Querying Shared Data with Security Heterogeneity

Querying Shared Data with Security Heterogeneity

Yang Cao[1], Wenfei Fan[1,2,3], Yanghao Wang[1], Ke Yi[4]
[1]University of Edinburgh [2]SICS, Shenzhen University [3]BDBC, Beihang University [4]HKUST
{yang.cao@, wenfei@inf, yanghao.wang@}ed.ac.uk yike@cse.ust.hk

ABSTRACT

There has been increasing need for secure data sharing. In practice a group of data owners often adopt a heterogeneous security scheme under which each pair of parties decide their own protocol to share data with diverse levels of trust. The scheme also keeps track of how the data is used.

This paper studies distributed SQL query answering in the heterogeneous security setting. We define query plans by incorporating toll functions determined by data sharing agreements and reflected in the use of various security facilities. We formalize query answering as a bi-criteria optimization problem, to minimize both data sharing toll and parallel query evaluation cost. We show that this problem is PSPACE-hard for SQL and Σ_3^p-hard for SPC, and it is in NEXPTIME. Despite the hardness, we develop a set of approximate algorithms to generate distributed query plans that minimize data sharing toll and reduce parallel evaluation cost. Using real-life and synthetic data, we empirically verify the effectiveness, scalability and efficiency of our algorithms.

ACM Reference Format:
Yang Cao, Wenfei Fan, Yanghao Wang, Ke Yi. 2020. Querying Shared Data with Security Heterogeneity. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

There have been increasing demands for sharing data from e-government [20], healthcare [37], finance [23] and the AI industry [15], among other things. For example, precision medicine requires sharing of clinical, genetic, environmental and lifestyle data for better disease treatment and prevention. However, security and privacy issues hamper data sharing, since organizations are becoming increasingly aware of the economical loss and legal liabilities due to data breaches.

To tackle it, a number of techniques have been devised to enable data sharing while providing certain security guarantees, *e.g.,* encryption schemes such as order-preserving symmetric encryption (OPE) [12] and homomorphic encryption (HOM) [19], Docker containers, or hardware-assisted enclaves [36] such as Intel SGX and ARM TrustZone.

The existing work assumes homogeneous settings, *i.e.,* the same security protocol is assumed between all pairs of peers. In many real-world scenarios, however, there are often various trust relationships and hence different security requirements between data owners, as illustrated by the following case study taken from our industry partner.

Example 1: A data-sharing company (name withheld) has built a blockchain-based platform (similar to, *e.g.,* MHMD [24]), to enable secure data analytics over distributed datasets owned by users such as government agencies, hospitals, clinics, researchers, drug stores, insurance firms and pharmaceutical companies, etc. Users of the same type are connected via an internal network, and these internal networks are connected to an external network via gateways.

Computations within an internal network can be carried out in Docker containers hosted by nodes of the network. Depending on the trust levels among the data owners, the container may use encryption schemes (*e.g.,* OPE or HOM) to secure data uploaded to it, or use plaintext when the users trust each other. Computations across multiple internal networks may be conducted in hardware-assisted enclaves. Such an enclave incurs much higher upfront costs than Docker since, among other things, it requires a consensus among all gateways in the blockchain and will be audited. Datasets uploaded to the enclaves can be either encrypted or not, depending on the contract used for the consensus.

These security facilities (*e.g.,* Docker and enclaves) reflect different security protocols and requirements between data owners and users. They cope with different threat levels, and incur *security costs* by their usage and upfront costs.

Below are some example applications on the platform.

(**A1**) One is to find people who register in multiple clinics. Since clinics maintain high trustworthiness, the computation can be done in the clinic internal network by using a Docker container: all clinics upload their registration datasets to the container, where the answers are computed.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

(**A2**) A government agent wants to find addresses of all households with at least one member who has contracted disease Z but has no health insurance. This involves government with household registration data, hospitals that have electronic medical records (EMR) of patients, and insurance firms. Government data can be uploaded to enclaves in the hospital network without encryption; hospital data can be loaded to enclaves in the insurance network with HOM encryption; and insurance data can be loaded to hospital enclaves with OPE encryption (more efficient but less secure than HOM) as hospitals have higher trust levels than insurance firms. □

Such emerging applications introduce new challenges.

(a) The need for a heterogeneous security scheme. An application may involve multiple types of peers (data owners and users), and various security facilities and guarantees are needed due to the trust levels between these peers. For example, plaintext can be shared between hospitals (via Docker) for **A1** and hospitals only share encrypted data with insurance firms via more costly enclaves (**A2**). To support these applications, a *heterogeneous* scheme is needed, to support *varying* trust levels and security means between different peers.

(b) Query processing with security heterogeneity. Query answering in a heterogeneous setting is much more challenging than in the homogeneous settings. We need to take into account various security charges when deciding where and how the computations should be carried out (*i.e.,* query planning), *e.g.,* how many containers/enclaves are needed and how to distribute them among sites at multiple networks. This is nontrivial. We want to reduce the security costs. At the same time, we want to minimize the parallel execution time of the query plan. For example, for **A2** of Example 1, it would be better to send insurance data to enclaves in the hospital network instead of the other way around.

Contributions & organization. This paper studies query evaluation in a heterogeneous security setting.

(1) Abstraction of data sharing scheme (Section 2). We make a first attempt to study query processing under a data sharing scheme with *heterogeneous security protocols*. We demonstrate that the scheme can support emerging applications that bear various levels of trust between different peers, as commonly found in the real world. We define distributed query plans and incorporate data sharing cost (security charge) in terms of toll functions that abstract the usage of various security facilities (*i.e.,* Docker containers/enclaves with plain/encrypted data) based on the access rights, trust levels and security demands among the peers.

(2) Querying shared data (Section 3). We formalize the problem of querying shared data under heterogeneous security as a bi-criteria optimization problem, to minimize both parallel query evaluation cost and data sharing toll. We show that the problem is highly nontrivial: while it is decidable (NEXPTIME), it is already PSPACE-hard for SQL and Σ_3^p-hard for SPC to find optimal plans in special cases. Despite these, we introduce a framework for querying shared data.

(3) Distributed plan generation (Section 4). Underlying the framework, we develop a polynomial-time (PTIME) algorithm to generate distributed query plans while minimizing data sharing toll. We show that the algorithm is an $O(\log n)$-approximation for joins, *i.e.,* its join plans are within $O(\log n)$ of optimal ones, where n is the number of data owners.

(4) Distributed plan optimization (Section 5). We further minimize the parallel evaluation cost of query plans generated in (3) while retaining a toll budget via workload rebalance. We show that the problem is also intractable. This said, we give a PTIME 2-approximate algorithm for rebalancing workload.

(5) Experimental study (Section 6). Using real-life and synthetic data, we empirically evaluate the effectiveness and efficiency of our plan generation algorithms. We find the following. (a) Security heterogeneity does have an evident impact on query evaluation performance. (b) Our proposed method is effective in reducing both data sharing toll and parallel execution cost, outperforming its competitors by 25.57 and 10.27 times on average, respectively. (c) Existing security systems can be readily incorporated into the data sharing scheme and serve as security facilities.

Position of the work. This work is not to introduce another security protocol. It is not to investigate security guarantees of heterogeneous security models; nor is it to improve existing secure database systems (*e.g.,* [9, 35]). Heterogeneous security protocols are already being used in real life, and the security community is studying their properties (*e.g.,* [14, 25]).

Instead, we study query evaluation over shared data when we are given heterogeneous security protocols. We aim to study the impact of such heterogeneous security protocols on query processing, in terms of costs incurred by data sharing agreements and reflected by the use of different security facilities. The costs stem from existing security facilities, protocols and systems. We make a first attempt to evaluate queries in their presence for emerging applications.

Related work. Distributed secure query processing has been well studied in homogeneous environments.

Complete trust. With complete trust among the data owners, the problem becomes the standard parallel/distributed query processing problem [33]; its goal is to minimize the communication costs of answering queries [4, 11, 27].

