A Client-biased Cooperative Search Scheme in Blockchain-based Data Markets

Suhan Jiang, Yubin Duan, Jie Wu
Department of Computer and Information Sciences, Temple University
{Suhan.Jiang, yubin.duan, jiewu}@temple.edu

***Abstract*—Lots of privacy and security issues in the current cloud-based data markets will be eliminated by taking advantage of blockchain-based decentralized storage services, which can provide a new paradigm for safe data outsourcing and correct remote search. However, existing data markets are also questioned on their inflexible and opaque pricing, where the value of data ownership and the cost of query search are mixed. Thus, a better pricing model is necessarily needed in an emerging decentralized data market. In this paper, we envision an Ethereum-based data market, in which the pricing model for each query includes two parties: owner (paid for his data ownership) and miner (rewarded by query search). We study a new cooperative search scheme through a proxy to reduce cost on the client (user) side. Suppose each user query is charged based on the number of keywords in the query. The cost reduction is based on combining multiple queries into a group subject to the constraint that the resulting combined query is not significantly larger than any of its original query in terms of the number of keywords. The total price is based on total number of keywords in all groups. As the optimal grouping depends on the pricing of both owner and miner, we build a small testbed to analyze how price setting will affect grouping results. Since it is a cooperative model with shared resources, we also study various incentive properties on the client side, thereby yielding a cost sharing mechanism to split joint cost in a truth-revealing and fair manner.**

***Index Terms*—Blockchain, cooperative search, cost model, cost sharing, grouping strategy.**

I. Introduction

Data has become a tradeable good in our society nowadays. Most online data markets, *e.g*. Amazon Athena and Xignite, are cloud-based. Cloud offers a convenient single platform for trading data, and provides value-added services that help derive data products. However, centralized clouds also give rise to privacy and security issues in data storage and search. Thus, works on decentralized storage services have been proposed to alleviate these concerns. Blockchain techniques [1] are widely used to guarantee data integrity and provide scalability for handling big data. In terms of remote search, smart contract [2] is leveraged to protect the correctness and immutability of search results. Unfortunately, lots of existing works focus on the single-user setting, where a database is queried only through its owner. As in real data markets, a data owner makes profit by sharing his data with legitimate clients. Thus, a more complex multi-user setting should be discussed in the decentralized storage.

In this paper, we envision an Ethereum-based data market that provides services related to data storage, search and trades. Since Ethereum [3] is a blockchain-based decentralized computing platform [2] merged with smart contract, it guarantees reliable data storage and correct remote search. As is shown in Fig. 1, this new type of data market has three basic entities. As a data provider, an *owner* profitably shares his database by allowing third-party called *users* to query his database. Unlike any cloud-based data market, where private data is uploaded to the cloud and in the central control of the cloud provider, an Ethereum-based data market allows databases to be managed in a decentralized fashion. That is, an owner can either send his small-size information to the Ethereum blockchain [2] for the convenience of searching, or distribute his large data in an off-blockchain storage [4], *e.g*. IPFS [5], with a pointer to the data on the distributed ledger of blockchain. We assume all data is encrypted under a searchable symmetric encryption (SSE) scheme before being outsourced to keep it confidential while still allowing query-based searches. Users are data consumers and are willing to retrieve information from a designated database with some payment. We assume that users directly send queries to a corresponding owner, and the owner issues search transactions as if he is querying. As data searchers, *miners* in Ethereum make money by executing search functions in smart contracts.

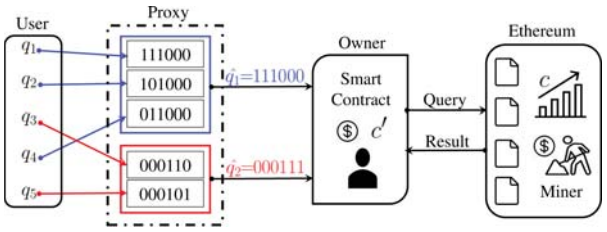


Fig. 1: Given a database with six keywords, five queries q_1, q_2, q_3, q_4, q_5, each being a six-bit binary string with 1 for search and 0 for not search, are issued from users to retrieve information.

Thus, query pricing in such a decentralized data market is unilaterally decided by both owners and miners. Each user query is charged for two parts: one for the data owner at a pre-set price based on data value, and the other for miners based on their workload, which is a pay-as-you-go mode. Thanks to Ethereum's gas system, a miner's computation consumption is traceable and transparent in Ethereum. Besides, a query's computation consumption and the corresponding search delay, as well as its cost, tend to be proportional to its keyword number. Since all users want to query at a cheap price, *cooperative search* can be a good approach to save total cost, hence decreasing individual payments. To illustrate, let us

978-1-7281-1856-7/19/$31.00 ©2019 IEEE

consider an example in Fig. 1. Five users want to search over the same database to retrieve information with six keywords. Given per-query cost c' from the owner and per-keyword cost c from miners, each query will be separately charged by the owner and miners. For example, the cost of query $q_1 = 111000$ (a six-bit binary string for six keywords, with 1 for select and 0 for not select) is c' for the owner plus $3c$ due to 3-keyword search in Ethereum. Without cooperation, the overall cost of these queries is $5c' + 11c$ (5 users and 11 accumulative keywords). To save cost, their queries can be grouped. Among all grouping strategies, we briefly mention two here: (1) a *cost-saving-based* strategy without considering delay constraints. Five users are grouped together, and their combined query is 111111 at a total cost of $c' + 6c$. Search latency largely increases for each client. (2) a *delay-tolerant-oriented* strategy (as is shown in Fig. 1) that groups q_1, q_2, q_4 in G_1, and q_3, q_5 in G_2. Thus, the total cost for all users is $2c' + 6c$ and any user at most waits additional 1-keyword search time. This example reflects a tradeoff between delay constraint and cost efficiency.

We consider users with limited delay-tolerance. We design a client-biased cooperative search scheme, which facilitates group formation among users driven by cost savings subject to a uniform delay constraint all users agree upon. Users submit their individual queries and delay constraints to a front *proxy*. The proxy gathers users with similar delay constraints into a set, and matches users from the same set as search groups in order to minimize overall cost subject to the delay constraint (*i.e.*, the number of 1s in each group query should not exceed a given number). After grouping, the data owner receives combined queries from the proxy, and issues search requests to Ethereum. This cooperative search scheme improves a data owner's processing capacity by reducing the query number.

In a cooperative model, cost sharing must be regulated in a truth-revealing and fair manner. Truth-revealing helps avoid free-riding users who want to get some information without payment, and fairness can promote a stable and long-term cooperation among users. For example, we consider a common cost sharing mechanism, which equally distributes group cost among its members. After applying it to the grouping result in Fig. 1, q_3 and q_5 will be equally charged with $(c' + 3c)/2$. It seems unfair because q_5 has fewer keywords in its original form compared with q_3. In our paper, we design a keyword-based cost sharing mechanism according to original queries, which yields some desirable properties like group-strategyproofness and sharing incentive. In Fig. 1, the cost $2c'$ paid to the owner will be equally distributed among 5 users, each of whom is responsible for $2c'/5$. The total cost of searching for the first keyword is c, equally shared by q_1 and q_2. The last keyword also costs c, which is only borne by q_5, since there is no other user querying this keyword.

