Hu-Fu: A Data Federation System for Secure Spatial Queries

Xuchen Pan[†], Yongxin Tong[†], Chunbo Xue[†], Zimu Zhou[#], Junping Du[◇], Yuxiang Zeng[‡], Yexuan Shi[†], Xiaofei Zhang[§], Lei Chen[‡], Yi Xu[†], Ke Xu[†], Weifeng Lv[†]
[†]State Key Laboratory of Software Development Environment, Beihang University, China
[†]Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, China
[#]Singapore Management University [◇]Beijing University of Posts and Telecommunications
[‡]The Hong Kong University of Science and Technology [§]University of Memphis
[†]{panxuchen, yxtong, xuechunbo, skyxuan, xuy, kexu, lwf}@buaa.edu.cn, [#]zimuzhou@smu.edu.sg,
[◇]junpingd@bupt.edu.cn, [‡]{yzengal, leichen}@cse.ust.hk, [§]xiaofei.zhang@memphis.edu

ABSTRACT

The increasing concerns on data security limit the sharing of data distributedly stored at multiple data owners and impede the scale of spatial queries over big urban data. In response, data federation systems have emerged to perform secure queries across multiple data owners leveraging secure multi-party computation. However, existing systems are designed for relational data. They are highly inefficient on spatial queries and limited in usability. In this demonstration, we introduce Hu-Fu, the first data federation system for secure spatial queries with high efficiency and usability. Hu-Fu is designed from the perspectives of the query user and the data owner for high usability and decomposes a spatial query into as many plaintext operators and as few secure operators as possible for high efficiency. We demonstrate the deployment and usage of Hu-Fu via cross-company taxi-calling, a popular smart city application.

PVLDB Reference Format:
Xuchen Pan, Yongxin Tong, Chunbo Xue, Zimu Zhou, Junping Du, Yuxiang Zeng, Yexuan Shi, Xiaofei Zhang, Lei Chen, Yi Xu, Ke Xu, and Weifeng Lv. Hu-Fu: A Data Federation System for Secure Spatial Queries. PVLDB, 15(12): XXX-XXX, 2022.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and technical report have been made available at https://github.com/BUAA-BDA/Hu-Fu.

1 INTRODUCTION

Efficient and secure processing of spatial queries over big urban data is crucial to scale up smart city applications. Urban-scale spatial datasets are often distributedly owned by multiple parties, where sharing raw data among parties or uploading raw data to a third party (*e.g.*, a cloud) is prohibitive due to legal regulations (*e.g.*, GDPR [9]) or commercial reasons.

An emerging concept of secure queries over distributed data is data federation, which consists of multiple data owners, who manage their data autonomously. A query user can perform secure queries across data of all owners in a data federation. The concept [1, 10] advances conventional federated databases [6] by protecting the query execution via secure multi-party computation (SMC).

Despite recent data federation systems for relational data [1, 10], they are unfit for spatial queries in smart city applications. On the one hand, directly extending these systems to spatial data can be inefficient. For example, a kNN query on a data federation can be at least two orders of magnitude slower than a plaintext query, where secure operations take up over 90% of the time cost [8]. On the other hand, these systems are limited in usability. Data owners in smart city applications may have different schemas and heterogeneous databases, which are not supported by prior systems [1, 10].

In this paper, we propose Hu-Fu, a data federation system for secure spatial queries with high efficiency and usability. Hu-Fu mainly optimizes five basic secure spatial queries, which we call *federated spatial queries*, including federated range query/counting, kNN query, distance join, and kNN join. It follows the semi-honest adversary assumption in [1, 10] but eliminates the need for an honest broker and can support a data federation of ten data owners [8]. In terms of efficiency, Hu-Fu is up to 4 orders of magnitude faster and 5 orders of magnitude lower in communication than [1, 10] with spatial extensions. For high usability, Hu-Fu allows data owners to modify its local table schemas for schema mapping across owners and adapts to multiple spatial database systems, *e.g.*, PostGIS, MySQL, SpatiaLite, Simba, GeoMesa, and SpatialHadoop.

At the algorithm level, Hu-Fu decomposes a federated spatial query into as many plaintext operators and as few secure operators as possible without compromising security to improve efficiency, and all the operations to be independent of the data owner's environment to easily adapt to heterogeneous databases. At the system level, Hu-Fu implements a query rewriter with novel query decomposition and an easy-to-use query interface with SQL support at the query user side, and a secure engine with efficient secure operator implementations as well as an adapter for heterogeneous databases at the data owner side.