Related is the study of federated databases [28, 30, 40], which aim to provide an interface for querying a collection of distributed and autonomous relational databases. Hetero-

geneity has also been studied in this context, known as multistores or polystores *e.g.,* [17, 22, 26]. However, the heterogeneity here refers to various data and programming models used by different data owners, not security heterogeneity.

Hardware-assisted solutions. With hardware support (*e.g.,* security enclaves), query processing over distributed sources can be carried out with moderate overhead, *e.g.,* TrustedDB [8], Cipherbase [6] and EnclaveDB [36]. An enclave protects the data and code running inside of it from being spied upon, even if the entire software stack of the host is compromised. In addition, a remote party can verify the code running inside the enclave through a process known as *attestation*. Thus, two data owners without trust can still jointly compute a function, by sending their data to an enclave, which runs some code (attested by both owners) to compute the function. The enclave can be hosted by one of the data owners, or even a third (untrusted) party. Docker containers are used in lieu of an enclave if the system administrator can be trusted.

Software-only solutions. In the absence of special hardware support, one can still build a distributed query processing system over distrusted peers using homomorphic encryption [19] or secure multi-party computation (SMC) [13, 39, 44], such as CryptDB [35], Monomi [41] and SMCQL [9, 10]. Both homomorphic encryption and SMC are capable of computing arbitrary functions over the data, but those general-purpose solutions are not very practical. In practice, one has to limit the types of queries supported by designing special-purpose protocols. Even so, these software-only solutions tend to be much more expensive than those with hardware support.

Locally differential privacy (LDP). LDP [16] has recently emerged as another approach to querying distributed data. LDP algorithms for answering range queries on a single table have been developed [29, 43]. Different from security-based solutions, query results in the LDP model have to be probabilistic with certain random noise, and information leakage will accumulate in the LDP model when more queries are run on the same data.

This work differs from the prior work in the following. (1) We study a heterogeneous security environment, a setting being increasingly used in practice. It supports data sharing among a group of data owners and allows each pair of hosts to adopt a security protocol of their choice, as opposed to the homogeneous security assumption in the previous work. (2) We provide the first abstraction of heterogeneous security protocols. (3) We formalize the problem of querying shared data as a bi-criteria optimization problem, and study its complexity. (4) We propose an approach to answering generic SQL queries in the heterogeneous security setting.

2 DATA SHARING WITH SECURITY HETEROGENEITY

We start with a scheme to abstract data sharing with security heterogeneity (Section 2.1), and then formalize the problem of querying shared data under the scheme (Section 2.2).

2.1 An Abstraction of Data Sharing

The scheme, referred to as DShare, is to abstract computations over shared datasets with heterogeneous security protocols. It is characterized by the notions of *data owners*, a *query planner*, *data sharing pacts* and *distributed query plans*.

A group of data owners often agree upon data sharing protocols in practice. Each owner contributes its data for sharing, and is also a client of the shared data under the protocols.

Data owners. A collection of *data owners*, or simply *sites*, $\mathcal{S} = (\mathrm{S}_1, \ldots, \mathrm{S}_n)$ agree to support query services collectively over their private data. Each owner manages its data by its own DBMS and has its own local database schema.

We assume a global schema $\mathcal{R} = (R_1, \ldots, R_k)$, deduced by, *e.g.,* schema mapping, to provide a uniform interface to write queries against the data, where R_i is a relation schema. Owner S_i has an instance $\mathcal{D}_i$ of $\mathcal{R}$ for $i \in [1, n]$. Here some relations in $\mathcal{D}_i$ are possibly empty, *i.e.,* $\mathcal{D}_i$ does not necessarily have every relation of $\mathcal{R}$. We denote $(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ by $\mathcal{D}$ and refer to it as a *distributed instance of* $\mathcal{R}$ *at* $\mathcal{S}$. The answer to a query Q over $\mathcal{D}$ is defined as $Q(\bigcup_i^n \mathcal{D})$ under the normal interpretation. Note that via renaming, this also allows us to compute local answers at any single site or any subset of $\mathcal{S}$.

Query planner. Each data owner may issue a query Q, to compute the answer to Q over $\mathcal{D}$. The query will be handled by a trusted third party called *query planner*. Upon receiving a query Q from a data owner, the planner will come up with a *distributed query plan* that picks security facilities (Docker or enclave), to comply with the security protocols agreed among the sites, and encryption schemes required for the operations. It also estimates a security charge, referred to as a *toll*, for the query plan. If it is agreeable by the query issuer, the query planner will instruct the data owners to carry out the query plan, charges the toll to the query issuer, and allocates the revenue to the data owners accordingly.

The query planner oversees the execution of the query plans and acts as an intermediary between the data owners and the query issuer. However, it does not access any of the local databases or carry out operations in the query plans itself.

Data sharing pact. We next abstract the varying security protocols between pairs of data owners based on their trust levels. We use the following *computation model*.

(1) Data is shared using *capsules*, logic units to which data owners can transfer and upload datasets; physically, a capsule can be instantiated with a Docker container, an enclave,

an SMC system such as SMCQL [9], or even a trusted third party. All computations are carried out in capsules only.

(2) Each capsule C is associated with a site S_j, referred to as *a capsule hosted by* S_j. Computation in C has direct access to the data on S_j but cannot access data at other sites except the part that is uploaded to C. When the computation in C completes, the intermediate results are stored at S_j in a protected mode via, *e.g.,* access right controls, OPE encryption [12], or symmetric encryption with keys held only by the query issuer as commonly used in cloud cryptograph such as Azure [1]; then the capsule C will be terminated.

A *security protocol* for a pair of sites $(\mathsf{S}_i, \mathsf{S}_j)$ specifies:

(a) the *lowest* security guarantees that a capsule hosted by S_j must attain in order for S_j to share the data of S_i; and

(b) encryption of data at S_i, *i.e.,* which part of the data can be shared, what data needs to be encrypted before sending it to a capsule of S_j, and what encryption scheme to use.

A *data sharing pact* ρ for a set $\mathcal{S}$ of sites consists of (1) a security protocol for each pair of sites in $\mathcal{S}$, and (2) a *toll* function Toll() that measures the *costs* of all types of available capsules that can be employed to evaluate queries. In the real world, such costs are incurred by, *e.g.,* renting trusted third party facilities as the capsules and encryption overhead.

Distributed query plan. We next define query plans over a set $\mathcal{S}$ of sites *w.r.t.* a data sharing pact ρ.

Consider a distributed instance $\mathcal{D}$ at $\mathcal{S}$. A *distributed query plan* ξ on $\mathcal{D}$ at $\mathcal{S}$ is a DAG (directed acyclic graph), where the nodes of ξ denote *atomic operations* and edges represent their dependencies. Each atomic operation δ is an $(n+3)$-tuple $(\mathrm{op}, \mathrm{t_c}, X_1, \ldots, X_n, j)$, where (i) op is an operator in the relational algebra (RA; projection π, selection σ, natural join $\bowtie$, set difference $-$, set union $\cup$ and renaming λ); (ii) $\mathrm{t_c}$ is a capsule of a certain type, *e.g.,* Docker, enclave, SMC system, or trusted third party; (iii) $j \in [1, n]$ denotes the site that hosts a capsule to carry op out; and (iv) X_i is a relation, which is either part of data in $\mathcal{D}_i$ that can be shared with S_j by the security protocol between S_i and S_j, or the intermediate result $\mathcal{I}_i$ computed at site S_i by operations prior to δ in ξ.

Executing $\delta = (\mathrm{op}, \mathrm{t_c}, X_1, \ldots, X_n, j)$ involves steps below:

(1) Set up a capsule C of type $\mathrm{t_c}$ for op, hosted by site S_j, such that for each $i \in [1, n]$, C meets the security requirement of ρ for $(\mathsf{S}_i, \mathsf{S}_j)$, where (a) $X_i \neq \emptyset$ or (b) there exists $X_k \neq \emptyset$ that contains intermediate answers computed over data from S_i.

(2) For $i \in [1, n]$, transfer X_i from S_i to the capsule C hosted by S_j, based on the security protocol between S_i and S_j.

