Dynamic Grouping Strategy in Cloud Computing

Qin Liu[†‡], Yuhong Guo[‡], Jie Wu[‡], and Guojun Wang[*†]
[†]*School of Information Science and Engineering*
Central South University, Changsha, Hunan Province, P. R. China, 410083
[‡]*Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA*
**Correspondence to: csgjwang@mail.csu.edu.cn*

***Abstract*—Cloud computing has emerged as a new type of commercial paradigm. As a typical cloud service, each file stored in the cloud is described with several keywords. By querying the cloud with certain keywords, a user can retrieve files whose keywords match his query. An organization that has thousands of users querying the cloud can set multiple proxy servers inside itself to reduce the querying cost. All users can be classified into different groups, and the users in a group will send their queries to the same proxy server, which will query the cloud with a *combined* query, i.e., the union of keywords in a group of queries. In such an environment, an important problem is *cost efficiency*, i.e., how to classify users into different groups so that the total number of returned files is minimized. Observing that this is mainly affected by the number of keywords in the combined queries, our problem is translated to classifying n users into k groups in the case of k proxy servers, so that the number of keywords in k combined queries is minimized. Since more common keywords in a group of queries will generate less keywords in the combined queries, we should group users with the most common keywords together. Two additional aspects needed to be addressed are *load balancing* and *robustness*, i.e., the workloads among proxy servers are balanced and each user obtains search results even if some proxy servers fail. To solve above problems simultaneously, we propose mathematic grouping and heuristic grouping strategies, where mathematic grouping solves the relaxed problem by using a local optimization method, and heuristic grouping is based on the classical heuristic clustering algorithm, *K*-means. Extensive evaluations have been conducted on the analytical model to verify the effectiveness of our strategies.**

***Keywords*-Cloud computing, dynamic grouping, cost efficiency, load balancing, robustness.**

I. Introduction

Cloud computing has emerged as a new type of commercial paradigm due to its overwhelming advantages [1]. Organizations with limited budgets can achieve significant cost savings, scalability, and flexibility by outsourcing their data resources to the cloud. Let us consider the application that is shown in Fig. 1: University A outsources the online library resources to the cloud for easy access by its staff and students. Each file is described with several keywords, and the universal keywords are uniformly distributed in the file set. Each user can query the cloud with certain keywords, and the cloud will process the query on each file and return files whose keywords match the query. When University A has thousands of users querying the cloud, the querying cost will be excessive if each user executes queries individually.

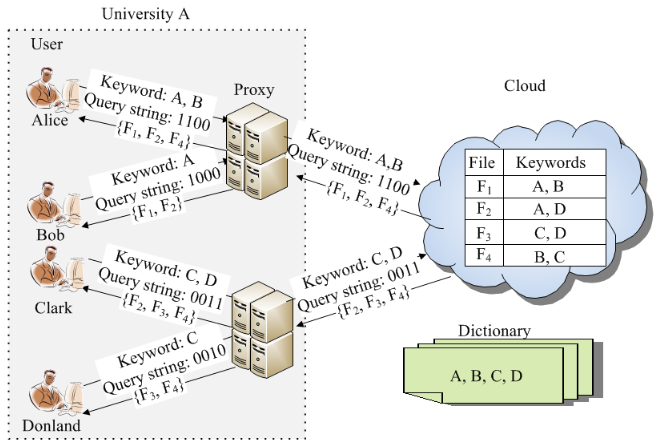


Figure 1. University A outsources online library resources to the cloud. Files F_1, F_2, F_3, and F_4 are described with keywords "A, B", "A, D", "C, D", and "B, C", respectively. There are two proxy servers and four users that query with keywords "A, B", "A" , " C, D", and "C", respectively.

To reduce querying costs, University A can deploy multiple proxy servers inside itself and classify all its users into different groups. The users in a group will send their queries to the same proxy server, which will aggregate a group of user queries and query the cloud with a *combined* query, i.e., the union of keywords in a group of user queries. Take the application in Fig. 1 as an example, without proxy servers, the cloud needs to execute queries four times and return 10 files in total; with two proxy servers, the cloud only needs to execute queries twice and return 6 files in total. In this way, the computation and communication costs incurred at the cloud are saved by 50% and 40%, respectively.

An important problem in this scenario is how to classify users into different groups so that the querying cost incurred at the cloud is minimized. A naïve grouping strategy would be getting each user to send his query to a random proxy server. The main drawback of this simple solution is the waste of unnecessary bandwidth. For example, if users with with no common keywords are randomly grouped together, e.g., Alice and Clark are in a group, and Bob and Donland are in another group, then 25% of the bandwidth is wasted compared to the scenario in Fig. 1. An alternative solution is for all users to send their queries to a single proxy server. The main drawback of this solution is that it can easily trigger a performance bottleneck and has a single point of failure. If the single proxy server fails, all users will lose their search results.