Specifically, given a set of n queries over a database, our objective is to classify n queries into k groups with each group satisfying a uniform delay constraint so that the total cost over all groups is minimized. The contributions of this paper are summarized as follows:

- Extending from the single-user setting in decentralized storage, we propose an Ethereum-based data market.
- A client-biased, limited-delay-tolerant cooperative search scheme driven by cost savings is designed to maximize social welfare on the user side.
- We formulate an n-query-grouping problem as a set partition problem, prove it as NP-hard, and we solve it through approximation with guaranteed bounds, as well as an efficient projected gradient descent method.
- A cost sharing mechanism is provided to fairly split total payment among all participating users. This mechanism guarantees several desirable properties, *e.g.* group-strategyproofness and sharing incentive.
- Extensive evaluations on real query traces AOL demonstrate the effectiveness of our cooperative scheme. A small testbed of Ethereum-based data market is also implemented to show the relationship between the number of keywords and the search delay by the miners.

II. Related Work

1) Blockchain-based storage platforms: Decentralized storage platforms [4, 6–9] are designed to allow users to store outsourced files on the peers who make profits by leasing their unused storage. Data can be stored either on-blockchain [6–8] or off-blockchain [4]. Filecoin [6] implements blockchain structured file storage and Datacoin [7] uses the blockchain as a data store for file blocks. Storj [8] employs end-to-end encryption and stores files on the blockchain, while [4] stores files in an off-blockchain storage, retaining only a pointer to the data on the public ledger.

2) Online data markets: On traditional online data markets, independent data owners sell their data through some digital platforms [10, 11]. Usually, buyers (users) have to buy the entire database from a seller, then download and query it offline, which is inefficient for buyers. On cloud-based data markets [12, 13], data owners upload their data in the cloud and make money by sharing it with users. Users pay to obtain what they want through a query interface instead of buying the entire database. Cloud providers are rewarded by providing services, *e.g.* storage and search. Our Ethereum-based data market is similar to cloud-based data markets, where users pay for querying, while owners and miners earn by providing data and services, respectively.

3) Cost models: Traditional online markets take a one-time payment set by data owners. On cloud-based data markets, either data owners [12] or cloud providers [13] can be sellers, with term-based offers such as monthly subscriptions. Some cloud providers use a pay-as-you-go mode. Each query is charged based on bytes of scanned data [14]. On the Ethereum-based data market, owners and miners are all sellers, jointly charging users. Previously-mentioned online data markets all adopt a unilateral cost model [15], while the cost model can also be bilateral, such as auction, in a more complex situation.

4) Cooperative search: The cooperative keyword-based search scheme has been proposed in [16] by grouping all received queries together within a fixed timeout to achieve privacy. In [17], authors equally divide queries into a fixed

number of groups to achieve k-anonymity and load balancing. Our work also deals with query grouping problems, while focusing on user-side cost saving and search delay guarantee.

5) Cost sharing schemes: A good cost sharing mechanism will distribute shared cost among users in a truth-revealing and fair manner [18]. Current cost sharing mechanism is group-based [17]. Our proposed mechanism is keyword-based, which guarantees group-strategyproofness and sharing incentive.

III. Preliminaries

1) Scheme Overview: Our multi-user cooperative search scheme consists of two stages, involving four entities, as is shown in Fig. 1. The first stage consists of three entities: users, a proxy and a data owner. Although we assume a data owner and a proxy, it can be easily extended to multiple owners and proxies. All users send their queries along with delay constraints to the proxy. The proxy gathers users with similar delay constraints into same sets. We assume each set is sufficiently large, thus treat them separately. In each set, the minimal value of all users' delay constraints is set as the set delay upper bound. Given an n-query set, the main function of the proxy is to run our grouping strategies which classify n queries into k groups (k is a variable) and send those combined queries to the data owner. In the second stage, a data owner issues search transactions to Ethereum, based on queries received from his proxy. Then miners execute related functions to obtain search results. The total cost is shared among n users using our cost sharing mechanism.

2) Design Goals: To implement a client-biased cooperative search scheme in the Ethereum-based data market, our main goals are divided into three parts. First, we need to design effective grouping strategies by simultaneously achieving *social optimum*: the total cost among n users is minimal, and *latency limitation*: each user can be guaranteed to retrieve desired information within their uniformly-agreed delay tolerance. Second, we want to design a cost sharing mechanism among n users, which offers *group-strategyproofness*: each user should truthfully reveal individual query request even if collusion is permitted, since lying provides no benefit to his interest, and *sharing incentive*: each user should achieve individual cost reduction if their total cost gets reduced. Third, we would like to implement an Ethereum testbed to verify the practicality of our proposed search schemem, and the actual relationship between the keyword number and the search delay.

IV. Grouping strategy

A. Notation and Cost Model

Assuming we have a set of n queries, $Q = \{q_1, q_2, \cdots, q_n\}$, that are issued from different users but over the same database. Corresponding notations are listed in Table I.

The cost $C(q)$ of a query q consists of two parts: c' is charged by a corresponding owner due to his contribution on data and $c \cdot |q|$ is paid to a miner in search of information.

TABLE I: Summary of Notations.

Symbol	Description
d	number of keywords in the dictionary
w_i	the i-th keyword in the dictionary
n	number of queries
Q	n-query set
G_i	group i, which is a subset of Q
$\|G_i\|$	number of queries in G_i
q_i	query from user i, in the form of a binary string
$\|q_i\|$	number of keywords in q_i
$\hat{q}_i$	combined query of G_i
$\|\hat{q}_i\|$	number of keywords in $\hat{q}_i$
k	number of groups
P_i	a partition over Q with i non-overlapping groups
c	cost of miner searching for a keyword
c'	cost of a data owner

1) Cost without Grouping: The n queries in the set Q are individually executed and their total cost is the accumulation of their individual costs.

$$\sum_{i=1}^{n} C(q_i) = c' \cdot n + c \cdot \sum_{i=1}^{n} |q_i| \tag{1}$$

2) Cost with Grouping: Another way to execute Q is to create a single group G that contains all n queries and execute it as a single query $\hat{q} = |q_1 \vee \cdots \vee q_n|$.

$$C(\hat{q}) = c' + c \cdot |\hat{q}| \tag{2}$$

We can determine if grouping is beneficial by comparing Eq. (1) with Eq. (2).

$$\sum_{i=1}^{n} C(q_i) - C(\hat{q}) = c'(n-1) + c(\sum_{i=1}^{n} |q_i| - |\hat{q}|) \tag{3}$$

Even in the worst case, where no overlapping keywords are among n queries, *i.e.*, $\sum_{i=1}^{n} |q_i| = |\hat{q}|$, Eq. (3) ≥ 0 still holds. It is obvious that grouping is always beneficial. Thus, greedily grouping all queries into a combined query is always a socially optimal choice, if no user has a constraint on search delay.

