminate.

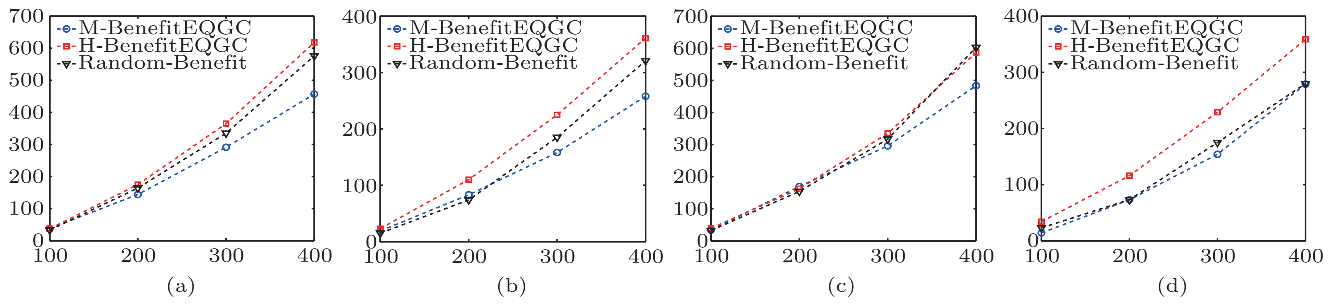


Fig.12. Comparison of the gained benefit in the real query traces. The X-axis denotes the number of users and the Y-axis denotes the corresponding benefit. (a) $d = 1\,000$ and $k = 5$. (b) $d = 1\,000$ and $k = 10$. (c) $d = 2\,000$ and $k = 5$. (d) $d = 2\,000$ and $k = 10$.

In terms of performance, both methods are local optimal. However, mathematic grouping strategies use random restarts to produce multiple rounds to mitigate this problem. The larger the number of random restarts, the better the performance, but the more the execution time. In our experiments, the number of random restarts is fixed to 10. Therefore, the mathematic grouping strategies generate more 1s than heuristic grouping strategies when the number of queries is small. In terms of load balancing, both methods can balance the bandwidth between proxy servers. Therefore, heuristic grouping strategies are more appropriate for cloud environments, and mathematic grouping strategies can be used as the baseline to measure the grouping quality.

8 Related Work

Our work provides effective query grouping strategies in cloud computing for cost efficiency, load balance, robustness, and benefit. The research on designing query grouping strategies for cost-based optimization has recently attracted increased interest[13-16]. For example, Pig[13] supports a large number of sharing mechanisms by executing multiple queries in a single group; MRShare[14] avoids redundant computations in the cloud by sharing work among a group of similar queries; Helix[16] deploys a sharing scheme among the recurring queries based on the sliced window-alignment techniques.

Unlike the above work that focuses on reducing the executing time in the cloud, our work aims to design effective and efficient grouping strategies to minimize the bandwidth consumed at the cloud. Existing research that is the most related to ours can be found in the areas of document clustering and query clustering. Document/query clustering is the process of partitioning a collection into clusters, such that documents/queries in the same cluster are similar among themselves and dissimilar to those belonging to other clusters[17-19].

Clustering algorithms are mainly divided into two categories: hierarchical algorithms and partition algorithms. In summary, hierarchical algorithms tend to be more robust in finding the best clusters, but incur high computational costs. Partition algorithms are more efficient, but may sometimes not be very effective[20]. As the volume of data increases, scalability becomes more and more important for clustering algorithms. Many novel hierarchical algorithms have been developed to cluster large-scale datasets, including Birch[21], CURE[22], Chameleon[23], ROCK[24], etc. For example, Birch utilizes a cluster feature (CF) tree to speed up the clustering process. The CF tree is a height-balanced tree, where the diameter of each entry in the leaf node is less than a threshold value. Given the CF tree, the computational complexity for clustering the leaf nodes can be reduced to $O(n)$, where n is the number of documents/queries.

One of the most widely used partition clustering algorithm is k-means. The computational complexity of k-means is $O(c \times n \times k)$, where c is the number of iterations, n is the number of documents/queries, and k is the number of clusters. The main merit of k-means is that it requires a small number of iterations in order to converge. Observations from [25] show that for many large datasets, 5 or fewer iterations are sufficient for an effective clustering. The main disadvantage of k-means is that it is quite sensitive to the initial set of seeds picked in the initial step[26]. Besides k-means, the other typical partition clustering algorithms include PAM[27], CLARA[28], CLARANS[29], etc. CLARANS, which adopts a serial randomized search strategy, outperforms PAM and CLARA in terms of both clustering quality and execution time[30]. Specifically, it views the clustering process as searching through a certain graph, starts from an arbitrary node in the graph, and randomly selects one of its neighbors. If the cost of the selected neighbor is less than that of the current node, CLARANS sets it as the current node, and continues the neighbor selection process. Otherwise, CLARANS randomly checks another neighbor until a better neighbor is found or the pre-determined maximal number of neighbors to check has been reached. The node with the minimum cost is selected as the final clustering. The computational complexity of CLARANS is $O(k \times n^2)$.