In this paper, we address three important issues in such an environment: *cost efficiency*, *load balancing*, and *robustness*. Cost efficiency refers to minimizing the total number of files returned from the cloud. Note that keywords are uniformly distributed in the file set, and the probability of each keyword in a file is the same. Thus, this problem is equivalent to minimizing the total number of keywords in the combined queries. Since more common keywords in a group of queries will generate less keywords in the combined queries, we need to group users with the most common keywords together. As shown in Fig. 1, the basic idea of our grouping strategy is to construct a public *dictionary* that consists of the universe of keywords. A user query or the combined query is converted to a 0-1 bit string, where a bit is set to 1, only when a corresponding keyword in the dictionary is chosen by the user or by at least one group member. For example, given dictionary that consists of $\langle A, B, C, D\rangle$, Alice's query string is $\langle 1100\rangle$, when she queries with keywords "A, B". The optimization problem is converted to grouping users with the most common keywords together to minimize the number of 1s in each combined query, causing the minimal total number of 1s.

Load balancing refers to balancing bandwidth among proxy servers. For each proxy server, the transfer-in bandwidth is mainly incurred by receiving results from the cloud, and the transfer-out bandwidth is mainly incurred by distributing results to each user. Our solution is to make the number of users in each group (*group size*) to be just the same to balance the dominating transfer-out bandwidth. As an extension, we relax the constraint of equal group size and make the number of keywords in each group to be almost the same to balance the transfer-in bandwidth.

Last, but not least, is robustness, which ensures that each user obtains search results even if some machines fail. Our solution is to generate multiple copies for each query, where each copy will be classified into different groups and sent to different proxy servers. In this way, so long as one proxy server runs, the user will not lose his results. The main merit is to allow a dynamic adjustment of the number of query copies that will be sent, and to make sure that the bandwidth will not grow linearly with the number of copies.

In this paper, we design group strategies to simultaneously solve above three issues. Note that our work is essentially different from content distribution network (CDN) [2]–[4]. CDN is an efficient approach to deliver Web content, where multiple replicas of each content are scattered over the Internet. A request for a single content is routed to its *closest* replica. CDN also considers load balancing among replicas.

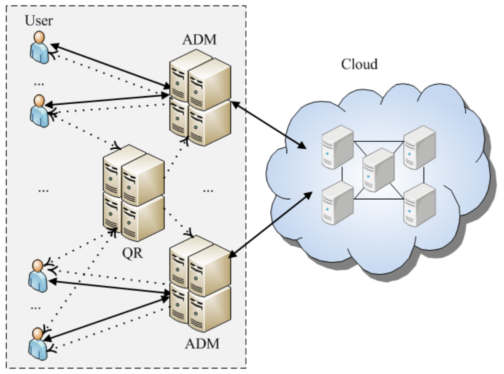


Figure 2. System model.

Our design goal is to equally classify n queries into k groups in the case of k proxy servers, so that the number of 1s in each combined query is minimized. This is similar to clustering and graph cutting problems, which have been proven to be NP-Hard [5]. We propose two kinds of grouping strategies: *mathematic grouping* and *heuristic grouping*. Mathematic grouping formulates the grouping problem in a mathematical way and solves the relaxed problem by using a local optimization method. Heuristic grouping, called *K*-Mean-based Dynamic Grouping (KMDG), is based on the classical heuristic clustering algorithm, *K*-Means [6]. We provide robust versions for both strategies by letting each user generate $2 \leq \alpha \leq k$ query copies with the constraint that α copies of each query cannot be in one group. The robust version of heuristic grouping is called KMDG1. Specifically, heuristic grouping has a second extension, called KMDG2, which relaxes the constraint of equal group size to balance the transfer-in bandwidth. Our key contributions are as follows:

1) To the best of our knowledge, it is the first attempt to devise a dynamic grouping strategy in a cloud.
2) We resolve the grouping problem in both mathematical and heuristic ways to simultaneously achieve cost efficiency, load balancing, and robustness.
3) Extensive experiments were performed on the analytical model to validate our grouping strategies.

II. Preliminaries

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

A. System model

The system consists of three entities: the cloud, many users, and many proxy servers, as shown in Fig. 2. The proxy servers can be classified into query router (QR) and aggregation and distribution machines (ADMs). Once in a while, a QR will be elected from the proxy servers and the remainder proxy servers become ADMs. To ensure the

Table I
Summary of Notations

Notation	Description
n	Number of users in a batch
d	Number of keywords in the dictionary
k	Number of groups
g_j, s_j	Group j, Seed of group j
Q_i, $\hat{Q}_j$	User i's query, Combined query of g_j
Q	An $n \times d$ matrix contains n users queries
S_i, $\hat{S}_j$	Number of 1s in Q_i, $\hat{Q}_j$

normal operation of the system, a new QR should be elected if current QR fails in an interval. The length of an interval may depend on the scale of the organization, which is out of the scope of this paper. The most relevant notations in this paper are shown in Table I.

The cloud holds a collection of t files $\{F_1, \ldots, F_t\}$, where each file can be described by a set of distinct keywords. The universe of d keywords form a *dictionary* that is publicly available. The main functionality of the cloud is to process queries to return search results to the ADMs.

The ADM has two functionalities: querying the cloud on behalf of a group with a combined query, and distributing results to each user in a group. User i's query Q_i or group j's combined query $\hat{Q}_j$ is a 0-1 bit string of length d, where the *l-th* bit is 1 only if the *l-th* keyword in the dictionary is chosen by user i or at least one user in group j.