B. Problem Formulation

Given $Q = \{q_1, q_2, ..., q_n\}$, let α be the pre-agreed maximal search delay, measured in the numbers of keywords among n users. That is, each user is willing to wait for at most α-keyword search time, whatever their original keyword numbers before grouping. An optimal grouping problem is defined as:

Problem 1 (Optimal Query Grouping, OQG)**.** *Given a set of queries $Q = \{q_1, q_2, ..., q_n\}$, group the n queries into k non-overlapping groups $G_1, G_2, ..., G_k$ (k is a variable), such that the number of keywords in each combined query is no more than α and the overall cost of all groups is minimized.*

C. Problem Hardness

Theorem 1. *The OQG problem is NP-hard.*

Proof. The Set Partitioning (SP) problem , known as NP-hard, can be reduced to the OQG problem. The SP problem is expressed as: given a set of n positive integers $\{a_1, a_2, \cdots, a_n\}$ and an integer A, where $\forall i \in [1, n],\ a_i \leq A$, such that $\sum_{i=1}^{n} a_i = 2A$, decide if this set can be partitioned into two subsets with the same sum A. We can construct every instance of the SP problem as a valid instance of the OQG problem as follows. Let α, the upper limit of keyword number in each

combined query, be equal to A, and d, the number of keywords in the dictionary, be $2A$. Each a_i is mapped to a query q_i. We define the number of keywords for each query $|q_i|$ as a_i. We also assume that there is no overlapping among all queries. This is a valid instance of the OQG problem.

An optimal solution to the OQG problem with two groups exists, if and only if there exists a partition in the original SP problem. Obviously, merging n queries into a single group G is infeasible, because the combined query $\hat{q}$ contains $2A$ keywords, exceeding the upper limit. Hence, at least one query should be removed from G. According to our previous analysis, for any group G_i in the optimal solution, its cost is $C(\hat{q}_i) = c' + c \cdot |\hat{q}_i|$. We minimize $C(\hat{q}_i)$ over all k groups, but $c\sum_{i=1}^{k} |\hat{q}_i|$ is constant among all grouping strategies, hence our problem is equivalent to minimizing the number of groups.

If the OQG problem has an optimal solution with two groups G_1 and G_2, then there exists a partition of n queries into two sets, such that $\sum_{q_i \in G_1} |q_i| = \sum_{q_j \in G_2} |q_j| = \alpha$. This partition is definitely an optimal solution to SP problem. On the other hand, if the original SP problem has an equal-sum partition, the OQG problem also has an optimal strategy with two groups, since three groups yield more cost. Now, we can conclude that the OQG problem is NP-hard. □

D. Grouping Strategy

Since the OQG problem is NP-hard, we solve it using linear programming relaxation and greedy algorithms.

1) Mathematic Relaxation: First, we give the matrix representation of the above problem. Since each user has a query string with length d, we define an $n \times d$ matrix $\mathbf{Q}$ as representing all queries from n users. Let $\mathbf{Y} \in \{0,1\}^{n \times n}$ denote the grouping result, then each (i,j)-th element of $\mathbf{Y}$ takes either 0 or 1. $Y_{ij} = 1$ means query q_i is classified into group G_j. The matrix representation is shown below.

$$\arg\min_{\mathbf{Y}} \quad c' \cdot tr(\delta(\mathbf{Y}^T\mathbf{E})) + c \cdot tr(\mathbf{E}^T\delta(\mathbf{Y}^T\mathbf{Q})) \tag{4a}$$

$$\mathbf{Y} \in \{0,1\}^{n \times n},\ \delta(\mathbf{Y}^T\mathbf{Q})\mathbf{e} \leq \alpha\mathbf{e},\ \mathbf{Y}\mathbf{e} = \mathbf{e} \tag{4b}$$

where tr is the trace operator which sums up diagonal elements of a given matrix. $\mathbf{E}$ and $\mathbf{e}$ denote an all-1's matrix and an all-1's vector, respectively, and both of their size can be adjusted to fit the context. And $\delta(\cdot)$ is an indicator function such that $\delta(c) = 1$ if c is non-zero and $\delta(c) = 0$, otherwise. The value of (i,j)-th element in matrix $\mathbf{Y}^T\mathbf{Q}$ represents how many times the keyword w_j is queried among all members in the group G_i. Since each overlapping keyword in a group only needs querying once, $\delta(\mathbf{Y}^T\mathbf{Q})$ indicates combined queries for all groups. Thus, $tr(\mathbf{E}^T\delta(\mathbf{Y}^T\mathbf{Q}))$ calculates the total keyword number of all combined queries. Similarly, $tr(\delta(\mathbf{Y}^T\mathbf{E}))$ indicates the true group number.

We use the example in Fig. 1 to better explain each term in Eq. (4a). Five queries q_1, q_2, q_3, q_4, q_5 are represented as a matrix $\mathbf{Q}$ in Fig. 2(a), where the i-th row represents q_i. The grouping result $\mathbf{Y}$ is provided in Fig. 2(b), indicating there are two groups, q_1, q_2 and q_4 in G_1, and q_3, q_5 in G_2. Thus, the number of groups can be calculated using the expression $tr(\delta(\mathbf{Y}^T\mathbf{E})) = 2$ based on Fig. 2(c). Besides, the combined

$$\mathbf{Q}_{5\times6}=\begin{bmatrix}1&1&1&0&0&0\\1&0&1&0&0&0\\0&1&1&0&0&0\\0&0&0&1&1&0\\0&0&0&1&0&1\end{bmatrix} \qquad \mathbf{Y}_{5\times5}=\begin{bmatrix}1&0&0&0&0\\1&0&0&0&0\\0&1&0&0&0\\1&0&0&0&0\\0&1&0&0&0\end{bmatrix}$$

(a) (b)

$$\delta(\mathbf{Y}^T\mathbf{E})=\delta\left(\begin{bmatrix}1&1&0&1&0\\0&0&1&0&1\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}\times\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}\right)=\begin{bmatrix}1&&&&\\&1&&&\\&&0&&\\&&&0&\\&&&&0\end{bmatrix}$$

(c)

$$E^T\delta(\mathbf{Y}^T\mathbf{Q})=\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}\times\begin{bmatrix}1&1&1&0&0&0\\0&0&0&1&1&1\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}=\begin{bmatrix}1&&&&&\\&1&&&&\\&&1&&&\\&&&1&&\\&&&&1&\end{bmatrix}$$

(d)

Fig. 2: Matrix representation of queries and grouping result (We don't fill all elements in some matrices for saving space).

queries of G_1 and G_2 are $\hat{q}_1 = 111000$ and $\hat{q}_2 = 000111$, which are consistent with its matrix representation $\delta(\mathbf{Y}^T\mathbf{Q})$ shown in Fig. 2(d). Hence, the total keyword number over all combined queries $tr(\mathbf{E}^T\delta(\mathbf{Y}^T\mathbf{Q})) = 6$ is equal to $|\hat{q}_1| + |\hat{q}_2|$.