In this paper, k-means is adopted as the building block of our heuristic query grouping strategies due to its efficiency. The main difference between the existing clustering approaches and ours is that our objective is to provide more convenient and more personalized cloud services by achieving cost efficiency, load balancing, robustness, and benefit. Unlike the existing work that aims to maximize the overall similarity with no limit on the size of each group, our basic grouping objective is to minimize the total number of keywords in a combined query under the constraint of equal group size. Although the objective of our grouping strategies is different from the above clustering problems, the key techniques in these fields can be used to improve our work.

9 Conclusions

This paper studied the problem of effective query grouping in cloud computing. To achieve cost efficiency, load balancing, robustness, and benefit, two kinds of grouping strategies were provided: mathematic grouping and heuristic grouping. The experiment results showed that our strategies are practical and applicable to a cloud computing environment.

The experiments assumed that keywords are normally distributed on a set of files, each of which has the same size. In practice, various keyword distributions, e.g., Zipf distribution and Power-Law distribution, also exist in file sets. Therefore, as part of our future work, more experiments will be conducted under different parameter settings. Furthermore, our future work will attempt to deploy QRs and ADMs in an organization and to justify the performance of the grouping strategies in empirical studies.

References

[1] Mell P M, Grance T. The NIST definition of cloud computing. *Communications of the ACM*, 2010, 53(6): Article No. 50.

[2] Fu Z J, Shu J G, Sun X M, Zhang D X. Semantic keyword search based on trie over encrypted cloud data. In *Proc. the 2nd Int. Workshop on Security in Cloud Computing*, June 2014, pp.59-62.

[3] Fu Z J, Ren K, Shu J G, Sun X M, Huang F X. Enabling personalized search over encrypted outsourced data with efficiency improvement. *IEEE Trans. Parallel and Distributed Systems*, 2016, 27(9): 2546-2559.

[4] Liu Q, Tan C C, Wu J, Wang G J. Cooperative private searching in clouds. *Journal of Parallel and Distributed Computing*, 2012, 72(8): 1019-1031.

[5] Liu Q, Tan C C, Wu J, Wang G J. Towards differential query services in costefficient clouds. *IEEE Trans. Parallel and Distributed Systems*, 2014, 25(6): 1648-1658.

[6] Sweeney L. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty Fuzziness and Knowledge-Based Systems*, 2002, 10(5): 557-570.

[7] Niu B, Li Q H, Zhu X Y, Cao G H, Li H. Achieving k-anonymity in privacy-aware location-based services. In *Proc. IEEE INFOCOM*, April 27-May 2, 2014, pp.754-762.

[8] Yi X, Paulet R, Bertino E, Varadharajan V. Practical approximate k nearest neighbor queries with location and query privacy. *IEEE Trans. Knowledge and Data Engineering*, 2016, 28(6): 1546-1559.

[9] Kanungo T, Mount D M, Netanyahu N S, Piatko C D, Silverman R, Wu A Y. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2002, 24(7): 881-892.

[10] Guo Y H. Active instance sampling via matrix partition. In *Proc. NIPS*, December 2010, pp.802-810.

[11] Hamerly G. Making k-means even faster. In *Proc. SIAM Int. Conf. Data Mining*, April 2010, pp.130-140.

[12] Pass G, Chowdhury A, Torgeson C. A picture of search. In *Proc. the 1st Int. Conf. Scalable Information Systems*, May 30-June 1, 2006.

[13] Gates A F, Natkovich O, Chopra S, Kamath P, Narayanamurthy S M, Olston C, Reed B, Srinivasan S, Srivastava U. Building a high-level dataflow system on top of MapReduce: The pig experience. In *Proc. VLDB Endowment*, August 2009, pp.1414-1425.

[14] Nykiel T, Potamias M, Mishra C, Kollios G, Koudas N. MRShare: Sharing across multiple queries in MapReduce. In *Proc. VLDB Endowment*, September 2010, pp.494-505.

[15] Herodotou H, Lim H, Luo G, Borisov N, Dong L, Cetin F B, Babu S. Starfish: A self-tuning system for big data analytics. In *Proc. Biennial Conf. Innovative Data Systems Research*, January 2011, pp.261-272.