We demonstrate the design and usage of Hu-Fu from the perspectives of both a data owner and a query user. From the data owner perspective, attendees can publish local table schemas to Hu-Fu for data federation construction. From the query user perspective, attendees can connect to multiple data owners and execute federated spatial queries. We showcase Hu-Fu via cross-company taxi-calling, a prevailing smart city application.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:XX.XX/XXX.XX

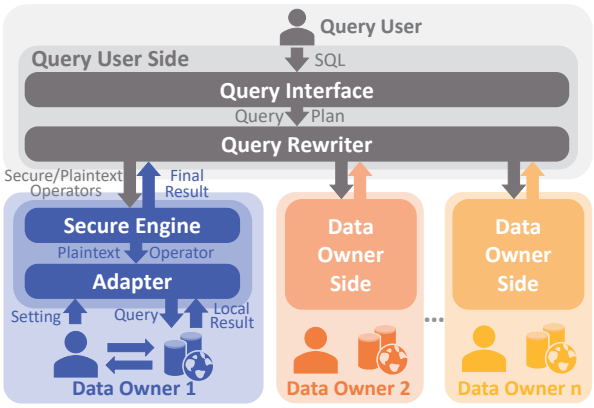

Figure 1: Illustration of Hu-Fu architecture.

2 HU-FU OVERVIEW

Hu-Fu is an efficient, easy-to-use data federation system for spatial queries. This section presents its overall architecture and workflow.

2.1 Architecture

Hu-Fu consists of both the *data owner side* and *query user side* (see Fig. 1). A data owner can set security constraints such as which tables are accessible by the query users and whether the attributes in the tables need to be protected by security techniques. A query user only knows the table schema shared by data owners, and issues federated spatial queries to Hu-Fu for the querying results. We explain the components of Hu-Fu on each side below.

Data Owner Side (Sec. 3). The data owner side of Hu-Fu consists of a *secure engine* and an *adapter*.

- **Secure Engine (Sec. 3.1).** It defines and implements a set of secure operators to execute federated spatial queries. The secure engine first checks whether the operators sent by the query user side meet the security constraints set by the data owner, then calls the adapter to execute the plaintext operators in the database, and finally assembles the local result with the secure operators.
- **Adapter (Sec. 3.2).** This module executes plaintext operators as spatial queries in the local database. To support a data federation with heterogeneous databases, the adapter implements different plaintext operators for different databases, *e.g.*, PostGIS, SpatiaLite, MySQL, GeoMesa, Simba and SpatialHadoop.

Query User Side (Sec. 4). The query user side of Hu-Fu includes a *query interface* and a *query rewriter*.

- **Query Interface (Sec. 4.1).** It provides the query user a unified global schema of the multiple tables from different data owners (horizontal sliced). The query interface also supports federated spatial queries written in SQL.
- **Query Rewriter (Sec. 4.2).** This module decomposes federated spatial queries into a series of plaintext and secure operators defined by the data owner side of Hu-Fu. The plaintext operators are performed within each owner's local database by the adapters, while the secure operators involve SMC protocols across owners, which are implemented by the secure engine.

2.2 Workflow

Consider Hu-Fu with a query user and n data owners (see Fig. 1). The query user issues a federated spatial query in SQL to the Hu-Fu user side. The query is first parsed by the query interface into a query plan. Then the query rewriter transforms the query plan into a sequence of plaintext and secure operators. These operators are then sent to the data owner side for execution. First, the adapter executes plaintext operators on the underlying spatial databases of each data owner to get the local results. Afterward, the secure engine collects the local results and performs the secure operators for the final result, which is returned to the user.

3 SYSTEM DESIGN: DATA OWNER SIDE

The data owner side of Hu-Fu defines and implements the operators and interfaces on top of the heterogeneous databases of individual data owners for efficient and easy-to-use federated spatial query execution. These functionalities are achieved by the secure engine and the adapter, as explained below.

3.1 Secure Engine

The secure engine defines and implements the *secure operators*. It is also responsible for checking the security constraints of the operators from the query user side.

Secure Operators. The secure operators securely assemble local results returned by plaintext operators across multiple data owners. The secure engine defines and implements three secure operators: *secure summation*, *secure comparison*, and *secure set union*.