(3) Perform the computation $\mathcal{I}'_j \leftarrow \mathrm{op}(X_1, \ldots, X_n)$ and store $\mathcal{I}'_j$ in a protected mode (*e.g.,* encrypted with OPE) at S_j.

(4) Add the relation $\mathcal{I}'_j$ to $\mathcal{I}_j$ at S_j.

That is, intermediate results of op are computed and stored as such to comply with the security guarantees of pact ρ. In particular, when $X_i = \emptyset$ for all $i \in [1, n], i \neq j$, $(\mathrm{op}, \mathrm{t_c}, X_1, \ldots, X_n, j)$ simply executes op on local data X_j at site S_j.

Condition (1b) ensures that each followup operation $\delta' = (\mathrm{op}', X'_1, \ldots, X'_n, p)$ of δ also complies with the security requirement of $(\mathsf{S}_i, \mathsf{S}_p)$ if $X'_j \neq \emptyset$ and X'_j contains intermediate results $\mathcal{I}'_j$ computed by δ from the data of S_i, even if $X'_i = \emptyset$.

Edges of the DAG plan ξ specifies the dependencies of the atomic operations in ξ. In particular, if there exists X_i (in atomic operation $\delta = (\mathrm{op}, \mathrm{t_c}, X_1, \ldots, X_n, j)$) that comes from relations at site S_j computed by atomic operation δ', then there exists a directed edge from δ' to δ in ξ.

The execution of the query plan is mediated and monitored by the query planer, and it takes place at the participating sites only. After executing the query plan, the sites will send the results to the query planner, who then decrypts and returns the results to the client who issued the query.

Toll. We next presents the toll function Toll(). For a query plan ξ over $\mathcal{S}$, $\mathsf{Toll}(\xi)$ is the sum of $\mathsf{Toll}(\delta)$ for all operations $\delta = (\mathrm{op}, \mathrm{t_c}, X_1, \ldots, X_n, j)$ in ξ, where $\mathsf{Toll}(\delta)$ estimates the charge for executing δ. It consists of the following:

(a) an upfront cost Toll_0 for setting up the capsule for op;

(b) cost Toll_d for transferring remote data via secure channel to the capsule for op hosted by S_j, determined by the amount of data and encryption overhead; and

(c) cost Toll_c for executing op in the capsule, determined by the duration that the computation op takes.

More specifically, (a) Toll_0 depends on the type of the capsule; (b) Toll_d is measured as $\sum_{i=1}^{n} \mathsf{Toll}_{(i,j)}(X_i)$ for all sites S_i that have data X_i required by δ, where $\mathsf{Toll}_{(i,j)}(X_i)$ is the cost of encrypting and transferring X_i of S_i to the capsule hosted by S_j; it is determined by the size of X_i and the encryption scheme required by the protocol between S_i and S_j; in particular, $\mathsf{Toll}_{(j,j)}(X_j) = 0$ since C is hosted by S_j and can access its local data X_j without extra cost. Finally, (c) Toll_c is the charge for sustaining the capsule for executing op.

Using our familiar terms, we refer to Toll_d and Toll_c as *communication* and *computation* cost, respectively. These costs stem from the security protocol between S_i and S_j, and are reflected as costs incurred by the security facility employed, including *e.g.,* encryption cost, beyond their normal scope.

Example 2: [**Case study**] Continuing with Example 1, we show that applications **A1** and **A2** can be abstracted by DShare. Denote by (a) Household(address, pid, name) the registration relation maintained by government, (b) Reg(pid, clinic) and EMR(pid, disease) the clinic registration relation and medical record relation, respectively, owned by the clinics and hospitals, and (c) Insurance(pid, company, policy) the

customer records maintained by insurance firms. We present their data sharing pacts ρ with associated toll functions.

Application **A1**. It is expressed as a join query $\pi_{\text{pid}}\text{Reg}(\text{pid}, \text{clinic}) \bowtie_{\text{pid}=\text{pid}' \wedge \text{clinic} \neq \text{clinic}'} \text{Reg}(\text{pid}', \text{clinic}')$. The pact ρ specifies the lowest security level for a capsule C. Based on this, the query planner may pick Docker container as C, and remote data will be directly uploaded to C. The shared data will not be stored after the computation since a capsule is ephemeral. These meet the security requirements of **A1** since clinics are trusted and do not risk side-channel leakage.

The toll function estimates the costs of available capsules C, and is determined by the type, configuration and duration of C to use (which in turn depends on the operations to be carried out in C). For Docker, $\text{Toll}_0 = 0$ since Docker incurs negligible upfront cost; $\text{Toll}_{(i,j)}(X_i) = c_{ij}|X_i|$ is the communication cost for transferring X_i from S_i to the Docker at S_j (c_{ij} is a coefficient denoting the unit network price); and $\text{Toll}_c(\bowtie)$ is $c|\text{Reg}|^2$ (c is a coefficient similar to c_{ij}), the cost of sustaining the Docker container for the join.

Application **A2**. It is an RA query $\pi_{\text{address}}((\text{Household} \bowtie \sigma_{\text{disease}=Z}\text{EMR}) - \text{Insurance})$[1]. As the query involves data from three internal networks, pact ρ requires a higher security level for data sharing as described in Example 1. Based on this, the query planner may take enclaves as capsules. In particular, since insurance firms have lower level of trust than hospitals, data is required to be encrypted using HOM [19] before uploading it to insurance enclaves, to prevent leakage of user identifications and their EMR records.

Here the toll function is determined by the capsule types, operations, encryption schemes and communication. First consider, *e.g.,* the join δ of Household and EMR at a hospital site S_j. For enclave, $\text{Toll}(\delta)$ is estimated as follows: $\text{Toll}_0 = L$, where L is the upfront cost of setting up the enclave at S_j, and is estimated as the average time for reaching the consensus among the gateways; $\text{Toll}_{(i,j)}(X_i)$ is $c_{ij}|X_i|$, where c_{ij} is a coefficient reflecting the cost of shipping Household to the enclave at S_j; and $\text{Toll}_c(\bowtie) = c|D_{\text{Household}}| * |D_{\text{EMR}}|$, where $D_{\text{Household}}$ and D_{EMR} are the datasets for the join at S_j.

Now consider operation $I - \text{Insurance}$, where I is the join result above. Assume that the set difference takes place in a capsule C hosted by the insurance network. Then the type of C is determined by both the protocol between government and insurance firm and the one between hospital and insurance, to warrant security for each data owner involved. □

As shown above, in practice toll functions can be readily deduced from the types of capsules and the complexity of operations. Alternatively, as a common practice, they can also be empirically estimated by testing or learning over small dataset samples. In addition, for blockchain-based data sharing systems similar to Example 1 or MHMD [24], tolls often denote the economic incentives defined by smart contracts for consensus. There has also been recent work from the security community on toll (cost) models of hybrid protocols, *e.g.,* [14, 25]. Note that toll functions only specify toll charges of all possible capsules usage in an application; they do not determine which, where and how capsules to be used.

DShare allows arbitrary positive polynomial functions as $\text{Toll}(\delta)$ that are composed of submodular set functions [21] for $\text{Toll}_{(i,j)}(X)$ (of Toll_d) and Toll_c (bi-modular [18] if op of δ is a binary operator, *e.g.,* join). All toll functions in Example 2 are of this type. Our industry partners find that these suffice to express common security charges in practice.

Guarantees and properties. We adopt the following threat model. The data owners are *semi-honest* (*a.k.a. honest but curious*), *i.e.,* each owner will faithfully execute the query plan, but may try to derive information about other parties' data. This is why all operations are executed in capsules and the intermediate relations are stored in protected mode. The query planner runs at a trusted third party and is assumed trustworthy, similar to the honest broker in secure database systems *e.g.,* [9, 35], but without direct access to data.

Under this threat model, DShare enforces the security protocols of data sharing pact ρ by (a) the trusted query planner, and (b) proper capsules for carrying out query plans.

(1) The pact specifies the *minimum* security requirement between each pair S_i and S_j of sites. The query planner enforces the security guarantee by picking right capsules. Each operation $(\text{op}, \text{t}_c, X_1, \ldots, X_n, j)$ is performed by a capsule that meets the *maximum* of all lowest security levels for (S_i, S_j) ($i \in [1, n]$). Followup operations retain no less security levels.