[16] Lei C, Zhuang Z F, Rundensteiner E A, Eltabakh M. Shared execution of recurring workloads in MapReduce. In *Proc. VLDB Endowment*, September 2015, pp.714-725.

[17] Aggarwal C C, Zhai C X. A survey of text clustering algorithms. In *Mining Text Data*, Aggarwal C C, Zhai C X (eds.), Springer, 2012, pp.77-128.

[18] Fahad A, Alshatri N, Tari Z, Alamri A, Khalil I, Zomaya A Y, Foufou S, Bouras A. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Trans. Emerging Topics in Computing*, 2014, 2(3): 267-279.

[19] Vu T T, Willis A, Song D W. Modelling time-aware search tasks for search personalisation. In *Proc. the 24th Int. Conf. World Wide Web*, May 2015, pp.131-132.

[20] Zhao Y, Karypis G. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 2004, 55(3): 311-331.

[21] Zhang T, Ramakrishnan R, Livny M. BIRCH: An efficient data clustering method for very large databases. *ACM SIGMOD Record*, 1996, 25(2): 103-114.

[22] Guha S, Rastogi R, Shim K. CURE: An efficient clustering algorithm for large databases. *Information Systems*, 2001, 26(1): 35-58.
[23] Karypis G, Han E H, Kumar V. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 1999, 32(8): 68-75.
[24] Guha S, Rastogi R, Shim K. ROCK: A robust clustering algorithm for categorical attributes. In *Proc. the 15th Int. Conf. Data Engineering*, March 1999, pp.512-521.
[25] Schütz H, Silverstein C. Projections for efficient document clustering. *ACM SIGIR Forum*, 1997, 31(SI): 74-81.
[26] Cutting D R, Karger D R, Pedersen J O, Tukey J W. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proc. the 15th Annual Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, June 1992, pp.318-329.
[27] Sarle W S. Finding groups in data: An introduction to cluster analysis. *Journal of the American Statistical Association*, 1991, 86(415): 830-833.
[28] Ng R J, Han J W. Efficient and effective clustering methods for spatial data mining. In *Proc. the 20th Int. Conf. Very Large Data Bases*, September 1994, pp.144-155.
[29] Ng R T, Han J W. CLARANS: A method for clustering objects for spatial data mining. *IEEE Trans. Knowledge and Data Engineering*, 2002, 14(5): 1003-1016.
[30] Wei C P, Lee Y H, Hsu C M. Empirical comparison of fast clustering algorithms for large data sets. In *Proc. the 33rd Annual Hawaii Int. Conf. System Sciences*, January 2000.

Qin Liu received her B.S. degree in 2004 from Hunan Normal University, Changsha, and her M.S. degree in 2007 and Ph.D. degree in 2012 both from Central South University, Changsha, all in computer science. She was a visiting student at Temple University, Philadelphia. Her research interests include security and privacy issues in cloud computing. Currently she is an assistant professor in the College of Computer Science and Electronic Engineering at Hunan University, Changsha.

Yuhong Guo is an associate professor in the School of Computer Science, Carleton University, Ottawa. She is also a Canada Research Chair in Machine Learning. Previously she worked as a research fellow at the Australian National University, Canberra, and a tenure-track assistant professor and a tenured associate professor at Temple University, Philadelphia. Dr. Guo received her Ph.D. degree in computer science from the University of Alberta, Edmonton, in 2007. Her primary research area is machine learning, with applications in natural language processing, computer vision, and bioinformatics.

Jie Wu is the associate vice provost for international affairs at Temple University, Philadelphia. He also serves as the director of Center for Networked Computing and Laura H. Carnell professor in the Department of Computer and Information Sciences. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University, Boca Raton. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Prof. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing and the Journal of Parallel and Distributed Computing. Prof. Wu was general cochair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Prof. Wu is a CCF distinguished speaker and a fellow of IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Guojun Wang received his B.S. degree in geophysics, in 1992, M.S. degree in computer science, in 1996, and Ph.D. degree in computer science, in 2002, all from Central South University, Changsha. He is the Pearl River scholar professor of Guangdong Province at School of Computer Science and Educational Software, Guangzhou University, Guangzhou. He was an adjunct professor at Temple University, Philadelphia, a visiting scholar at Florida Atlantic University, Boca Raton, a visiting researcher at the University of Aizu, Aizu-Wakamatsu, and a research fellow at the Hong Kong Polytechnic University, Hong Kong. His research interests include network and information security, Internet of Things, and cloud computing. He is a distinguished member of CCF, and a member of IEEE, ACM, and IEICE.