Our objective is to find a grouping strategy $\mathbf{Y}$, which will result in a minimal overall cost for n queries, as Eq. (2) shows, while each combined query has no more than α keywords ($\delta(\mathbf{Y}^T\mathbf{Q})\mathbf{e} \leq \alpha\mathbf{e}$) and any original query only belongs to a group ($\mathbf{Y}\mathbf{e} = \mathbf{e}$). Since it is an NP-hard problem, we consider a relaxation. According to [16], given a large constant number β, *e.g.* $\beta = 10,\ 20,\ 30$, the indicator function over the matrix $\mathbf{Y}^T\mathbf{Q}$, *i.e.*, $\delta(\mathbf{Y}^T\mathbf{Q})$, can be approximated with a smooth function, $\mathbf{E} - e^{(-\beta\mathbf{Y}^T\mathbf{Q})}$, and the indicator function over the matrix $\mathbf{Y}^T\mathbf{E}$, *i.e.*, $\delta(\mathbf{Y}^T\mathbf{E})$, can be approximated with a smooth function, $\mathbf{E} - e^{(-\beta\mathbf{Y}^T\mathbf{E})}$. With the above relaxations, we transfer the integer matrix $\mathbf{Y}$ into the continuous domain. Then, we get a relaxed version of the OQG problem below:

$$\arg\min_{\mathbf{Y}} \quad c' \cdot tr(\mathbf{E} - e^{-\beta\mathbf{Y}^T\mathbf{E}}) + c \cdot tr(\mathbf{E}^T[\mathbf{E} - e^{-\beta\mathbf{Y}^T\mathbf{Q}}])$$

$$\mathbf{Y} \in [0,1]^{n \times n},\ \mathbf{Y}\mathbf{e} = \mathbf{e},\ e^{-\beta\mathbf{Y}^T\mathbf{Q}}\mathbf{e} \geq (d - \alpha)\mathbf{e}$$

which equals to its dual problem as below:

$$\arg\max_{\mathbf{Y}} \quad c' \cdot tr(e^{-\beta\mathbf{Y}^T\mathbf{E}}) + c \cdot tr(\mathbf{E}^T e^{-\beta\mathbf{Y}^T\mathbf{Q}}) \tag{5}$$

$$\mathbf{Y} \in [0,1]^{n \times n},\ \mathbf{Y}\mathbf{e} = \mathbf{e},\ e^{-\beta\mathbf{Y}^T\mathbf{Q}}\mathbf{e} \geq (d - \alpha)\mathbf{e}$$

The objective of Eq. (5) is to maximize a convex function over multiple variables with nonlinear constraints. We apply the interior-point approach, transforming the original inequality-constrained problem into a sequence of equality constrained problems. A logarithmic barrier function with a dynamic coefficient μ is constructed and added to the original objective function to remove all inequality constraints. This algorithm has a two-level iteration. The outer level iterates over the coefficient μ, while the inner level optimizes the augmented objective function using Newton method under a fixed μ. Since Newton method may lead to a local optima, we can run the algorithm with different initial values and select the best one.

In addition, we also design a rounding algorithm to obtain a

TABLE II: Examples of 7 User Queries.

Queries	Content
q_1	11010000
q_2	00001101
q_3	11000000
q_4	00000111
q_5	00001100
q_6	00000011
q_7	10000000

(a) Example One.

Queries	Content
q_1	11010000
q_2	00001101
q_3	11000000
q_4	00000111
q_5	00001100
q_6	10000001
q_7	00110000

(b) Example Two.

feasible 0–1 solution based on the continuous optima achieved above. In each iteration, the rounding algorithm greedily sets Y_{ij} as 1 to maximize the objective function in Eq. (5) while still satisfying all constraints. To make our integer solution closer to the optimum one, the rounding order of queries matters. We always start with queries with the most keywords first, because chances are higher for them to overlap with other queries. Once their grouping result is determined, other queries can be assigned to corresponding groups.

We consider a dictionary that consists of (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8) and two sample queries are as shown in Table II(a) and II(b). We also assume $c = c'$. To show how it affects grouping results, the constraint of search delay will be set differently. Table III shows grouping results under different delay constraints using our Mathematic Relaxation.

2) Projected Gradient Descent Method: The Newton method is quite simple and gives a relatively fast rate of convergence. However, this method is very expensive in each iteration - it needs the function evaluation and then the derivative evaluation. If the volume of queries is large, then this might not be a good choice. Thus, to improve the efficiency when facing large amounts of queries, we consider a simple modification of gradient descent for constrained optimization: a projected gradient descent method. In general, projected gradient algorithms minimize an objective $f(x)$ subject to the constraint that $x \in \chi$ for some convex set χ. They do this by iteratively updating $x := \prod_\chi(x + \eta\nabla f(x))$, where η represents a step length of learning rate, and $\prod_\chi = \arg\max_x \{\|z - x\| \, | x \in \chi\}$ is the Euclidean projection onto set χ. First order projected gradient algorithms are effective when second order methods are infeasible because of the dimension of the problem.

3) Greedy Algorithm: Since Mathematic Relaxation uses a first order local optimization method to solve a non-convex optimization problem, of which computational complexity isn't proven, this strategy has no guarantee on time. Thus, we consider a greedy algorithm with a guaranteed bound.

According to section IV-A, combination always brings about cost saving. Here, we reconsider the OQG problem in terms of cost saving. Given a set of queries $Q = \{q_1, q_2, ..., q_n\}$ from n different users over the same database, group these n queries into k non-overlapping groups $G_1, G_2, ..., G_k$ where the number of keywords in each group is no more than α such that the overall sum of savings $\sum_{i=1}^{k} S(G_i)$, is maximized.

Considering two queries, q and q', we define the saving of merging q with q' as $c' + c(|q| + |q'| - |q \vee q'|)$. In other words, combining two queries, q and q' will save one charge from the owner and overlapping-keyword search cost. Since one token

TABLE III: Grouping Results using Mathematic Relaxation.

Constraint	Group	Combined Query
4	$G_1 = \{q_1, q_3, q_7\}$	$\hat{q}_1 = 11010000$
	$G_2 = \{q_2, q_4, q_5, q_6\}$	$\hat{q}_2 = 00001111$
3	$G_1 = \{q_1, q_3, q_7\}$	$\hat{q}_1 = 11010000$
	$G_2 = \{q_2, q_5\}$	$\hat{q}_2 = 00001101$
	$G_3 = \{q_4, q_6\}$	$\hat{q}_3 = 00001111$

(a) Example One.

Constraint	Group	Combined Query
5	$G_1 = \{q_1, q_3, q_6, q_7\}$	$\hat{q}_1 = 11010001$
	$G_2 = \{q_2, q_4, q_5\}$	$\hat{q}_2 = 00001111$
4	$G_1 = \{q_1, q_3, q_7\}$	$\hat{q}_1 = 11010000$
	$G_2 = \{q_2, q_4, q_5\}$	$\hat{q}_2 = 00001111$
	$G_3 = \{q_6\}$	$\hat{q}_3 = 10000001$

(b) Example Two.

generation cost can be saved by combining any two queries q and q', the more overlapping keywords q and q' have, the more search cost it will save through their combination. If q and q' have a containment relationship, we can prove there always exists some optimal grouping result that contains a group with both q and q', as is shown in Theorem 2.