All users will first send their queries to the QR. If a user cannot receive response from the QR in a period of time, he will send his query to a random ADM. The QR will wait for a period of time to aggregate enough queries, and dynamically classify n users into k groups based on user queries. Then, the QR will respond each user and send user queries to different ADMs based on the grouping decision.

Note that our system will incur some processing delay since the QR should batch enough queries to make grouping decision. However, the degree of aggregation can be controlled through a time-out mechanism to meet a given processing delay requirement, e.g., thousands of users may query the cloud in a sequence of batches, where each batch has hundreds of users. The length of the time-out is affected by many factors, such as transfer delay and system load.

B. Design goals

To enable our grouping strategies to work well, our design should simultaneously ensure the following guarantees:

- Effectiveness: obtain optimal grouping results within a polynomial time.
- Cost efficiency: minimize the bandwidth at the cloud.
- Load balancing: balance the bandwidth among ADMs.
- Robustness: make sure each user obtains search results even if some machines fail.

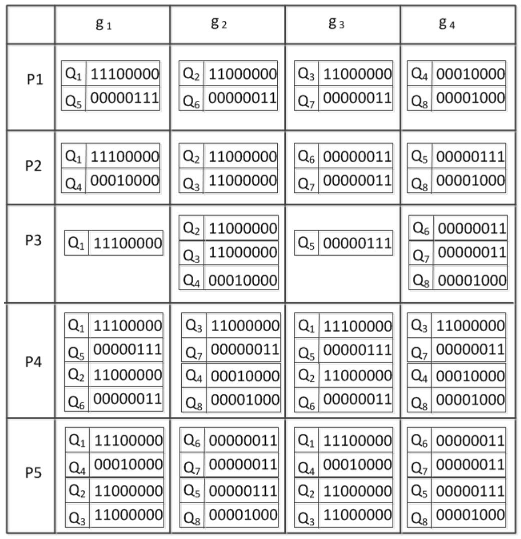
To illustrate our design goals, we provide the following example, which will be used as the sample application in this paper. The example assumes that the dictionary consists of $\langle A, B, C, D, E, F, G, H \rangle$, and files stored in the cloud are as shown in Table II. Suppose that there are four ADMs, $ADM_1, \ldots, ADM_4$, and eight queries, as shown in Table III. All users are classified into $k = 4$ groups $g_1, \ldots, g_4$, and all queries in g_j are sent to ADM_j, where $1 \leq j \leq k$.

Table II
Sample files

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

Table III
Sample user queries

$Q_1 = \langle 11100000 \rangle \Rightarrow$ (A,B,C)	$Q_5 = \langle 00000111 \rangle \Rightarrow$ (F,G,H)
$Q_2 = \langle 11000000 \rangle \Rightarrow$ (A,B)	$Q_6 = \langle 00000011 \rangle \Rightarrow$ (G,H)
$Q_3 = \langle 11000000 \rangle \Rightarrow$ (A,B)	$Q_7 = \langle 00000011 \rangle \Rightarrow$ (G,H)
$Q_4 = \langle 00010000 \rangle \Rightarrow$ (D)	$Q_8 = \langle 00001000 \rangle \Rightarrow$ (E)



	g_1	g_2	g_3	g_4
P1	Q_1 11100000; Q_5 00000111	Q_2 11000000; Q_6 00000011	Q_3 11000000; Q_7 00000011	Q_4 00010000; Q_8 00001000
P2	Q_1 11100000; Q_4 00010000	Q_2 11000000; Q_3 11000000	Q_6 00000011; Q_7 00000011	Q_5 00000111; Q_8 00001000
P3	Q_1 11100000	Q_2 11000000; Q_3 11000000; Q_4 00010000	Q_5 00000111	Q_6 00000011; Q_7 00000011; Q_8 00001000
P4	Q_1 11100000; Q_5 00000111; Q_2 11000000; Q_6 00000011	Q_3 11000000; Q_7 00000011; Q_4 00010000; Q_8 00001000	Q_1 11100000; Q_5 00000111; Q_2 11000000; Q_6 00000011	Q_3 11000000; Q_7 00000011; Q_4 00010000; Q_8 00001000
P5	Q_1 11100000; Q_4 00010000; Q_2 11000000; Q_3 11000000	Q_6 00000011; Q_7 00000011; Q_5 00000111; Q_8 00001000	Q_1 11100000; Q_4 00010000; Q_2 11000000; Q_3 11000000	Q_6 00000011; Q_7 00000011; Q_5 00000111; Q_8 00001000

Figure 3. Sample group patterns.

We provide 3 group patterns, (P1, P2, P3), for a single query copy, and two group patterns, (P4, P5), for the robust version, (two query copies), as shown in Fig. 3. P1 and P4 are the instances of the basic version and robust version of random grouping, respectively; P2 and P5 are the instances of the KMDG and KMDG1; P3 is the instance of KMDG2.