(2) Depending on the security protocols, different pairs of sites may have distinct security requirements. The planner picks capsules that meet the security requirements and minimize the cost, *e.g.,* it picks docker containers for **A1** of Example 1, which satisfy the security requirements specified by the protocol and are cheaper than enclaves and SMC systems.

Remark. Composing security protocols is a challenging and active topic of the security community (*e.g.,* [14, 25]). It has emerged in semi-trusted data federations, *e.g.,* MHMD [24] and Example 1. This paper takes a heterogeneous setting used in practice and focuses on query planner that selects capsules and distributes computations across the federation, to improve query performance while retaining the required security levels. This said, the query planner can be adapted to other security composition and propagation protocols.

2.2 The Problem of Querying Shared Data

Critical to DShare is its query planner. While there has been

[1]To simplify notation, we allow set difference between relations with different sets of attributes; more precisely, we define $R - S = R - (R \ltimes S)$.

a host of research on security protocols and facilities, no prior work has studied how to generate query plans that comply with a data sharing pact with security heterogeneity.

This motivates us to study the *toll-bounded query answering* problem, denoted by TBQA. Informally, it is to find the best distributed query plan for a given query subject to a toll budget imposed by a data sharing pact. It is stated as follows.

- *Input*: A global schema $\mathcal{R}$, n sites $\mathcal{S}$, a distributed instance $\mathcal{D}$ of $\mathcal{R}$ over $\mathcal{S}$, a data sharing pact ρ, a natural number B, and an RA query Q over $\mathcal{R}$.
- *Output*: A distributed plan ξ for Q over $\mathcal{D}$ such that the toll $\mathsf{Toll}(\xi)$ of ξ over $\mathcal{D}$ under ρ is no larger than B.
- *Objective*: Minimize the *parallel execution cost* of ξ over $\mathcal{D}$, denoted by $\boldsymbol{c}(\xi, \mathcal{D})$.

As remarked earlier, a data sharing pact ρ specifies only the minimum security requirements for sharing data between sites and their associated toll charges. We have to find query plan ξ that determines, in addition to conventional planning, how to select and distribute capsules for computations that can meet heterogeneous security requirements of ρ, while taking advantages of the heterogeneity and minimizing its parallel execution cost $\boldsymbol{c}(\xi, \mathcal{D})$.

To complete the statement of TBQA, we define $\boldsymbol{c}(\xi, \mathcal{D})$ below. Let $c(\delta, \mathcal{D})$ be the execution cost of atomic operation δ over $\mathcal{D}$. Then $\boldsymbol{c}(\xi, \mathcal{D})$ is inductively defined as follows:

- If ξ is a single atomic operation δ, $\boldsymbol{c}(\xi, \mathcal{D}) = c(\delta, \mathcal{D})$.
- If ξ consists of sub-plans $\xi_1, \ldots, \xi_l$ and an atomic operation δ, where $\xi_1, \ldots, \xi_l$ are predecessors of δ in ξ, then $\boldsymbol{c}(\xi, \mathcal{D}) = \max(\boldsymbol{c}(\xi_1, \mathcal{D}), \ldots, \boldsymbol{c}(\xi_l, \mathcal{D})) + c(\delta, \mathcal{D})$.

Intuitively, $\boldsymbol{c}(\xi, \mathcal{D})$ characterizes the total parallel execution costs of atomic operations in ξ when parallel execution of independent atomic operations is fully exploited.

We assume that the query planner can efficiently estimate the cost incurred by an atomic operation $\delta = (\mathrm{op}, \mathrm{t_c}, X_1, \ldots, X_n, j)$. For example, when op is $R \bowtie S$, $c(\delta, \mathcal{D})$ is $|R| \times |S|$.

3 QUERYING SHARED DATA

In this section, we first study the complexity of querying shared data and then outline our approach to solving TBQA.

Complexity of TBQA. Denote by $\mathsf{TBQA}^{\mathsf{d}}$ the decision version of TBQA. That is to decide, given the same input of TBQA and an additional number L, whether there exists a distributed query plan ξ for Q over $\mathcal{D}$ such that its toll is at most B and its parallel execution cost $\boldsymbol{c}(\xi, \mathcal{D})$ is at most L.

We also study a related problem, referred to as *toll-bounded answerability* problem and denoted by TBA. Given $\mathcal{R}$, $\mathcal{S}$, ρ and B as in TBQA, it is to decide whether there exists a distributed query plan ξ for Q over $\mathcal{D}$ with toll at most B.

Intuitively, TBA is to check whether TBQA even has a feasible solution or not. TBQA is at least as hard as TBA.

We say that a data sharing pact ρ is *simple* if $\mathsf{Toll}_0 = \mathsf{Toll}_c = 0$ and $\mathsf{Toll}_{(i,j)}(X) = c_{ij}|X|$. Both problems are intractable even under such simple pacts that involve only two sites.

Theorem 1: *Both* $\mathsf{TBQA}^{\mathsf{d}}$ *and* TBA *are*

(1) decidable in NEXPTIME*;*

(2) PSPACE*-hard even when ρ is simple; and*

(3) Σ_3^p-hard even when Q is in SPC *and ρ is simple.*

Moreover, (2) and (3) hold even when $\mathcal{S}$ has two sites only. □

Our approach. In light of Theorem 1, practical solutions to TBQA have to be approximate. We next propose such an approach, which consists of two steps outlined below.

Step (1): Finding toll-minimized canonical plans (Section 4). We first generate a distributed query plan ξ_Q for Q in a *canonical* form: ξ_Q extends the algebra tree T_Q (cf. [3]) of Q into a DAG by replacing each algebra operation op of Q with a distributed query plan ξ_{op} that has minimized the toll in $\mathcal{D}$. Note that here an edge from op_1 to op_2 of Q in T_Q may be extended to multiple edges, which connect atomic operations in ξ_{op_1} to those in ξ_{op_2} based on their dependencies.

Step (2): Reducing parallel execution cost (Section 5). Given ξ_Q of step (1), we check whether $\mathsf{Toll}(\xi_Q)$ of ξ_Q exceeds the toll budget B. We return "No" if so, *i.e.,* budget B is too small to answer Q in $\mathcal{D}$ under ρ. Otherwise, we further improve ξ_Q by making use of the remaining toll allowance, to reduce its parallel execution cost $\boldsymbol{c}(\xi_Q, \mathcal{D})$ without exceeding B.

4 GENERATING TOLL-MINIMIZED PLANS

In this section, we show how to carry out step (1) of our approach (Section 3). Given an RA query Q, a distributed instance $\mathcal{D}$ of schema $\mathcal{R}$ at n sites $\mathcal{S}$ and a data sharing pact ρ, we generate a distributed plan ξ_Q for Q, which consists of toll-minimized plan ξ_δ for each operation δ in Q. Below we focus on joins; the other RA operations are similar and simpler.

Approximability. One can verify that even for joins, TBQA remains intractable, by reduction from the vertex cover problem, which is NP-complete [34]. Nonetheless, there exists a PTIME approximation. Assume that $\mathcal{D}$ is reasonably large, and constant Toll_0 is negligible *w.r.t.* Toll_d or Toll_c.

Theorem 2: *There exists a* PTIME *$O(\log n)$-approximation algorithm for computing plans with minimum toll for joins.* □

As a proof, we give such an algorithm, denoted by MTJ.

Consider query $Q = R \bowtie T$. Algorithm MTJ generates a plan ξ for Q over $\mathcal{D}$ by reduction to the minimum set cover (MSC) problem [34]. Below we first present algorithm MTJ by reduction to MSC, which gives us an $O(\log n)$-approximation. However, a direct use of the reduction yields a naive version of MTJ with an exponential time (EXPTIME) complexity. Nonetheless, we develop a technique that is able to reduce its complexity from EXPTIME to PTIME.