Theorem 2. *Given a set of queries $Q = \{q_1, q_2, ..., q_n\}$, for any two queries q and q', if the keywords of q are entirely contained in the keywords of q', then there is some optimal grouping strategy that contains a group with both q and q'.*

Proof. Assume the optimal partition P over Q has two groups G_1 and G_2, such that $q \in G_1$ and $q' \in G_2$. Thus, individual group saving of G_1 is $S(G_1) = c'(|G_1| - 1) + c(\sum_{i=1}^{|G_1|} |q_k| - |\hat{q}_1|)$, and the same for G_2. We can obtain the overall savings of G_1 and G_2 :

$$S(G_1) + S(G_2) = \tag{6}$$
$$c'(|G_1| + |G_2| - 2) + c(\sum_{i=1}^{|G_1|+|G_2|} |q_k| - |\hat{q}_1| - |\hat{q}_2|)$$

Moving q to G_2 leads to new groups $G_{1'} = G_1 \setminus \{q\}$ and $G_{2'} = G_2 \cup \{q\}$, with $|G_{1'}| = |G_1| - 1$ and $|G_{2'}| = |G_2| + 1$. q is entirely contained by q', hence $|\hat{q}_{2'}| = |\hat{q}_2|$. Thus, $S(G_{2'}) = c'(|G_2|) + c(\sum_{i=1}^{|G_2|} |q_i| + |q| - |\hat{q}_2|)$. Although it is difficult to directly know the exact value of $|\hat{q}_{1'}|$, we can bound it as $|\hat{q}_1| - |q| \le |\hat{q}_{1'}| \le |\hat{q}_1|$, such that $S(G_{1'}) \ge c'(|G_1| - 2) + c(\sum_{i=1}^{|G_1|} |q_i| - |q| - |\hat{q}_1|)$. Listed below is a lower bound of the overall group savings for new groups $G_{1'}$ and $G_{2'}$:

$$S(G_{1'}) + S(G_{2'}) \ge \tag{7}$$
$$c'(|G_1| + |G_2| - 2) + c(\sum_{i=1}^{|G_1|+|G_2|} |q_i| - |\hat{q}_1| - |\hat{q}_2|)$$

Comparing Eq. (6) and Eq. (7), we conclude that, moving q to G_2 yields a new partition, which is at least as good as P and thus still optimal. □

Hence, we can greedily group queries q and q', and treat them cost-wise as a single query q', which can be further merged with other queries. The containment can easily be determined by OR operation. This is a Naive Greedy solution, of which the time complexity is $O(n^2)$. For the rest of the paper, we assume all such containments have been identified.

It is obvious that, without any constraints on the search delay, the optimal solution is to combine n users as a group,

Algorithm 1 Greedy Partition

Input: a system, (Q, C)
Output: a partition over Q, P

function GREEDY-PARTITION(Q, C)
  $P_1 \leftarrow \{Q\}$
  for each $i \in [1, n]$ **do**
    for all $W \in P_i$ **do**
      $(S, W) \leftarrow$OPTIMAL-SUBSET(W)
    $(S_i, W_i) \leftarrow argmin\,(C(S) + C(W/S) - C(W))$
    $P_{i+1} \leftarrow (P_i - \{W_i\}) \cup \{S_i, W_i/S_i\}$
    if each $W \in P_i$ satisfies search delay **then**
      return P_i

and issue a single combined query. However, when constraints are added, it becomes an NP-hard problem, thereby we consider a Greedy Partition solution to efficiently approximate its optimal result with an upper bound. Greedy Partition starts with a single group, iterating to split a group of which the splitting cost is minimal among all existing groups, until all query groups are subject to the search delay constraints. As it is a typical set partition problem, we will consider Problem 1 in terms of set theory as follows.

First, given a query set Q, we define a set function $C : 2^Q \rightarrow \mathbb{R}$ where

$$\forall\, G \subseteq Q,\ C(G) = \begin{cases} 0 & G = \emptyset \\ c' + c \cdot |\hat{q}| & \text{otherwise} \end{cases} \tag{8}$$

Then, we can reformulate the OQG problem as below. Given a system (Q, C, k), where Q is a set of queries, $C : 2^Q \rightarrow \mathbb{R}$ is a set function, and k is a variable with $1 \leq k \leq n$.

$$\begin{aligned} \text{minimize} \quad & C(G_1) + C(G_2) + \cdots + C(G_k) && \text{(9a)} \\ \text{subject to} \quad & G_1 \cup G_2 \cup \cdots \cup G_k = Q && \text{(9b)} \\ & G_i \cap G_j = \emptyset \qquad 1 \leq i < j \leq k && \text{(9c)} \\ & |\hat{q}_i| \leq \alpha \qquad 1 \leq i \leq k && \text{(9d)} \end{aligned}$$

As is proven in the paper [19], given a nondecreasing submodular system (V, f, k), where $f(V) + f(\emptyset) \geq f(S)$ holds for any nonempty subset S of V, the set partition problem can be approximated within a factor of $(2 - 2/k)$ in polynomial time. They provide a greedy algorithm to guard this result. Since our system is also submodular (proven below), we present Greedy Partition which satisfies delay constraints.

As is shown in Algorithm 1, Greedy-Partition has two functions. The main function GREEDY-PARTITION returns the final partition P over a given set Q. Starting with the 1-partition $P_1 = Q$, in its ith iteration, we obtain an $i+1$-partition P_{i+1} by partitioning some members of the previous i-partition P_i. We halt when a partition P satisfies the delay constraints. For any member W in i-partition P_i, we call function OPTIMAL-SUBSET [19] to find its minimal-partitioning-cost subset. Since a minimal-cost solution is desired, we choose the least-cost partition among all members. The time complexity of this algorithm is $O(kn^3)$, where k is the number of groups, and n is the query number. Grouping results of previous examples are listed in Table IV.

In the rest of this section, we will demonstrate that Greedy

TABLE IV: Grouping Results using Greedy Partition.

Constraint	Group	Combined Query
4	$G_1 = \{q_1, q_3, q_7\}$	$\hat{q}_1 = 11010000$
	$G_2 = \{q_2, q_4, q_5, q_6\}$	$\hat{q}_2 = 00001111$
3	$G_1 = \{q_1, q_3, q_7\}$	$\hat{q}_1 = 11010000$
	$G_2 = \{q_2, q_5\}$	$\hat{q}_2 = 00001101$
	$G_3 = \{q_4, q_6\}$	$\hat{q}_3 = 00001111$

(a) Example One.

Constraint	Group	Combined Query
5	$G_1 = \{q_1, q_3, q_7\}$	$\hat{q}_1 = 11010000$
	$G_2 = \{q_2, q_4, q_5, q_6\}$	$\hat{q}_2 = 10001111$
4	$G_1 = \{q_1, q_3, q_7\}$	$\hat{q}_1 = 11010000$
	$G_2 = \{q_2, q_4, q_5\}$	$\hat{q}_2 = 00001111$
	$G_3 = \{q_6\}$	$\hat{q}_3 = 10000001$

(b) Example Two.

Partition can solve the OQG problem within a factor of $(2-2/k)$ in polynomial time. According to [19], our proposed algorithm on a given system (Q, C) can achieve above properties if the following two conditions hold: (1) C is submodular, and (2) C is non-decreasing.