Cost efficiency. This design goal is equivalent to grouping users with the most common keywords together to minimize the total number of 1s in the combined queries. For example, in P1, where users with no common keywords are grouped together, the number of 1s in each group are 6, 4, 4, 2, respectively, and the total number of 1s is 16. For file set in Table II, the cloud returns 600 files to ADM_1, 400 files to ADM_2, 400 files to ADM_3, and 200 files to ADM_4, thus the total number of returned files is 1,600; in P2 and P3, where users with more common keywords are grouped together, the total number of 1s is 12. For file set in Table II, the cloud returns 1,200 files, saving 25% of the bandwidth.

Load balancing. KMDG balances the transfer-out bandwidth among ADMs by making each group size just the same. KMDG2 balances the transfer-in bandwidth among ADMs by relaxing the constraint of equal group size to make the number of 1s in each combined query basically the same. For example, each group has 2 members in P2, but the group size of g_2 and g_4 is triple that of g_1 and g_3 in P3. For file set in Table II, in P2, each ADM needs to return 400 files to all users in a group and consumes the same transfer-out bandwidth, however, ADM_1 and ADM_4 need to receive 400 files from the cloud, which consume double the transfer-in bandwidth compared to ADM_2 and ADM_3; In P3, each ADM receives 300 files from the cloud and consumes the same transfer-in bandwidth, however, ADM_2 and ADM_4 need to return 500 files to all users in a group, which consume 1.7 times the transfer-out bandwidth compared to ADM_1 and ADM_3.

Robustness. To ensure that each user obtains search results even some machines fails, we generate multiple copies for each query, with the constraint that each copy will be classified into different groups. Our solution still groups users with the most common keywords together, and thus the increased bandwidth will not grow linearly with the number of copies. For example, in P4, the cloud needs to return 2,400 files, which is double that of P2; in P5, the cloud needs to return 1,600 files, and the bandwidth only increases 33% compared to P2.

C. Parameter analysis

We first analyze which factors will impact cost efficiency and load balancing.

Cost efficiency refers to minimizing the number of files returned from the cloud. Suppose that the dictionary contains d keywords, and each file is described by γ keywords. The probability of a keyword in a file is γ/d. Then, the expected value of the number of returned files can be calculated with $\sum_{j=1}^{k} t \cdot (1-(1-\gamma/d)^{\hat{S}_j})$, 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 1s in group j's combined query $\hat{Q}_j$. Given that t, k, γ, and d are fixed, the total number of returned files depends on $\hat{S}_j$ where $1 \le j \le k$. Therefore, cost-efficiency is equivalent to minimizing the number of 1s in each combined query.

Load balancing refers to balancing the transfer-in and transfer out bandwidth among ADMs. The transfer-in bandwidth at each ADM is mainly incurred by receiving files from the cloud. As described above, the number of returned files depends on the number of 1s in each combined query. The transfer-out bandwidth at each ADM is mainly incurred by transferring files to each user in a group. Suppose that the dictionary contains d keywords, and each file is described by γ keywords. The probability of a keyword in a file is γ/d. Then, for ADM_j that combines group j's queries, the estimated value of the number of files that are returned to group j can be calculated with $\sum_{i}^{|g_j|} t \cdot (1-(1-\gamma/d)^{S_i}$, where t is the number of files stored in the cloud, $|g_j|$ is the number of users in group j, and S_i is the number of 1s in user i's query Q_i. Given that t, γ, and d are fixed, and S_i is basically the same, transfer-out bandwidth mainly depends on the group size. Therefore, achieving load balancing is equivalent to balancing the number of 1s in each combined query and group size among ADMs.

III. Mathematic grouping strategy

In this section, we provide a basic mathematic grouping strategy and will provide the robust version in Section V.

Let Q be an $n \times d$ matrix, representing n query strings with length d. We consider 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 $Y \in \{0,1\}^{n\times k}$ denote the grouping setting, then the problem can be formulated into the following optimization

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

where $\mathbf{1}$ denotes a vector of all 1s, assuming that its length can be determined from the context; E denotes a $k\times d$ matrix of all 1s; $\delta(\cdot)$ denotes an indicator function. The problem we formulated above, however, is *NP-hard*, and it is difficult to conduct optimization directly over it. We thus propose to solve a relaxation problem instead. Firstly, we approximate the indicator matrix, $\delta(Y^\top Q)$, with a smooth function, $1-\exp(-\beta Y^\top Q)$, where β is a large constant number, e.g., $\beta = 10, 20, 30, 40$. We then relax the integer matrix Y into a continuous matrix. After relaxation, we obtain the following optimization problem:

$$\min_{Y} \quad tr(E^\top (1-\exp(-\beta Y^\top Q))) \qquad (2)$$
$$0 \le Y \le 1, \quad Y^\top \mathbf{1} = \frac{n}{k}\mathbf{1}, \quad Y\mathbf{1} = \mathbf{1}$$

which is equivalent to:

$$\max_{Y} \quad tr(E^\top \exp(-\beta Y^\top Q)) \qquad (3)$$
$$0 \le Y \le 1, \quad Y^\top \mathbf{1} = \frac{n}{k}\mathbf{1}, \quad Y\mathbf{1} = \mathbf{1}$$

Let $f(Y)$ denote the objective function in (3). Then $f(Y)$ is a convex function of Y. However, maximization over a convex function is a non-convex optimization problem. We use a first-order local optimization method to conduct optimization, which is similar to our previous work [7]. The gradient can be computed as $\nabla_Y f(Y) = -\beta Q \exp(-\beta Q^\top Y)$.