- **Secure Summation.** It calculates $\sum_{i=1}^{n} v_i$, where v_i is a number held by data owner i. We implement the operator based on [4] to avoid leaking v_i to any data owner j $(i \neq j)$ nor the query user.
- **Secure Comparison.** It compares a user given value k with $\sum_{i=1}^{n} v_i$, and v_i is a number held by data owner i. We implement the operator by extending [3]. It ensures that either v_i or $\sum_{i=1}^{n} v_i$ is confidential to any data owner j $(i \neq j)$ and the query user.
- **Secure Set Union.** It computes the union of spatial objects from owners $\bigcup_{i=1}^{n} S_i$ (S_i is the object set in owner i) without leaking the ownership of any object of owner i to owners j $(i \neq j)$ nor the query user. We implement the operator based on [5].

We select dedicated SMC protocols to implement each secure operator for high efficiency and support more than two parties. More detailed definition and implementation can be found in [8].

Security Constraint Checking. The security constraints determine which tables can be accessed by the query user side and which attributes in the tables require security technique protection.

Data owners set the security constraints on the owner side and send them to query users along with the published table schemas. The basic operators should comply with these constraints. However, these basic operators may be forged by adversaries to violate these security constraints to access the prohibited tables or bypass security operations. Thus, the security engine will check each operator received and compare the tables and attributes that the operator needs to access with security constraints set by the data owner. The operators will be rejected if any constraint violation is found.

3.2 Adapter

The adapter defines and implements the *plaintext operators*. It also facilities easy construction of a data federation by providing interfaces to modify local table schemas.

Plaintext Operators. The plaintext operators can be executed locally in the databases of each data owner without security concerns. The adapter of Hu-Fu defines only two plaintext operators:

plaintext range query, and *plaintext range counting*, because they are simple and supported by almost all spatial data systems. The plaintext operators are implemented by calling the corresponding queries on local databases, which harnesses the optimization of spatial queries brought by spatial databases. For example, in the PostGIS adapter, a plaintext range counting on table `taxi` with the center p and radius r is implemented by calling the SQL below.

```
SELECT COUNT(*) FROM taxi
WHERE ST_DWithin(p, taxi.location, r);
```

To adapt to other spatial databases, the data owner only needs to re-implement the two plaintext operators.

Interface to Publish Table Schemas. To facilitate data federation construction, Hu-Fu adapter provides interfaces for data owners to modify the local table schemas before publishing them to query users. This function is necessary because the tables representing the same information may have different schemas across data owners, while unified schemas are needed for querying. When publishing a schema, the adapter allows modifying the name of the schema and its attributes, and each attribute can be assigned security constraints or marked as hidden. Consequently, data owners can negotiate a unified schema for the data federation (example in Sec. 5.2.1).

4 SYSTEM DESIGN: QUERY USER SIDE

The query user side of Hu-Fu is responsible to parse the federated spatial queries input by the query user and generate efficient federated spatial query plans without compromising security. These functionalities are achieved by the query interface and query rewriter, as explained below.

4.1 Query Interface

The query interface integrates schemas published by different owners into unified global schemas and supports federated spatial queries in SQL format over these global schemas.

Unified Global Schema. The query interface collects all the schemas published by the data owners and combines multiple schemas with the same attributes along with the security constraints into a single global schema (see Sec. 5.2.2), which is then used to perform federated spatial queries.

Federated Spatial Queries in SQL. The query interface extends the SQL parser of Calcite [2] to support spatial queries in SQL. To improve usability, we add two keywords: `DWithin` and `kNN` for federated range query/counting and federated kNN query. For example, a federated kNN query with point p and parameter k on a global schema `taxi` can be written in SQL as

```
SELECT * FROM taxi WHERE kNN(p, taxi.location, k);
```

4.2 Query Rewriter

For efficient query execution, the query rewriter decomposes a federated spatial query into as many plaintext operators and as few secure operators as possible without compromising security. With the basic operators defined in Sec. 3, the query rewriter classifies common federated spatial queries (range query, range counting, distance join, kNN, kNN join) into two categories: *radius-known queries* and *radius-unknown queries*. More importantly, the query rewriter designs novel decomposition plans for these two categories of queries, as described below.

Decomposing Radius-Known Queries. Federated range query, range counting and distance join (which can be decomposed into multiple federated range queries) belong to radius-known queries. Assuming a data federation with n data owners, *federated range query/counting* can be decomposed into n plaintext query/counting to retrieve the local results, and then uses secure summation/set union to assemble the final result.