ALGORITHM 1: Algorithm MTJ (naive implementation)

Input: query $Q = R \bowtie T$, database $\mathcal{D}$ over n sites $\mathcal{S}$, pact ρ.
Output: a distributed plan ξ for Q.

1 $C \leftarrow \emptyset$;
2 **while** C does not include all work units of Q **do**
3 $\quad (i_*, W_*) \leftarrow \arg\min_{(i,W)\in\mathcal{W}} \frac{t(i,W)}{|W \setminus \bigcup_{(i',W')\in C} W'|}$;
4 $\quad C \leftarrow C \cup \{(i_*, W_*)\}$;
5 interpret C as atomic operations ; // pick capsule types t_c as the lowest possible to meet all security requirements of protocols for work units in C
6 **return** ξ consisting of the atomic operations;

Approximation by reduction. We start with a naive version of MTJ by approximation-preserving reduction [7] to MSC, so that MTJ computes toll minimized join plans by making use of available approximate algorithms for MSC.

Reduction. The idea of the reduction is to (a) represent each query plan ξ for join query Q as a "workload" distribution plan that assigns necessary data movement for answering Q in $\mathcal{D}$ among the sites; and (b) reformulate the assignment problem as a variant of MSC that admits a PTIME logarithmic-factor approximation algorithm [42].

Consider a join query $Q = R \bowtie T$, distributed database $\mathcal{D}$ over sites $S_1, \ldots, S_n$ and data sharing pact ρ. We construct an instance of MSC, *i.e.,* a universe U of elements and a set $\mathcal{W}$ of weighted subsets of U, such that each c-approximation answer to MSC encodes a distributed join plan for Q with toll at most c-times of the minimum toll among all plans for Q.

Denote by D_i^R the instance of relation R at site S_i ($i \in [1, n]$); similarly for D_i^T. For convenience, we assume *w.l.o.g.* that neither D_i^R nor D_i^T is empty for all $i \in [1, n]$. For any $i, j \in [1, n]$, $u_{ij} = [D_i^R, D_j^T]$ is called a *work unit* of Q in $\mathcal{D}$. Then:

(1) U consists of all work units of Q in $\mathcal{D}$; and
(2) $\mathcal{W}$ consists of pairs (i, W) for all $i \in [1, n]$ and $W \subseteq U$. We say that (i, W) *covers element* $u_{jk} = [D_j^R, D_k^T]$ in U if $u_{jk} \in W$. The weight of (i, W), denoted by $t(i, W)$, is defined as the sum of the total Toll_d of fetching D_j^R and D_k^T from sites S_j and S_k to site S_i, and total Toll_c of computing $D_j^R \bowtie D_k^T$, for all units $[D_j^R, D_k^T]$ in W. Note that this has to take into account toll sharing for relations appearing in multiple work units in W.

Algorithm. One can readily verify that the reduction is approximation-preserving [7]. This gives us an $O(\log n)$-approximation algorithm, as a naive implementation of MTJ, for computing minimum toll join plans (Algorithm 1), by using the $O(\log |U|)$-approximation of MSC [42] ($|U| = n^2$). Here set covering C specifies the assignment of atomic operations for the work units in C to their host sites (line 4). The capsule types of the atomic operations are such picked that they minimize the cost while satisfying all the relevant security levels specified in the protocols for C (line 5).

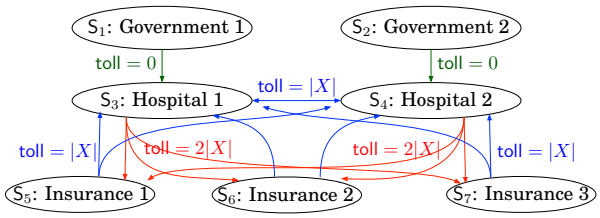


Figure 1: A simplified scenario of A3 for Example 3 *(assume that ρ is simple and $\mathsf{Toll}_{(i,j)}(X)$ for Toll_d are color-coded)*

Example 3: Recall the query for **A2** given in Example 2, denoted by Q. Assume a simplified data sharing scenarios shown in Fig. 1. We show how algorithm MTJ generates the distributed query plan ξ_Q depicted in Fig. 2 for Q.

Take the join $\mathsf{op}_1 = \mathsf{Household} \bowtie \sigma_{\mathsf{disease}=Z}\mathsf{Medical}$ of Q for example. Then op_1 has a set $\mathsf{W}_{\mathsf{op}_1}$ of 4 work units $u_{ij} = [\mathcal{D}_i, I_{j2}]$ for all $i \in \{1, 2\}, j \in \{3, 4\}$ (see Fig. 2). After the reduction, the set $\mathcal{W}$ consists of (i, W) for all $W \subseteq \mathsf{W}_{\mathsf{op}_1}$. MTJ picks $(3, \{u_{13}, u_{23}\})$ and $(4, \{u_{14}, u_{24}\})$ as C that covers $\mathsf{W}_{\mathsf{op}_1}$, with total toll 0. It then interprets C as ξ_{op_1} consisting of the atomic joins for I_{32} and I_{42} of Fig. 2. Note that this is actually the optimal plan for op_1 since ξ_{op_1} incurs no toll at all.

Assume that sizes $|I_{32}|$, $|I_{42}|$ and $|\mathcal{D}_i|$ for all $i \in [5, 7]$ are N. Then along the same lines, MTJ generates a distributed plan ξ_{op_2} with the atomic operations for I_{33} and I_{43} of Fig. 2, which is also optimal for op_2 with $\mathsf{Toll}(\xi_{\mathsf{op}_2}) = 6N$. □

From exponential to polynomial. While Algorithm 1 is an $O(\log n)$-approximation, it has an exponential time complexity since $\mathcal{W}$ of line 3 is of size exponential in $|U|$ (*i.e.,* exponential in n^2). Nonetheless, below we show that this can be reduced to PTIME, based on the following:

(a) (i_*, W_*) identified in line 3 of Algorithm 1 equals
$$\arg\min_{i\in[1,n]} \arg\min_{W\subseteq U} \frac{t(i,W)}{|W \setminus \bigcup_{(i',W')\in C} W'|},$$
where U is the set of all work units of Q in $\mathcal{D}$.
(b) For each i, computing $\arg\min_{W\subseteq U} \frac{t(i,W)}{|W\setminus\bigcup_{(i',W')\in C} W'|}$ is equivalent to finding the minimum α such that there exists a subset W of U with $f_\alpha(W) \le 0$, where
$$f_\alpha(W) = t(i, W) - \alpha|W \setminus \cup_{(i',W')\in C} W'|.$$
(c) For any fixed α, checking whether this holds can be done efficiently in PTIME since one can prove that $f_\alpha(W)$ is a submodular function, and submodular minimization can be done in PTIME via, *e.g.,* [21].

From these, the **while** loop (line 3) of Algorithm 1 can actually be implemented in PTIME by computing $\arg\min_{W\subseteq U} \frac{t(i,W)}{|W\setminus\bigcup_{(i',W')\in C} W'|}$ as W_i for each $i \in [1, n]$, which can be implemented by a binary search for the minimum α in $[0, \alpha_{\max}]$ such that $\min_W f_\alpha(W) \le 0$, where $\alpha_{\max} = \max_{i,j\in[1,n]}(\max_{k\in[1,n]} c_{(i,k)} * (|D_i^R| + |D_j^T|))$, in which $c_{(i,k)}$ is

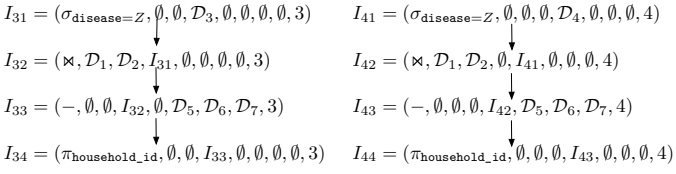


Figure 2: Distributed query plan ξ_Q in Example 3 (the type $\mathsf{t_c}$ of atomic operations is omitted since ρ is simple)

the constant in the toll function $\mathsf{Toll}_{(i,k)}(X) = c_{(i,k)}|X|$ specified by ρ. The search terminates when the range $[a, b]$ for α has a gap ($|b-a|$) less than $\frac{1}{n^2}$. Hence, there are at most $\log n^2 * \alpha_{\max}$ rounds of search, where each round is an invocation of submodular minimization, which is in PTIME (*e.g.,* [21]).

