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 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.

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 \( (A, B, C, D) \), Alice’s query string is \( (1100) \), 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.

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 \( \alpha \) 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 (ADM). Once in a while, a QR will be elected from the proxy servers and the remainder proxy servers become ADMs. To ensure the
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 \( Q_j \) 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 \( 1 \leq j \leq k \).

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 \( ADM_4 \),

### Table I

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>Number of users in a batch</td>
</tr>
<tr>
<td>( d )</td>
<td>Number of keywords in the dictionary</td>
</tr>
<tr>
<td>( k )</td>
<td>Number of groups</td>
</tr>
<tr>
<td>( g_j, s_j )</td>
<td>Group ( j )'s Seed of group ( j )</td>
</tr>
<tr>
<td>( Q_i, Q_j )</td>
<td>User ( i )'s query, Combined query of ( g_j )</td>
</tr>
<tr>
<td>( Q )</td>
<td>An ( n \times d ) matrix contains ( n ) users queries</td>
</tr>
<tr>
<td>( S_i, S_j )</td>
<td>Number of 1s in ( Q_i, Q_j )</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>Sample files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>H</td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th>Sample user queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
</tr>
<tr>
<td>( Q_1 = \langle 11100000 \rangle \Rightarrow \langle A,B,C \rangle )</td>
</tr>
<tr>
<td>( Q_2 = \langle 11100000 \rangle \Rightarrow \langle A,B \rangle )</td>
</tr>
<tr>
<td>( Q_3 = \langle 11100000 \rangle \Rightarrow \langle A,B \rangle )</td>
</tr>
<tr>
<td>( Q_4 = \langle 00011000 \rangle \Rightarrow \langle D \rangle )</td>
</tr>
</tbody>
</table>

**Figure 3. Sample group patterns.**
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 $g_2$ 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 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 $\gamma$ keywords. The probability of a keyword in a file is $\gamma/d$. Then, for ADM, 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 $t$, $\gamma$, 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 problem:

$$\min_{Y} \quad tr(E^T \delta(Y^T Q))$$

$$Y \in \{0, 1\}^{n \times k}, \quad Y^T 1 = \frac{n}{k}1, \quad Y 1 = 1$$

where $1$ 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, $\delta(Y^T Q)$, with a smooth function, $1 - \exp(-\beta Y^T Q)$, where $\beta$ 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:

$$\min_{Y} \quad tr(E^T (1 - \exp(-\beta Y^T Q)))$$

$$0 \leq Y \leq 1, \quad Y^T 1 = \frac{n}{k}1, \quad Y 1 = 1$$

which is equivalent to:

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

$$0 \leq Y \leq 1, \quad Y^T 1 = \frac{n}{k}1, \quad Y 1 = 1$$

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 $\nabla_Y f(Y) = -\beta Q \exp(-\beta Q^T Y)$. 

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 $\gamma$ keywords. The probability of a keyword in a file is $\gamma/d$. Then, for ADM, 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 $t$, $\gamma$, 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.

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.
Algorithm 1 KMDG

1: Construct a set, $\text{CandiQ}$, with the universal user queries
2: Randomly choose $k$ distinct queries from $\text{CandiQ}$ as seeds $s_1, \ldots, s_k$ for groups $g_1, \ldots, g_k$, and remove them from $\text{CandiQ}$

{Runs the following process multiple rounds}
3: while $\text{CandiQ}$ is not empty do
4:   for $j = 1$ to $k$ do
5:     $\text{Neighbor}_j$ is a subset of $\text{CandiQ}$ that accommodate $s_j$’s nearest neighbors
6:     Choose a random element $Q_i \in \text{Neighbor}_j$ into $g_j$ and remove it from $\text{CandiQ}$
7: end while
8: Initialize $\text{CandiQ}$ with the universal user queries
9: for $j = 1$ to $k$ do
10:   Randomly choose a query from $g_j$ as the seed $s_j$ for the next round and remove it from $\text{CandiQ}$

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 $j$th 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 $j$th 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 $\text{Dist}(Q_i, Q_j)$, is the number of increased 1s for $Q_i$ after combining with $Q_j$.

For example, if $Q_1 = \{1100000\}$ and $Q_2 = \{11100000\}$, $\text{Dist}(Q_1, Q_2) = 0$, and $\text{Dist}(Q_2, Q_1) = 1$.

**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, $\text{CandiQ}$, with $2 \leq \alpha \leq k$ query copies
2: Line 2 in Alg. 1
3: {Runs the following process multiple rounds}
4: while $\text{CandiQ}$ is not empty do
5:   Neighbor is a subset of $\text{CandiQ}$ that accommodate $s_j$’s nearest neighbors
6:   Choose a random element $Q_i \in \text{Neighbor}_j$ and $Q_i \notin g_j$ into $g_j$ and remove it from $\text{CandiQ}$
7:   Initialize $\text{CandiQ}$ 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 $Q_j$. 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 $\text{Neighbor}_j$, that accommodates the nearest neighbors of $s_j$, is first constructed. Then, we classify a random element in $\text{Neighbor}_j$ 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 $\text{Neighbor}_j$ with $\text{Dist}(Q_i, s_j) \leq \text{Dist}(Q_i, s_l)$ 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 mathematical grouping and heuristic grouping. For mathematical 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:

$$\min_Y \text{tr}(E^T \delta(Y^T Z))$$

$$Y \in \{0,1\}^{2n \times k}, Y^T 1 = \frac{2n}{k}, Y 1 = 1, AY \leq 1$$

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

$$\max_Y \quad tr(E^T \exp(-\beta Y^T Z))$$

$$0 \leq Y \leq 1, \quad Y^T 1 = \frac{2n}{k}, \quad Y 1 = 1, \quad AY \leq 1$$

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 $j$th column, we set $Y_{ij} = 1$, $Y_{ij} = 0$ (if $i > t$, then $i = i - n$, otherwise $i = i + n$), set all of the other entries on the $i$th row to 0, and remove this row from further consideration. If there are already $2n/k$ 1s in the $j$th 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 $g_1 = Q_4, Q_5, Q_7$, $g_2 = Q_4, Q_5, Q_8, g_3 = Q_6, Q_7, Q_8$, 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 $C_j$.

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 $C_1$ 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 $C_j$ should also be recorded. Then, a random element in $Friend_j$, with $C_j \leq C_l$ for $1 \leq j, l \leq k$, will be grouped into $g_j$. Note that a query, $Q_i$, may be the friendliest neighbor of $1 < m \leq k$ 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.

**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
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 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

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
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