Before showing Greedy Partition satisfies the above two conditions, we introduce their definitions. Given a finite set V and a set function $f : 2^V \rightarrow \mathbb{R}$, f is (1) submodular if $\forall S \subseteq V$ and $s_1, s_2 \in V \setminus S$, $f(S \cup \{s_1\}) + f(S \cup \{s_2\}) \geq f(S \cup \{s_1, s_2\}) + f(S)$ always holds; (2) non-decreasing if $f(V) + f(\emptyset) \geq f(S)$ holds for any nonempty subset S of V .

Lemma 1. *The set function $C : 2^Q \rightarrow \mathbb{R}$ is submodular.*

Proof. Now, we will show for every $G \subseteq Q$ and $q_1, q_2 \in Q \setminus G$, Eq. (10) always holds. Without loss of generality we can assume that $G \neq \emptyset$, otherwise the answer is immediate.

$$C(G \cup \{q_1\}) + C(G \cup \{q_2\}) \geq C(G \cup \{q_1, q_2\}) + C(G) \tag{10}$$

Since $C(G \cup \{q_1\}) + C(G \cup \{q_2\}) = 2c' + c(|\hat{q} \vee q_1| + |\hat{q} \vee q_2|)$ and $C(G \cup \{q_1, q_2\}) + C(G) = 2c' + c(|\hat{q} \vee q_1 \vee q_2| + |\hat{q}|)$, we need to prove Eq. (11) ≥ 0 always holds.

$$\begin{aligned} & C(G \cup \{q_1\}) + C(G \cup \{q_2\}) - C(G \cup \{q_1, q_2\}) - C(G) \\ & = c(|\hat{q} \vee q_1| + |\hat{q} \vee q_2|) - c(|\hat{q} \vee q_1 \vee q_2| + |\hat{q}|) \\ & = c(|\hat{q} \vee q_1| + |\hat{q} \vee q_2| - |\hat{q} \vee q_1 \vee q_2| - |\hat{q}|) \end{aligned} \tag{11}$$

Assume that, nonnegative integers x, y, z represent the number of overlapping keywords between $\hat{q}$ and q_1, $\hat{q}$ and q_2, q_1 and q_2, respectively. Let m be the overlapping keyword number among $\hat{q}$, q_1, and q_2. It is obvious that $z \geq m \geq 0$.

$$\begin{aligned} & |\hat{q} \vee q_1| + |\hat{q} \vee q_2| = 2\,|\hat{q}| + |q_1| + |q_2| - x - y \\ & |\hat{q} \vee q_1 \vee q_2| + |\hat{q}| = 2\,|\hat{q}| + |q_1| + |q_2| - x - y - z + m \\ & |\hat{q} \vee q_1| + |\hat{q} \vee q_2| - |\hat{q} \vee q_1 \vee q_2| - |\hat{q}| = z - m \geq 0 \end{aligned} \tag{12}$$

Based on Eq. (12), we conclude, for every $G \subseteq Q$ and $q_1, q_2 \in Q \backslash G$, Eq. (10) always holds. Thus, $C : 2^Q \rightarrow \mathbb{R}$ is a submodular set function. □

Lemma 2. *The set function $C : 2^Q \rightarrow \mathbb{R}$ is non-decreasing.*

Proof. Based on Definition 2, we should prove $C : 2^Q \rightarrow \mathbb{R}$, $C(Q) + C(\emptyset) \geq C(G)$ holds for any nonempty subset G of Q. According to Eq. (8), $C(\emptyset) = 0$. Since $\forall G \subseteq Q$, it is obvious that set Q's combined query contains more keywords than

its subset G's combined query. Thus, we can obtain $C(Q) - C(G) \geq 0$, hence, $C(Q) + C(\emptyset) \geq C(G)$. □

Theorem 3. *The QOG problem can be approximated within a factor of* $(2 - 2/k)$ *by Greedy Partition.*

This theorem easily follows from Lemmas 1 and 2. This grouping strategy is suitable for those users who have requirements on cost reductions.

V. Fair Cost Sharing

Our grouping strategies will yield a total cost for n users. Thus, one must find a way to distribute the cost among all users. A major purpose of our proposed grouping strategies is to seek high efficiency of the whole network, in the fields of both finance and computation. As self-interested and autonomous entities, clients may behave strategically by misreporting their willingness to query to maximize their profit, thereby harming the efficiency. Thus, we want our cost sharing mechanism to be incentive compatible, i.e., it is in the interest of clients to be truth telling [19]. Also, it should provide incentive for clients in their assigned groups to participate in the coalition without coercion, *i.e.*, it is fair and maintains the stability of a given grouping result.

A. Cost Sharing Mechanism

To address this challenge, we design a cost sharing mechanism with two desirable properties: (1) *group-strategyproofness* and (2) *sharing incentive*. In the following, we first present our mechanism, then prove it can satisfy the above two properties. In our cost sharing mechanism, the total cost of n clients is composed of two parts: one part goes to the data owner's account, and the other is for Ethereum miners; so does it for individual cost. Each user is equally responsible for the total payment to the data owner. Given a grouping result of k combined queries, the data owner will make a revenue of kc', each client paying kc'/n to him.

Any keyword in a combined query may be redundant for some of its group members, and it is unfair for a user to pay for a keyword he never requests. Thus, the total cost of searching a certain keyword is only borne by those users who request it. Thus, the cost sharing is at the granularity of n clients instead of each group. For each unique keyword, we calculate its total cost in all combined queries, and then evenly distribute the cost among all users querying this keyword. That is, if a keyword is queried by m of n users and appears in t of k combined queries, its total search cost is tc, and each one from m users is equally responsible for a cost share of tc/m.

We show how to share the total cost using the grouping result shown in Table IV(a) under the constraint of 3 keywords. For the rest of this paragraph, each client i is identified by his query q_i. The grouping result is $G_1 = \{q_1, q_3, q_7\}$, $G_2 = \{q_2, q_5\}$, and $G_3 = \{q_4, q_6\}$. Thus, the cost paid to the corresponding data owner is $3c^{'}$, equally distributed among 7 clients. Table Va presents total cost for each keyword and who should be fairly responsible for the corresponding cost. Table IV(b) gives the final split cost for each client. For example, client 1's total cost is $3c'/7 + (1/3 + 1/2 + 1)c$, where (1) $3c'/7$ is paid to the data owner, shared with all other 6 clients; (2) $c/3$ comes from querying keyword w_1, shared with users q_3 and q_7; (3) $c/2$ comes from querying keyword w_2, shared with q_3; (4) c comes from querying keyword w_4 by himself.

TABLE V: An Example of User Cost Sharing.