That is, the **while** loop of Algorithm 1 is reduced to PTIME in n and $\log(n|\mathcal{D}|)$. Therefore, MTJ can be implemented in PTIME in n and $\log|\mathcal{D}|$, and is an $O(\log n)$-approximation for computing minimum-toll distributed plans for joins.

5 PLAN REBALANCING

After generating a toll-minimized canonical plan ξ_Q for Q in Section 4, we next study how to further optimize ξ_Q by reducing its parallel execution cost $\boldsymbol{c}(\xi_Q, \mathcal{D})$. This is to carry out step (2) of our approach outlined in Section 3. While the adjustments may increase the toll of the revised plan, we make sure that the toll is below the budget B, *i.e.,* we make use of the remaining toll allowance $B - \mathsf{Toll}(\xi_Q)$ to reduce $\boldsymbol{c}(\xi_Q, \mathcal{D})$.

Our technique, referred to as *plan rebalancing*, is motivated by the following. Consider the sub-plan ξ_{op_i} of ξ_Q for an operation op_i of Q. Here ξ_{op_i} is generated to minimize toll (Section 4) and hence could be imbalanced. Observe that $\boldsymbol{c}(\xi_Q, \mathcal{D})$ is dominated by the maximum cost of individual sites (Section 2.2); hence imbalanced workloads increase $\boldsymbol{c}(\xi_Q, \mathcal{D})$.

In light of this, rebalancing works by iteratively applying an *atomic balancing* operator κ_b to optimize ξ_{op_i} under its allocated toll budget B_i (Section 4) for each operation op_i, such that (a) the optimized sub-plan $\kappa_\mathsf{b}(\xi_{\mathsf{op}_i})$ is guaranteed to have a lower cost than ξ_{op_i}, and (b) $\kappa_\mathsf{b}(\xi_{\mathsf{op}_i})$ incurs at most B_i of toll. That is, κ_b makes use of toll allowance B_i on op_i, and re-distributes the work units handled by ξ_{op_i} in a more balanced and optimized way, to reduce the cost of ξ_{op_i}.

However, there are two key challenges to rebalancing.

(C1) How to design κ_b such that it can optimize ξ_{op_i} in an optimal way under a given toll budget B_i on op_i?

(C2) How to distribute the total toll budget B over all sub-plans of ξ_Q (*i.e.,* operations of Q) so that the total cost reduction of ξ_Q is maximized?

We tackle (C2) by iteratively allocating toll budget to individual sub-plans of ξ_Q in the same spirit of the gradient descent algorithm [38] for optimization problems Below we focus on (C1): we propose operator κ_b and show that it is a near-optimal design of its kind.

Given a sub-plan ξ_{op_i}, operator κ_b works in two phases: (1) it first re-distributes the work units of ξ_{op_i} across n sites subject to a toll budget B_i allocated to op_i; this yields plan ξ'_{op_i} that guarantees to reduce the cost; and (2) it then prepares the answers of ξ'_{op_i} for $\xi_{\mathsf{op}_{i+1}}$ that is subsequent to ξ_{op_i}. Here phase (2) is carried out by simply recovering the input distribution for the subsequent sub-plan $\xi_{\mathsf{op}_{i+1}}$ of ξ_{op_i}. It is to ensure that ξ'_{op_i} is compatible with its subsequent sub-plan $\xi_{\mathsf{op}_{i+1}}$, since $\xi_{\mathsf{op}_{i+1}}$ works with a certain input distribution (*i.e.,* the distribution of the answer of ξ_{op_i}) due to the heterogeneous security protocols (see Section 2.1).

Below we focus on phase (1) of κ_b. We parameterize κ_b with an integer k that controls the degree of changes to ξ_{op_i}: the larger k is, the larger cost is reduced but more toll is consumed. We denote by $\kappa_\mathsf{b}[k]$ the operator κ_b instantiated with k. We apply $\kappa_\mathsf{b}[k]$ to ξ_{op_i} by selecting k work units of ξ_{op_i} for re-distribution, to reduce its parallel execution cost.

It is nontrivial to pick k units that inflict the lowest cost. Below we first provide an algorithm for unit selection, and then prove that it is near-optimal among all such algorithms.

Algorithm ReBal. The algorithm, denoted by ReBal works as follows. Given a sub-plan ξ_{op_i} of ξ_Q computed in Section 4, a database $\mathcal{D}$ over n sites $\mathcal{S}$, a data sharing pact ρ and parameter k for κ_b, ReBal returns an optimized sub-plan ξ'_{op_i} by re-distributing k work units for ξ_{op_i}.

More specifically, ReBal first (a) identifies a set $\mathcal{W}_k$ of k bottleneck work units for op_i, and then (b) re-distributes them to improve ξ_{op_i}. It does (a) by iteratively identifying bottleneck sites, picking and adding its bottleneck work units to $\mathcal{W}_k$, where bottleneck sites have work units of maximum costs among all. It carries out (b) by assigning work units in $\mathcal{W}_k$ one by one to sites with least workload *w.r.t.* the cost of executing all work units of op_i

Analysis. Algorithm ReBal is near-optimal of all algorithms of its kind. More specifically, denote by $O[k](\xi_{\mathsf{op}_i})$ the class of algorithms that optimize ξ_{op_i} by selecting and re-distributing k work units of ξ_{op_i}. Then we have the following.

Proposition 3: *(1) It is* NP*-complete to find the optimal optimization of ξ_{op_i} by re-distributing k work units.*

(2) ReBal *is a 2-approximation of the optimal in $O[k](\xi_{\mathsf{op}_i})$ and is in $O(n^2 \log n)$-time.* □

6 EXPERIMENTAL STUDY

Using benchmarks and real-life datasets, we conducted experiments to evaluate (1) the impact of heterogeneous security protocols on querying shared data; (2) the effectiveness of our toll-minimized planning technique; (3) the effectiveness of our toll-bounded plan optimization; and (4) integration of SMC-based system (SMCQL [9]) and related comparison.

Experimental setting. We start with the setting.

Real-life dataset. We used TFACC, a real-life dataset that integrates the MOT Test Data [32] of Ministry of Transport test for vehicles in the UK from 2005 to 2016, and National Public Transport Access Nodes (NaPTAN) [31]. It has 19 tables with 113 attributes, about 46.7GB of data in total.

We generated 30 RA queries over TFACC. We used 5 query templates with the number #join of joins varying from 1 to 5. We generated the queries by instantiating the templates with values randomly selected from the dataset.

TPCH *benchmark*. We also used standard benchmark TPCH [2] with its built-in queries. TPCH generates data using TPC-H dbgen [2], with 8 relations. It has 22 built-in SQL queries, which were rewritten into RA queries in our tests. Along the same lines as for TFACC, we also additionally generated 30 random queries with #join varying from 1 to 5.

Each relation of the datasets was randomly partitioned and distributed over a random subset of the machines (sites).

Data sharing pacts. We used three simple data sharing pacts.

(1) Uniform pact $\boldsymbol{\rho_U}$. Under $\boldsymbol{\rho_U}$, the toll functions are of the form $\mathsf{Toll}_{i,j}(X) = c_{i,j}(|X|)$ for all pairs S_i and S_j of sites, and the coefficients $c_{i,j}$ are from a uniform distribution of $[0, 100]$. Here $|X|$ is the size of the transferred data X.

(2) Power law pact $\boldsymbol{\rho_P}$. Under $\boldsymbol{\rho_P}$, the toll functions are similar to those under $\boldsymbol{\rho_U}$ except that the toll coefficients are from a power law distribution of [0, 100].

(3) Constant pact $\boldsymbol{\rho}_{c/\infty}[p]$. Under $\boldsymbol{\rho}_{c/\infty}[p]$, for any pair of sites S_i and S_j, $\mathsf{Toll}_{i,j}(X)$ is either a random constant $C_{ij} = C$ or $+\infty$, where the probability of $\mathsf{Toll}_{i,j}(X) = +\infty$ is $p\%$.