Decomposing Radius-Unknown Queries. The radius-unknown queries consist of federated kNN query and kNN join (which can be broken into multiple federated kNN queries). For a *federated kNN query*, we first use binary search to obtain a radius (denoted by *thres*) and then retrieve the spatial objects within this radius. Note that obtaining the exact counting result during the binary search via secure summation may leak extra information. For example, the query user can get the number of objects within a range, which reveals the federation's data distribution. Hence, we only judge whether the counting result is larger than k and adopt a secure comparison instead. As long as *thres* is between the k^{th} and the $(k+1)^{th}$ nearest distance, we use a secure set union to retrieve the objects within *thres* distance, which is exactly the k nearest objects. The following example illustrates how a federated kNN query is decomposed into plaintext and secure operators and executed.

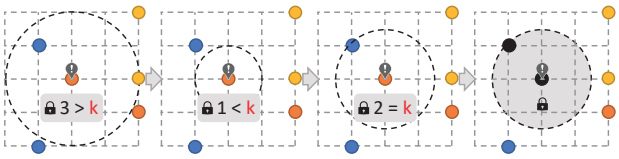

Figure 2: Example of federated kNN query.

Example 1. Consider a federated kNN query with query point $(2, 2)$ and $k = 2$ over 3 data owners in Fig. 2. The objects marked with the same color belong to the same owner. The query rewriter decomposes the query into multiple rounds of radius-known queries. In the first round, a plaintext range counting with center $(2, 2)$ and radius 2 is sent to each owner and a secure comparison with k is performed across owners. Then we get 3 objects, which is greater than k. So in the second round, the radius decreases to 1 and is resent to owners for plaintext range counting and secure comparison. Then we get 1 objects, which is smaller than k. Thus in the third round, the radius increases to 1.5, where the range counting result equals to k and the search terminates. Finally, a plaintext range query with center $(2, 2)$ and radius 1.5 plus a secure set union are performed to get the 2 objects.

5 DEMONSTRATION

The demonstration consists of two parts: *(i)* the deployment of Hu-Fu and *(ii)* its application in cross-company taxi-calling, a common smart city application [7]. We demonstrate Hu-Fu to attendees from both the data owner and the query user perspectives.

5.1 Hu-Fu Deployment

We provide a web application startup script for both the owner and user sides. *(i)* Data owners need to pass in some parameters (*e.g.*, database type and connection parameters) to connect to the underlying database at initialization. Each owner can connect to different types of databases by specifying the corresponding database type. Then the data owner can view local data in the data owner page of

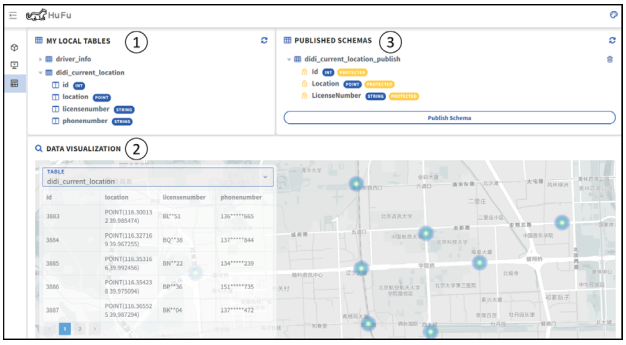

Figure 3: Owner side of Hu-Fu.

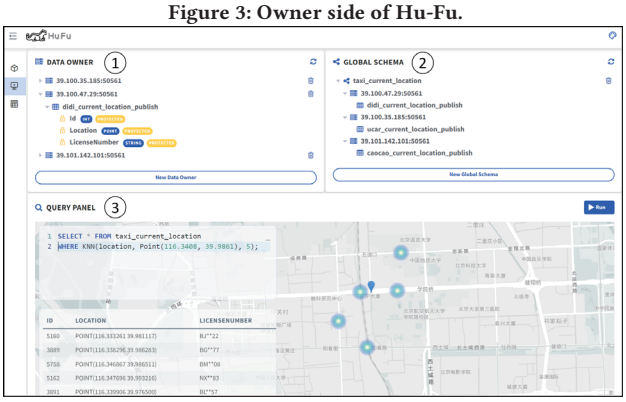

Figure 4: User side of Hu-Fu.