Algorithm 1 KMDG

```
1: Construct a set, $CandiQ$, with the universal user queries
2: Randomly choose $k$ distinct queries from $CandiQ$ as seeds $s_1, \ldots, s_k$ for groups $g_1, \ldots, g_k$, and remove them from $CandiQ$
   {Runs the following process multiple rounds}
3: while $CandiQ$ is not empty do
4:   for $j = 1$ to $k$ do
5:     $Neighbor_j$ is a subset of $CandiQ$ that accommodate $s_j$'s nearest neighbors
6:     Choose a random element $Q_i \in Neighbor_j$ into $g_j$ and remove it from $CandiQ$
7: Initialize $CandiQ$ with the universal user queries
8: for $j = 1$ to $k$ do
9:   Randomly choose a query from $g_j$ as the seed $s_j$ for the next round and remove it from $CandiQ$
```

After obtaining a continuous solution, Y^*, to (3), we can then round it back to a feasible integer matrix by using a heuristic greedy procedure: in each iteration, we find the 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 jth column, we set $Y_{ij} = 1$, and we set all the other entries on the same row to 0. We then remove this row from further consideration. If there are already n/k 1s in the jth column, we set $Y_{ij} = 0$ and go to the next iteration. When the maximum Y_{ij} returned is 0, we complete the procedure.

The solution we obtained after using the proposed optimization method above is a local optimal solution. To overcome the drawback of local optima, we use random restarts to produce multiple local optimal solutions, and we pick the best one.

IV. Heuristic grouping strategy

In this section, we will first provide related definitions as background knowledge, and then we describe a basic heuristic grouping (KMDG). We will describe the robust version (KMDG1) in Section V and an additional extension (KMDG2) in Section VI.

For ease of illustration, we use the example in Section II to illustrate the following definitions.

Group seed. *For $1 \leq j \leq k$, the group seed s_j is the center and first member of group g_j.*

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 $Q_1 = \langle 11100000 \rangle$ and $Q_2 = \langle 11000000 \rangle$, $Dist(Q_1, Q_2) = 0$, and $Dist(Q_2, Q_1) = 1$.

Nearest neighbor. *Given a query Q_j, query Q_i with the minimal distance from Q_j is called as Q_j's nearest neighbor.*

Algorithm 2 KMDG1 (Robust version of KMDG)

```
1: Construct a set, $CandiQ$, with $2 \leq \alpha \leq k$ query copies
2: Line 2 in Alg. 1
   {Runs the following process multiple rounds}
3: while $CandiQ$ is not empty do
4:   for $j = 1$ to $k$ do
5:     $Neighbor_j$ is a subset of $CandiQ$ that accommodate $s_j$'s nearest neighbors
6:     Choose a random element $Q_i \in Neighbor_j$ and $Q_i \notin g_j$ into $g_j$ and remove it from $CandiQ$
7: Initialize $CandiQ$ with $2 \leq \alpha \leq k$ query copies
8: Line 8 to line 9 in Alg. 1
```

Take the queries in Table III 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.

The basic idea of KMDG is to classify the top n/k nearest neighbors of the group seed s_j in group g_j to minimize the total number of 1s in the combined query $\hat{Q}_j$. The grouping strategy (Alg. 1) generally has two steps. 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 classifying $n - k$ queries into k groups based on k seeds. Specifically, given seed s_j, a set $Neighbor_j$, that accommodates the nearest neighbors of s_j, is first constructed. Then, we classify a random element in $Neighbor_j$ into g_j. Note that there will be a conflict when a query, Q_i, is the nearest neighbor of $1 < m \leq k$ seeds, e.g., $s_1, \ldots, s_m$. We resolve this conflict by classifying Q_i into $Neighbor_j$ with $Dist(Q_i, s_j) \leq Dist(Q_i, s_l)$ for $1 \leq j, l \leq m$. The above two steps will be run within multiple rounds, and in each round, the grouping result will be recorded. At the end of this algorithm, the optimal result will be the output.

V. Extensions

In this section, we provide robust versions for both mathematic grouping and heuristic grouping. For mathematic grouping, we only provide the construction for two query copies. However, our grouping strategy can be easily adapted to more than two query copies.