Implementation. We developed a prototype system, referred to DASH (DatA SHare), for querying shared data under a heterogeneous data sharing pact such as $\boldsymbol{\rho_U}$, $\boldsymbol{\rho_P}$ and $\boldsymbol{\rho}_{c/\infty}[p]$.

Prototyping. DASH employs PostgreSQL as the DBMS at each site. It implements the framework of Section 3 as the query planner to generate distributed plans. By default, DASH uses PostgreSQL to execute plans at each site. Given a toll budget B, DASH employs the techniques of Sections 4 and 5 to generate plans subject to B while minimizing parallel cost. If B is not specified, DASH generates plans with minimized toll.

Baselines. We are not aware of any existing systems that query shared data and support heterogeneous security protocols. Nonetheless, we designed and implemented three variants of DASH as baselines for comparison:

- ONE: selects the *best* site S_* to evaluate a query Q centrally at S_*, *i.e.,* transferring all queried relations to S_* and executing Q at S_*; it ensures that at site S_*, the evaluation incurs the minimum toll among all sites.
- DASH^0: follows DASH to process operations op of Q one by one, but centrally at the best site for each op.
- DASH^-: follows the framework of DASH to process operations of Q one by one, but randomly assigns work units to sites with data involved, *e.g.,* assigning $D_i^R \bowtie D_j^S$ to S_i.

Model	TPC-H				TFACC			
	DASH	DASH^-	DASH^0	ONE	DASH	DASH^-	DASH^0	ONE
$\boldsymbol{\rho_U}$	49.94	4079.5	399.7	179.35	32.9	1523.2	58.8	55.4
$\boldsymbol{\rho_P}$	10.56	1455.02	73.42	30.77	8.2	545.4	15.5	14.2
$\boldsymbol{\rho}_{c/\infty}[5\%]$	30.4	$+\infty$	70.08	42.88	16.1	$+\infty$	21.2	19.6
$\boldsymbol{\rho}_{c/\infty}[10\%]$	31.49	$+\infty$	84.96	44.32	20.9	$+\infty$	23.1	21.2
$\boldsymbol{\rho}_{c/\infty}[15\%]$	45.47	$+\infty$	$+\infty$	$+\infty$	24.8	$+\infty$	$+\infty$	$+\infty$
$\boldsymbol{\rho}_{c/\infty}[50\%]$	74.88	$+\infty$	$+\infty$	$+\infty$	30.8	$+\infty$	$+\infty$	$+\infty$
$\boldsymbol{\rho}_{c/\infty}[55\%]$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

**In all toll functions* $\mathsf{Toll}_{i,j}(X) = c_{ij}|X|$, $|X| = 1$ *if* X *has 1GB of size.*

Table 1: Average toll usage per query (Exp-1)

Configuration. The experiments were conducted on 20 Linux servers, each with 6-core Intel i5-8400 2.8GHz CPU, 32 GB of memory and 1TB of HDD. The instances are fully connected with high speed intra-network channels. By default, we used model $\boldsymbol{\rho_P}$, entire TFACC, 32 GB of TPCH, and all queries.

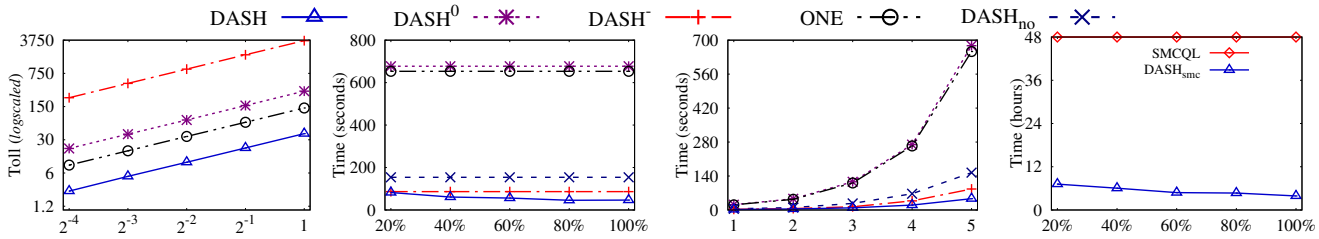
Experimental Results. We next report our findings.

Exp-1: Impact of heterogeneous security protocols. We first evaluated the impact of security protocols on toll consumed by query evaluation over the distributed datasets. We evaluated all queries over both datasets under all three data sharing pacts $\boldsymbol{\rho_U}$, $\boldsymbol{\rho_P}$ and $\boldsymbol{\rho}_{c/\infty}[p]$, when $p\%$ ranges from 5% to 55%. Table 1 reports the average toll usage per query by all four methods. The results tell us the following.

(1) Different security pacts charge toll differently. Under $\boldsymbol{\rho}_{c/\infty}[p]$, some TFACC or TPCH queries cannot be answered with a finite toll by DASH^-, DASH^0 or ONE when $p \geq 15\%$, while DASH can answer all the queries even when $p = 50\%$.

(2) On both TPCH and TFACC, DASH consistently generates plans that incur the minimum toll under all the security pacts. For example, on TFACC under $\boldsymbol{\rho_P}$, the average toll consumption per query of DASH is 45.2, 1.8 and 1.7 times less than that of DASH^-, DASH^0 and ONE, respectively.

Exp-2: Effectiveness of toll-minimized planning. We next evaluated the effectiveness of toll-minimized planning of DASH. We tested the average toll usage per query for query evaluation when varying the sizes $|D|$ of datasets from $2^{-4} \times |D_{\max}|$ to $|D_{\max}|$, where $|D_{\max}| = 46.7$GB for TFACC and 32 GB for TPCH. As reported in Fig. 3a for TPCH, we can see the following. (a) Over larger datasets all methods consume larger toll, as expected. (b) However, DASH consistently charges much smaller toll than the other methods, *e.g.,* 3.48, 7.83 and 91.51 times less than ONE, DASH^0 and DASH^- on average over TPCH, respectively; moreover, the gap increases with larger D. The results for TFACC are similar (see [5]).



(a) TPCH: *varying D* ($\times|D_{\max}|$) (b) TPCH: *varying toll budget B* (c) TPCH: *varying D* ($\times|D_{\max}|$) (d) TPCH: *varying toll budget B*

Figure 3: Evaluation of the effectiveness and scalability of DASH **for querying shared data (Exp-2, 3, 4);** results show that (a) DASH generates faster plans than all baselines in all cases while consuming smaller tolls; (b) our optimization effectively reduce evaluation time with larger toll budges; (c) DASH can work with SMCQL (SMC-based secure database) and speed it up

Exp-3: Effectiveness of optimization. We next evaluated the effectiveness of toll-bounded query optimization of DASH. We compared with a variant of DASH, denoted by DASH$_{no}$, which turned off the optimization of Section 5. We evaluated the average query evaluation time of DASH and DASH$_{no}$ with all queries, full datasets, and a total toll budget $B_m = 10|D|$, where $|D|$ is the total size of the dataset of all sites. To favor ONE, DASH^0 and DASH$^-$, we set B_m large so that these baselines can answer all the queries within the toll budget.

(1) Varying toll budget. Varying the total budget B from $20\%B_m$ to B_m, we tested the query evaluation time of all methods. The result for TPCH is reported in Fig. 3b and shows the following. (a) DASH is the fastest among all. *e.g.,* DASH is 1.86, 14.54 and 14.02 times faster than DASH$^-$, DASH^0 and ONE, respectively, when $B = B_m$ on TPCH. (b) The optimization of DASH is effective: DASH is on average 2.76 and 2.55 times faster than DASH$_{no}$ on the two datasets, respectively.

(3) Varying datasets. Varying the size of datasets in the same way as Exp-2 with full toll budget B_m, we tested the average evaluation time per query. The results on TPCH is given in Fig. 3c (the results on TFACC are similar and omitted). Similar to Exp-3(1), DASH consistently performs the best among all the methods, and does better when the datasets get larger.