Keyword	Cost	Shared by
w_1	$1 \cdot c$	q_1, q_3, q_7
w_2	$1 \cdot c$	q_1, q_3
w_3	$0 \cdot c$	
w_4	$1 \cdot c$	q_1
w_5	$2 \cdot c$	q_2, q_5
w_6	$2 \cdot c$	q_2, q_4, q_5
w_7	$1 \cdot c$	q_4, q_6
w_8	$2 \cdot c$	q_2, q_4, q_6

(a) Cost of each keyword

Clients	Cost
q_1	$\frac{3}{7}c' + (\frac{1}{3} + \frac{1}{2} + 1) \cdot c$
q_2	$\frac{3}{7}c' + (\frac{2}{2} + \frac{2}{3} + \frac{2}{3}) \cdot c$
q_3	$\frac{3}{7}c' + (\frac{1}{3} + \frac{1}{2}) \cdot c$
q_4	$\frac{3}{7}c' + (\frac{2}{3} + \frac{1}{2} + \frac{2}{3}) \cdot c$
q_5	$\frac{3}{7}c' + (\frac{2}{2} + \frac{2}{3}) \cdot c$
q_6	$\frac{3}{7}c' + (\frac{1}{2} + \frac{2}{3}) \cdot c$
q_7	$\frac{3}{7}c' + \frac{1}{3} \cdot c$

(b) User individual cost

B. Theoretical Analysis

We present theoretical analysis to demonstrate that our cost sharing mechanism achieves some desirable properties. For group-strategyproofness, we should demonstrate each client will honestly disclose his real query request even if they are permitted to collude. For each keyword, if a client's dominant strategy is to truthfully tell whether he wants to query it or not, then truth-revealing is his dominant strategy. Thus, we can divide the whole proof into d steps, and the j-th step shows that each client would prefer revealing his real request on the keyword w_j in our cost-sharing mechanism. Thereby, we divide our cost sharing mechanism on keyword search part into d cost sharing methods, one for each keyword, then we prove each method satisfies group-strategyproofness.

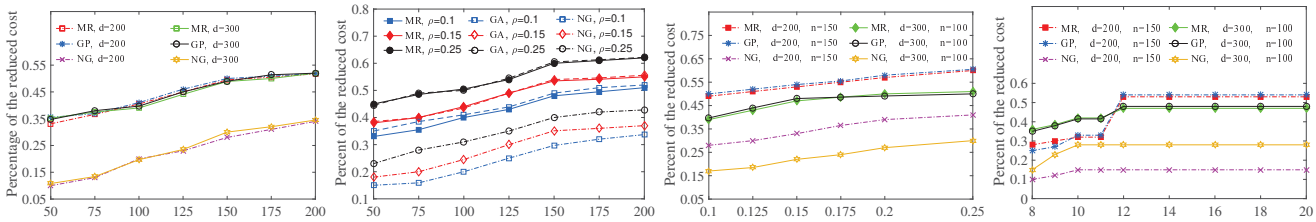
The cost sharing method of keyword w_j is a function, ξ_j, which distributes the total cost of searching for the j-th keyword, denoted as C_j, to its requesters. More formally, ξ_j takes two arguments, a subset of users G and a user q_i, and returns a nonnegative real number satisfying the following: (1) if $q_i \notin G$ then $\xi_j(G, q_i) = 0$, and (2) $\sum_{q_i \in G} \xi(G, q_i) = C_j$. As is proven in [20], if ξ_j is a cross-monotone, then the mechanism specified above is group-strategyproofness for keyword w_j. Thus, we need to prove ξ_j is cross-monotone. A cost sharing method can be said as cross-monotone if for $G \subseteq R$, $\xi_j(G, q_i) \geq \xi_j(R, q_i)$ for every $q_i \in G$.

Lemma 3. *For every* $j \in [1, d]$ *,* ξ_j *is cross-monotone.*

Proof. Any $q_i \in R \setminus G$ refers to a client not requesting the j-th keyword, thereby they are charged zero cost share. Thus, $G \subseteq R$, $\xi_j(G, q_i) = \xi_j(R, q_i)$ for every $q_i \in G$. Thus, ξ_j is a special cross-monotone cost sharing mechanism. □

Theorem 4. *Our cost sharing mechanism satisfies group-strategyproofness and sharing incentive for all clients.*

Proof. The property of group-strategyproofness can be proven using Lemma 3. To show sharing incentive, we should reveal, for any client, leaving his current assigned group would not bring him more benefits. Sending an individual query definitely brings more cost paid to the data owner, which is cost-inefficient. As is shown in Eq. (3), grouping is always



(a) X-axis : number of users. (b) X-axis : number of users. (c) X-axis : keyword density. (d) X-axis : delay constraint.

Fig. 3: Evaluations of grouping strategies on real query traces. MR: Mathematic Relaxation, GP: Greedy Partition, and NA: Naive Greedy.

beneficial for each client to save cost paid to data owners. Thus, no one has incentive to leave. □

VI. Performance Evaluation

Our evaluation consists of two parts. In the first part, we focus on evaluating our proposed cooperative search scheme on real query traces AOL [21]. In the second part, we implement an Ethereum testbed to demonstrate the practicality of our scheme, and also analyze the actual relationship between the keyword number and the search delay.

A. Cooperative Search Scheme

Our experiments evaluate two grouping strategies in terms of total cost reductions and the cost sharing mechanism in terms of individual cost saving. Mathematic relaxation was implemented with MATLAB-R2017b and greedy algorithms were implemented with Eclipse 4.6 in Java. All experiments are conducted on AOL. Ad AOL is a huge query collection, we randomly choose 200 users with 31 804 queried keywords in total, among which 17 786 are unique. Thus, a 400×17786 binary matrix is constructed to reveal each query's request on each keyword. Since it is still a large array, we semi-randomly select part of the matrix in each experiment to satisfy preset constraints on dictionary size, query number, and keyword density. For simplicity, we define two parameters: *keyword density* and *charge ratio*. Given a d-size dictionary and an n-query set Q, *keyword density* ρ of Q is defined as $\rho = \sum_{i=1}^{n} |q_i| / (n \times d)$. Given c' from a data owner and c from miners, *charge ratio* r is defined as $r = c'/c$.

Grouping strategies: We analyze the percentage of reduced total cost using our proposed grouping strategies: Mathematic Relaxation (using PGD here since the query volume is large), Naive Greedy and Greedy Partition. Fig. 3 shows, in all parameter settings, all strategies achieve cost reduction by at least 24.8%. Greedy Partition works slightly better than Mathematic Relaxation, and Naive Greedy achieves the least total cost reduction, which is around 50% of the other two strategies. Since the complexity of Naive Greedy and Greedy Partition is $O(n^2)$ and $O(n^3)$, respectively, we could see an inevitable tradeoff between efficiency and performance. Now, we analyze how each parameter influences the total cost reduction. In Fig. 3(a), as n increases, the total cost reduction also increases. Given a fixed ρ, changing d has little effect on the cost reduction. Fig. 3(b) reflects, as ρ increases from 0.1 to 0.25, the total cost is reduced by about 10% for each unique n. In Fig. 3(c), we have two set comparable parameters: $(d = 200, n = 150)$ and $(d = 300, n = 100)$. Given a fixed ρ, each set has the same number of 1s. From this experiment, we could see the first set has more cost savings, since smaller size of d yields higher chances of keyword overlapping. Fig. 3(d) reflects that the effect of delay constraint α on the total cost reduction decreases as its value increases.

Cost sharing mechanism: In the second part, we study individual cost saving under our cost sharing mechanism by picking up 10 users with $d = 100$. Each time, we change r and select a better one from grouping results from Mathematic Relaxation and Greedy Partition. We compare the cost reduction between individuals and the average. As is shown in Fig. 4, individuals can benefit from our grouping strategies. Besides, no user largely deviates from the average level, which shows our cost sharing mechanism can achieve fairness.