Mathematic grouping. Now we assume we have two copies of Q; assigning the two copies of each query string to different groups is required. Let $Z = [Q; Q]$ be a $2n \times d$ matrix, we then have:

$$\min_{Y} \quad tr(E^\top \delta(Y^\top Z)) \tag{4}$$
$$Y \in \{0,1\}^{2n \times k}, Y^\top \mathbf{1} = \frac{2n}{k}\mathbf{1}, Y\mathbf{1} = \mathbf{1}, AY \leq 1$$

where $A = [I, I]$, and I is an $n \times n$ identity matrix. The relaxed optimization problem is:

$$\max_{Y} \quad tr(E^{\top} \exp(-\beta Y^{\top} Z)) \tag{5}$$

$$0 \le Y \le 1, \quad Y^{\top}\mathbf{1} = \frac{2n}{k}\mathbf{1}, \quad Y\mathbf{1} = \mathbf{1}, \quad AY \le 1$$

The heuristic greedy rounding procedure can be updated correspondingly: in each iteration, we find the 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 jth column, we set $Y_{ij} = 1$, $Y_{\hat{i}j} = 0$ (if $i > t$, then $\hat{i} = i - n$, otherwise $\hat{i} = i + n$), set all of the other entries on the ith row to 0, and remove this row from further consideration. If there are already $2n/k$ 1s in the jth column, we set $Y_{ij} = 0$ and go to the next iteration. When the maximum Y_{ij} returned is 0, we complete the procedure.

Heuristic grouping. Heuristic grouping can be extended to a robust version, as in Alg. 2, with the constraint that each copy should be classified into different groups.

Note that Alg. 2 will cause a *dilemma*: a query cannot be grouped unless we break our constraint. To illustrate, let us assume that $g_1 = Q_4, Q_5, Q_6, Q_7$, $g_2 = Q_4, Q_5, Q_8$, $g_3 = Q_6, Q_7, Q_8$, and $g_4 = Q_1, Q_2, Q_3$. Thus, we need to classify Q_1, Q_2, Q_3 into g_2, g_3, g_4. However, the constraint that each copy should be classified into different groups will be broken, no matter what we classify (either Q_1, Q_2, or Q_3) in g_4. The reason for this dilemma is that the strategy requires each group size to be just the same and some queries which are the farthest neighbors of $k-1$ group seeds are classified into the same group. To avoid this dilemma, we should track the grouping process and make different decision at the critical point.

VI. Discussion

In this section, we will provide an additional extension, KMDG2, for heuristic grouping. We first provide the following definition:

Friendliest neighbor. *Given group g_j with seed s_j, Query Q_i that is the nearest neighbor of s_j and causes the minimal number of 1s after combining Q_i into g_j is g_j's friendliest neighbor. The number of 1s after combining a friendliest neighbor to g_j is called group cost, denoted as $\hat{C}_j$.*

Take the queries in Table III as an example, if Q_2 and Q_3 are in g_1 with Q_2 as the seed, then the friendliest neighbor of g_1 include Q_1, Q_4, Q_8, and related group cost $\hat{C}_1$ is 3.

KMDG2 relaxes the constraint of equal group size to balance the transfer-in bandwidth among ADMs. Since KMDG is a special case of KMDG2, KMDG2 can perform better. The basic idea of KMDG2 is to classify a group's friendliest neighbors into this group to minimize the number of 1s in the combined queries.

KMDG2 (Alg. 3) includes two steps: (1) choose k group seeds; (2) classify $n - k$ queries into k groups. The first step is the same as in KMDG, and the second step is changed as follows: Given a group g_j with seed s_j, a set $Friend_j$ that accommodates the friendliest neighbors of g_j is first generated. The related group cost $\hat{C}_j$ should also be recorded. Then, a random element in $Friend_j$, with $\hat{C}_j \le \hat{C}_l$ for $1 \le j, l \le k$, will be grouped into g_j. Note that a query, Q_i may be the friendliest neighbor of $1 < m \le k$ groups, e.g., $g_1, \ldots, g_m$. However, this will not cause a conflict since each time only one query that causes the minimal group cost will be grouped.

Algorithm 3 KMDG2 (Balancing transfer-in bandwidth)

```
1: Line 1 to line 2 in Alg. 1
   {Runs the following process multiple rounds}
2: while $CandiQ$ is not empty do
3:    for $j = 1$ to $k$ do
4:       $Friend_j$ is a subset of $CandiQ$ that accommodate
         friendliest neighbors of $g_j$ and $\hat{C}_j$ is group cost
5:    if $\hat{C}_j \le \hat{C}_l$ where $1 \le j, l \le k$ then
6:       Add a random query in $Friend_j$ to $g_j$
7: Line 7 to line 9 in Alg. 1
```

Table IV
Parameters

Notation	Description	Value
$\|F\|$	File size	$500KB$
n	Number of users in a batch	1-200
d	Number of keywords in the dictionary	100
t	Number of files stored in the cloud	$1,000$
k	Number of groups	5, 10
r	Number of rounds	500
S_i	Number of 1s/keywords in Q_i	1-5
α	Number of query copies	2
γ	Number of keywords in each file	5

VII. Evaluation

In this section, we will compare the proposed strategies from the following aspects: the bandwidth incurred at the cloud and the load balancing among ADMs. Our simulations are conducted with MATLAB R2010a, running on a local machine with an Intel Core 2 Duo E8400 3.0 GHz CPU and 8 GB RAM. We summarize the parameters in Table IV. For the ease of comparison, mathematic grouping and its robust version are denoted as Optimization and Optimization Robust; random grouping and its robust version are denoted as Random and Random Robust; KMDG's robust version (KMDG1) is denoted as KMDG Robust.