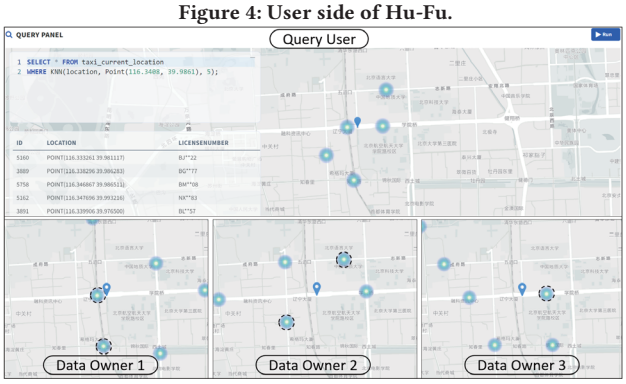

Figure 5: Example of federated spatial query.

Hu-Fu web application (Fig. 3). *(ii)* A query user can directly launch the script and explore the user side functionality in the query user page of Hu-Fu web application (Fig. 4).

5.2 Use Case: Cross-Company Taxi-Calling

The demonstration uses cross-company taxi-calling to showcase Hu-Fu. In this application, multiple taxi companies act as data owners, and passengers are query users. The attendees can explore the functionality of Hu-Fu from both the owner and user sides.

5.2.1 Data Owner Perspective. A data owner (*e.g.*, a taxi company) uses Hu-Fu owner side to view local tables and publish local table schemas to Hu-Fu for federated spatial queries. As illustrated in Fig. 3, the owner side presents the local table schemas in panel ①, visualizes the spatial data of local tables in panel ②, and supports publishing local table schemas to Hu-Fu in panel ③.

In Fig. 3, the data owner has published the schema of table `didi_current_location` as `didi_current_location_publish` to Hu-Fu in panel ③. Attributes `id`, `location`, and `licensenumber` in panel ① are renamed to `Id`, `Location`, and `LicenseNumber` and have security constraints, while the attribute `phonenumber` is hidden. Under the setting, query users can only perform spatial queries on these three columns of the table through secure multi-party computation. All other columns (*e.g.*, `phonenumber`) and other tables (*e.g.*, `dirver_info`) are inaccessible to query users.

5.2.2 Query User Perspective. From the query user's perspective (Fig. 4), the user can connect to multiple taxi companies' owner side in panel ①, construct global schemas by combining multiple schemas from different companies in panel ②, and issue federated spatial queries over these global schemas in panel ③.

As displayed in Fig. 4, the passenger has connected to three companies in panel ①, where each company is represented by its IP address and the passenger can view the table schemas published by these companies. The passenger has constructed a global schema `taxi_current_location` by combining current location table schemas from these companies in panel ②. To get the nearby taxis, the passenger has entered a federated kNN query in the panel ③, and Hu-Fu has returned the nearest 5 taxis and visualized them on the map. Fig. 5 illustrates the query from both the owner and the user perspectives. The final results of the federated kNN query come from the three data owners' local data, which are marked with dotted circles.

6 ACKNOWLEDGEMENTS

We are grateful to anonymous reviewers for their constructive comments. This work is partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant No. U21A20516, 62192784, U1811463, 62076017, the State Key Laboratory of Software Development Environment Open Funding No. SKLSDE-2020ZX-07, and the Lee Kong Chian Fellowship awarded to Zimu Zhou by Singapore Management University. Yongxin Tong is the corresponding author.

REFERENCES

[1] 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.
[2] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. 2018. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. In *SIGMOD*. 221–230.
[3] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS*. 192–206.
[4] Fatih Emekçi, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. 2007. Privacy preserving decision tree learning over multiple parties. *Data & Knowledge Engineering* 63, 2 (2007), 348–361.
[5] Pawel Jurczyk and Li Xiong. 2011. Information Sharing across Private Databases: Secure Union Revisited. In *SocialCom/PASSAT*. 996–1003.
[6] Amit P. Sheth and James A. Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys* 22, 3 (1990), 183–236.
[7] Numerous Beijing Taxi Brands to Collectively Connect to Amap's Ride-hailing Platform to Enable Online Operation. 2021. https://aag.cc/newsinfo/517126.html
[8] Yongxin Tong, Xuchen Pan, Yuxiang Zeng, Yexuan Shi, Chunbo Xue, Zimu Zhou, Xiaofei Zhang, Lei Chen, Yi Xu, Ke Xu, and Weifeng Lv. 2022. Hu-Fu: Efficient and Secure Spatial Queries over Data Federation. *PVLDB* 15, 6 (2022), 1159–1172.
[9] Paul Voigt and Axel Von dem Bussche. 2017. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Vol. 10. Springer International Publishing.
[10] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In *EuroSys*. 3:1–3:18.