Exp-4: Integration with SMCQL [9]. We evaluated the feasibility and performance of integrating DASH with SMC systems such as SMCQL [9]. We took SMCQL as the capsules for DASH and denote the integrated system by DASH$_{smc}$. We evaluated the performance of DASH$_{smc}$ and SMCQL over 1 GB of TPCH (SMCQL does not scale to larger datasets). In particular, to simulate the case study of Example 1, we used 20 machines and partitioned them into three groups, with 2, 10 and 8 machines representing governments, hospitals and insurance firms, respectively. To favor SMCQL and prevent DASH$_{smc}$ from bypassing SMCQL capsules, we set the protocols the same as Fig. 1 except that (a) insurance machines do not send data to hospitals, and (b) all computations over insurance machines must use SMCQL capsules.

We randomly distributed TPCH relations over the machines. Using three TPCH queries Q4, Q12 and Q19 (simplified due to the restriction of query support on SMCQL), we evaluated the performance of DASH$_{smc}$ and SMCQL.

(1) SMCQL can be naturally integrated into DASH as capsules and becomes more practical in the heterogeneous setting. DASH$_{smc}$ is on average more than 18.89 times faster than SMCQL (SMCQL cannot finish within 48 hours for all cases).

(2) DASH$_{smc}$ improves by 1.83 times when B increases from $20\%B_m$ to B_m while SMCQL is insensitive to B (Fig. 3d).

Summary. We find the following. (1) Security heterogeneity has a big impact on querying shared data. (2) Our proposed method effectively reduces both toll consumption and parallel execution cost. On average DASH consumes 2.59, 4.64, 69.47 times less toll than ONE, DASH^0 and DASH$^-$, respectively, and is 14.16, 14.44 and 2.2 times faster. (3) Existing systems can be integrated with our method as capsules and alleviate efficiency bottleneck in the heterogeneous setting; it speeds up SMCQL by 18.89 times over 1GB of TPCH.

7 CONCLUSION

We have made a first attempt to study query answering under heterogeneous security models. We have abstracted security heterogeneity, formalized the problem of querying shared data, studied its complexity, and developed approximate algorithms for generating distributed query plans. Our experimental study has shown that our method is promising in reducing both security charge and parallel execution cost.

The work aims to demonstrate the need, challenges and feasibility of querying shared data with security heterogeneity. We are currently evaluating costs incurred by enclaves, Docker, secure communication channels and various encryption schemes, in order to make our toll functions more accurate. We are also developing a guidance for practitioners to set up a realistic toll budget for answering their queries.

Acknowledgements. Fan and Cao are supported in part by ERC 652976, Royal Society Wolfson Research Merit Award WRM/R1/180014, EPSRC EP/M025268/1, Shenzhen Institute of Computing Sciences, and Beijing Advanced Innovation Center for Big Data and Brain Computing.

REFERENCES

[1] 2018. Azure encryption. *https://docs.microsoft.com/en-us/azure/security/fundamentals/encryption-overview*.

[2] 2019. TPC-H. *http://www.tpc.org/tpch/*.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[4] Foto N. Afrati and Jeffrey D. Ullman. 2011. Optimizing Multiway Joins in a Map-Reduce Environment. *TKDE* 23, 9 (2011), 1282–1298.

[5] Anonymous. 2020. Full version. *https://bit.ly/it2lx*.

[6] Arvind Arasu, Spyros Blanas, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, Prasang Upadhyaya, and Ramarathnam Venkatesan. 2013. Secure database-as-a-service with Cipherbase. In *SIGMOD*.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. 1999. *Complexity and Approximability Properties: Combinatorial Optimization Problems and Their Approximability Properties*.

[8] Sumeet Bajaj and Radu Sion. 2011. TrustedDB: A Trusted Hardware based Database with Privacy and Data Confidentiality. In *SIGMOD*.

[9] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel N. Kho, and Jennie Rogers. 2017. SMCQL: Secure Query Processing for Private Data Networks. *PVLDB* 10, 6 (2017), 673–684.

[10] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. ShrinkWrap: Efficient SQL Query Processing in Differentially Private Data Federations. *PVLDB* 12, 3 (2018), 307–320.

[11] Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. Communication steps for parallel query processing. In *PODS*. 273–284.

[12] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. 2009. Order-Preserving Symmetric Encryption. In *EUROCRYPT*.

[13] Elette Boyle, Kai-Min Chung, and Rafael Pass. 2015. Large-Scale Secure Computation: Multi-party Computation for (Parallel) RAM Programs. In *CRYPTO*. 742–762.

[14] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. 2018. HyCC: Compilation of Hybrid Protocols for Practical Secure Computation. In *CCS*.

[15] Department for Digital, Culture, Media & Sport and Department for Business, Energy & Industrial Strategy. 2017. Growing the artificial intelligence industry in the UK.

[16] John C Duchi, Michael I Jordan, and Martin J Wainwright. 2013. Local privacy and statistical minimax rates. In *FOCS*. 429–438.

[17] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Record* 44, 2 (2015), 11–16.

[18] Satoru Fujishige and Satoru Iwata. 2005. Bisubmodular Function Minimization. *SIAM J. Discrete Math.* 19, 4 (2005).

[19] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC*.

[20] Government Digital Service, HM Passport Office and UK Statistics Authority. 2018. Information sharing code of practice.

[21] Martin Grötschel, László Lovász, and Alexander Schrijver. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1, 2 (1981), 169–197.

[22] Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. 2014. Demonstration of the Myria big data management service. In *SIGMOD*.

[23] HM Treasury. 2015. Data sharing and open data in banking: Response to the call for evidence.

[24] Horizon 2020 Research and Innovation Action. 2016. My Health, My Data. *http://www.myhealthmydata.eu/*.

[25] Muhammad Ishaq, Ana L. Milanova, and Vassilis Zikas. 2019. Efficient MPC via Program Analysis: A Framework for Efficient Optimal Mixing. In *CCS*.

[26] Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jiménez-Peris, Raquel Pau, and José Pereira. 2016. The CloudMdsQL Multistore System. In *SIGMOD*.

[27] Paraschos Koutris and Dan Suciu. 2011. Parallel evaluation of conjunctive queries. In *PODS*. ACM, 223–234.

[28] Ralf Kramer. 1997. Databases on the Web: Technologies for Federation Architectures and Case Studies (Tutorial). In *SIGMOD*. 503–506.

[29] Tejas Kulkarni. 2019. Answering Range Queries Under Local Differential Privacy. In *SIGMOD*.

[30] Ee-Peng Lim, San-Yih Hwang, Jaideep Srivastava, Dave Clements, and M. Ganesh. 1995. Myriad: Design and Implementation of a Federated Database Prototype. *Softw., Pract. Exper.* 25, 5 (1995), 533–562.

[31] Find open data. 2014. *http://data.gov.uk/dataset/naptan*.

[32] Find open data. 2019. *https://data.gov.uk/dataset/e3939ef8-30c7-4ca8-9c7c-ad9475cc9b2f/anonymised-mot-tests-and-results*.

[33] M. Tamer Özsu and Patrick Valduriez. 2011. *Principles of Distributed Database Systems, Third Edition*. Springer.

[34] Christos H Papadimitriou. 1994. *Computational Complexity*. Addison-Wesley.

[35] Raluca Ada Popa, Catherine M. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *SOSP*.

[36] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: a secure database using SGX. In *IEEE Security & Privacy*.

[37] The Register. 2016. *https://www.theregister.co.uk/2016/07/06/caredata_binned/*.

[38] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *CoRR* abs/1609.04747 (2016).

[39] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613.

[40] Amit P. Sheth and James A. Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.* 22, 3 (1990), 183–236.

[41] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing analytical queries over encrypted data. In *PVLDB*.

[42] Vijay V. Vazirani. 2003. *Approximation Algorithms*. Springer.

[43] Tianhao Wang, Bolin Ding, Jingren Zhou, Cheng Hong, Zhicong Huang, Ninghui Li, and Somesh Jha. 2019. Answering Multi-Dimensional Analytical Queries under Local Differential Privacy. In *SIGMOD*.

[44] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *FOCS*. 160–164.