A. Performance

We first compare the total number of 1s in all combined queries in different grouping strategies under different numbers of groups k. In Fig. 4, we know that our strategies generate less number of 1s than random grouping in the case of a single query copy and the robust version. Take the setting of $d = 100$ and $k = 5$ as an example, KMDG can reduce the amount of number 1s by 8.4% under a

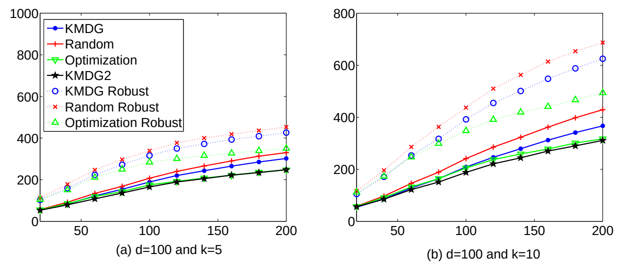


Figure 4. Comparison of total number of 1s in all combined queries. X-axis denotes the number of users and Y-axis denotes the number of 1s.

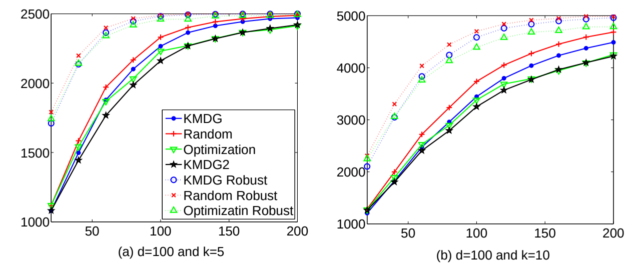


Figure 5. Comparison of bandwidth at the cloud. X-axis denotes the number of users and Y-axis denotes bandwidth at the cloud (MB).

single query copy, and by 8.9% under the robust version, compared to Random. Furthermore, as the number of groups k increases, heuristic grouping works better. For example, as the number of groups increases from 5 to 10, under the setting of $n = 200$ and $d = 100$, the percentage of reduced 1s increases to 13% and 15% in the case of a single query copy and the robust version, respectively.

We also observe that (1) KMDG2 performs the best; (2) While the number of users exceeds 100, Optimization works better than KMDG. The reason of (1) is that Optimization and KMDG, which require equal group size, are special cases of KMDG2. The reason of (2) is that heuristic grouping is apt to obtain the local optimal result while the number of users is sufficiently large. Optimization can relieve this problem by setting multiple random restarting points. Then, we compare the bandwidth incurred at the cloud. We assume that keywords are *uniformly* distributed in file set. From Fig. 5, we observe that our strategies can save more bandwidth. However, as the number of files that have been returned to the ADMs approaches 1,000, all strategies perform similarly.

B. Load balancing

The transfer-in bandwidth at each ADM is mainly affected by the amount of 1s in the combined query, and the transfer-out bandwidth at each ADM is mainly affected by the sum of 1s in a group of queries. We will first compare the imbalanced number of 1s by calculating the average differences of 1s between any two ADMs.

From Figs. 6 and 7, we know that heuristic grouping can

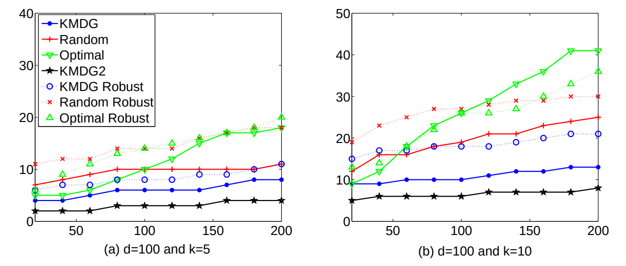


Figure 6. Comparison of imbalanced number of 1s. X-axis denotes the number of users and Y-axis denotes the imbalanced number of 1s.

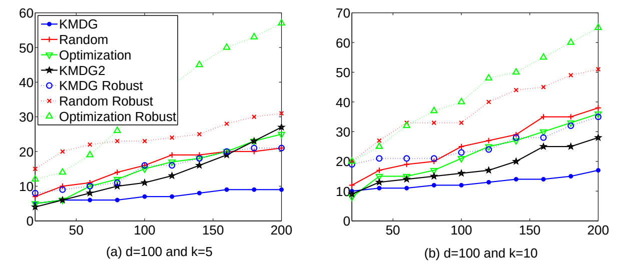


Figure 7. Comparison of imbalanced sum of 1s. X-axis denotes the number of users and Y-axis denotes the imbalanced number of 1s.

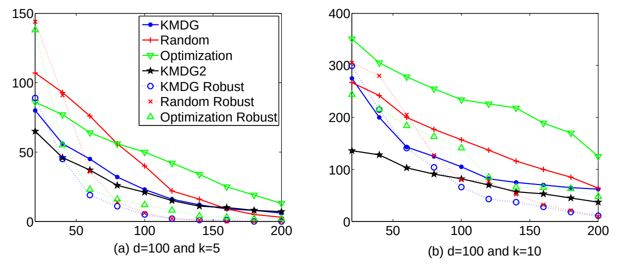


Figure 8. Comparison of transfer-in bandwidth. X-axis is the number of users and Y-axis is the imbalanced transfer-in bandwidth (MB).

achieve better results than random grouping in both a single query copy and the robust version. Furthermore, KMDG is generally more effective than KMDG2 in balancing the sum of 1s in a group of queries, and KMDG2 is generally more effective than KMDG in balancing the number of 1s in the combined query. Optimization and Optimization Robust have worse balancing results.