Summary: Both grouping strategies are local optimal. Mathematic Relaxation uses 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. Therefore, Greedy Partition is more appropriate for large scale query systems, and Mathematic Relaxation can be used as the baseline to measure the grouping quality. In terms of the cost sharing mechanism, each user can achieve cost savings near around the average saving.

B. Ethereum Testbed

To demonstrate the practicality of our scheme, we implemented a testbed in a simulated Ethereum network called TestRPC [22]. TestRPC is a fast and customizable blockchain emulator. It sets mining time as 0 while truly revealing execution time and gas consumption of a transaction. This design allows us to focus on the search delay itself without being affected by mining or waiting delays. Our Ethereum testbed can be helpful in revealing the real relationship between the number of keywords and search delay by the miners. This provides a better estimate for search delay in the real system, and thereby, a better estimation of the grouping constraint.

Keyword number and search delay: We conduct two experiments to verify the actual relation between the keyword number and search delay. We stored a 5KB database with 20 keywords and 30 files, each tagged with one or more keywords. In the first experiment, we randomly select 5 keywords and incrementally add 20 more each time to see how the execution time changes. The result shows, as the number of keywords increases, the delay time also increases while there exists a slowdown in its growth rate. In the second

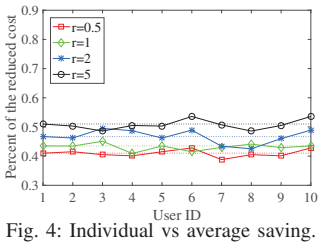


Fig. 4: Individual vs average saving.

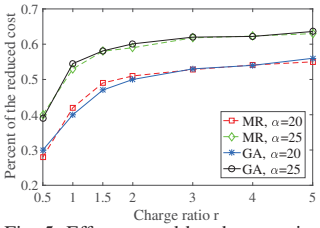


Fig. 5: Effect caused by charge ratio.

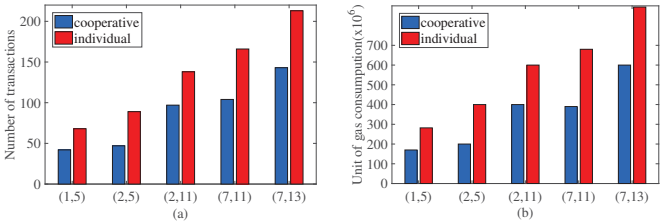


Fig. 6: Cost reduction using testbed (X-axis : number of (overlapping files, unique files) tuples in non-grouping search result).

experiment, we dedicatedly design 5 query sets, each including 4 queries. The total keyword number in each set is fixed at 10, while the unique keyword number changes. For each query set, we execute them in two ways: (1) executing all queries individually, and (2) executing a single query composed of 4 queries. We compare accumulative execution times and combined execution times, and analyze how execution time reduces as the number of overlapping keywords increases. The result of this experiment shows that the relationship between the number of overlapping keywords and execution time is nearly proportional. Based on the above results, we conclude, the search delay is at least sublinear to keyword numbers.

Charge ratio: In our real implementation, we store a 1.4MB database with 300 unique keywords and 2000 files. Each file is tagged with some different keywords. We issue 75 transactions in order to store the entire database in blockchain. When previously evaluating our cost sharing mechanism, we find that charge ratio r can affect individual cost savings as well as total cost reductions. Thus, when performing experiments on our testbed, we first analyze how charge ratio r can affect grouping results, hence changing cost reductions. As is shown in Fig. 5, there is a positive sublinear relationship between the total cost reduction and charge ratio r. In our previous sections, assuming $r = 1$ to yield a maximal reduction on total cost is acceptable, since it can be adjusted by a factor.

Four-user cooperative search: We also envision a small four-user setting with different queries, and conduct several optimal cooperative searches and their individual searches. Fig. 6 reflects the cost reduction in the form of transaction number and gas consumption amount, both of which are important cost measures in Ethereum. These two parameters follow a very similar changing pattern if given the same inputs. The reason is each transaction invokes executions of the same search function. As we can see, the cost reduction is positively related to the ratio of overlapping matched file number and the unique matched file number, which is a reflection of overlapping keyword number in original queries.

Summary: Using our testbed, we analyze the actual relation between the keyword number and the search delay, which is sublinear. Experiments are conducted to see how charge ratio affects cost reduction. The pricing for search part has more effects on the total cost reduction compared with the owner's pricing. The last experiment on four-user cooperative search has demonstrated the practicality of our proposed scheme.

VII. Conclusion

In this paper, we present a cooperative search scheme on an Ethereum-based data market. We take advantage of smart contract and gas system in Ethereum to separate a query cost into two parts: one for data owners and the other for miners. We also make use of grouping strategies to provide efficiency and cost savings at the user side. We provide three methods, suitable for different scenarios, to compute an efficient grouping result. Besides, we propose a fair cost sharing mechanism to split total cost among users given a grouping result. This mechanism guarantees some desirable properties such as group-strategyproofness and sharing incentive to avoid free-riders. The experiment results show that our scheme is efficient in terms of cost reduction for both the group as a whole and individuals.

References

[1] C. Cai, X. Yuan, and C. Wang, "Towards trustworthy and private keyword search in encrypted decentralized storage," in *ICC'17*.

[2] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *INFOCOM'18*.

[3] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[4] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *SPW'15*.

[5] "Ipfs," https://ipfs.io/.

[6] "Filecoin," https://filecoin.io/.

[7] "Datacoin," http://datacoin.info/.

[8] "Storj," https://storj.io/storj.pdf.

[9] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun, "Blockchain for large-scale internet of things data storage and protection," *IEEE Trans. on Services Computing*, 2018.

[10] "Aggdata," http://www.aggdata.com/.

[11] "Customlists.net," http://www.customlists.net/home.

[12] "Azure data market," https://datamarket.azure.com/.

[13] "Infochimps," http://www.infochimps.com/.

[14] "Amazon athena," https://aws.amazon.com/athena/.

[15] M. Balazinska, B. Howe, and D. Suciu, "Data markets in the cloud: An opportunity for the database community," *Proc. of the VLDB Endowment*, 2011.

[16] Q. Liu, C. C. Tan, J. Wu, and G. Wang, "Cooperative private searching in clouds," *J. Parallel Distrib. Comput.*, 2012.

[17] Q. Liu, Y. Guo, J. Wu, and G. Wang, "Effective query grouping strategy in clouds," *J. Computer Science and Technology*, 2017.

[18] N. Immorlica, M. Mahdian, and V. S. Mirrokni, "Limitations of cross-monotonic cost-sharing schemes," *ACM Trans. on Algorithms*, 2008.

[19] L. Zhao, H. Nagamochi, and T. Ibaraki, "Greedy splitting algorithms for approximating multiway partition problems," *Math Program*, 2005.

[20] N. R. Devanur, M. Mihail, and V. V. Vazirani, "Strategyproof cost-sharing mechanisms for set cover and facility location games," *Decision Support Systems*, 2005.

[21] "Aol," https://archive.org/details/AOL_search_data_leak_2006.

[22] "testrpc," https://www.npmjs.com/package/ethereumjs-testrpc.