Then, we compare the imbalanced transfer-in and transfer-out bandwidth by calculating the average differences of bandwidth between any two ADMs. In Figs. 8 and 9, we know that KMDG is generally more effective than KMDG2 in balancing the transfer-out bandwidth, and KMDG2 is generally more effective than KMDG in balancing the transfer-in bandwidth. Furthermore, Optimization and Optimization Robust still have the worse results. However, as the number of users increases, the number of files returned to each

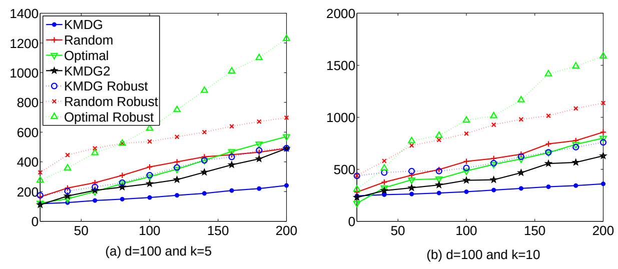


Figure 9. Comparison of transfer-out bandwidth. X-axis is the number of users and Y-axis is the imbalanced transfer-out bandwidth (MB).

ADM will approach 1,000, and the imbalanced transfer-in bandwidth among ADMs will decrease.

VIII. Related work

Our work provides feasible grouping strategies in cloud computing to simultaneously achieve cost efficiency, load balance, and robustness. To the best of our knowledge, no previous works have addressed this problem. Existing research that is the most related to ours can be found in the areas of K-means [6], [8]–[14].

K-means, as a building block of the heuristic grouping strategy, is closely related to a number of other clustering and location problems, such as Euclidean K-medians [15] and the geometric K-center problem [16]. In statistics and data mining, K-means clustering is a method of cluster analysis which aims to partition n members into k clusters so that each member belongs to the cluster with the nearest mean [9].

Although this problem is computationally difficult (*NP-hard*) [5], there are efficient heuristic algorithms that are commonly employed that converge quickly to a local optimum [17]. For example, [6] solved the K-means problem by using a simple iterative scheme for finding a locally minimal solution; [10] solved the K-means problem for scalar data based on the simple observation that the optimal placement of a center is at the centroid of the associated cluster; [11] proposed an asymptotically efficient approximation for the K-means clustering problem; [12] provided an efficient implementation of the K-means algorithm in [10].

IX. Conclusion

In this paper, we study the problem of dynamic grouping in cloud computing. To simultaneously achieve cost efficiency, load balancing, and robustness, we propose two kinds of grouping strategies: mathematic grouping and heuristic grouping. Extensive experiments have been performed to verify the effectiveness of our strategies. For our future work, we plan to improve our work in the follows aspects: First, we will try to conduct experiments on other keyword distributions in the file set to test our strategies; Second, we will try to improve the heuristic grouping strategy to accelerating convergence.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "A view of cloud computing," *Communications of the ACM*, 2010.

[2] K. Johnson, J. Carr, M. Day, and M. Kaashoek, "The measured performance of content distribution networks," *Computer Communications*, 2001.

[3] J. Kangasharju, J. Roberts, and K. Ross, "Object replication strategies in content distribution networks," *Computer Communications*, 2002.

[4] M. Pathan and R. Buyya, "A taxonomy of CDNs," *Content delivery networks*, 2008.

[5] R. Michael and D. Johnson, "Computers and intractability: A guide to the theory of NP-Completeness," *WH Freeman & Co.*, 1979.

[6] E. Forgy, "Cluster analysis of multivariate data: efficiency versus interpretability of classifications," *Biometrics*, 1965.

[7] Y. Guo, "Active instance sampling via matrix partition," in *Proc. of NIPS*, 2010.

[8] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[9] J. Hartigan and M. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society*, 1979.

[10] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, 1982.

[11] J. Matousek, "On approximate geometric k-clustering," *Discrete & Computational Geometry*, 2000.

[12] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[13] G. Frahling and C. Sohler, "A fast k-means implementation using coresets," in *Proc. of the ACM Symposium on Computational Geometry*, 2006.

[14] D. Arthur, B. Manthey, and H. Roglin, "K-means has polynomial smoothed complexity," in *Proc. of IEEE Symposium on FOCS*, 2009.

[15] S. Arora, P. Raghavan, and S. Rao, "Approximation schemes for euclidean k-medians and related problems," in *Proc. of ACM symposium on Theory of computing*, 1998.

[16] P. Agarwal and C. Procopiuc, "Exact and approximation algorithms for clustering," *Algorithmica*, 2002.

[17] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The planar k-means problem is *NP-hard*," *WALCOM: Algorithms and Computation*, 2009.