Federated Database System for Scientific Data

Sangchul Kim Bongki Moon

Department of Computer Science and Engineering
Seoul National University
Seoul 08826, Korea
{stdio,bkmoon}@snu.ac.kr

ABSTRACT

Much like traditional databases, scientific data are managed in multiple separate databases by different sources and organizations. When such distributed data are analyzed together for more comprehensive understanding and prediction, it is necessary to access data via multiple simultaneous connections or collected in a single location. The inevitable consequence is, however, that a significant overhead is incurred due to differences in schemas, data transformation, and extraneous cost for storing intermediate data. This demo presents *SDF*, *Scientific Database in Federation*, which facilitates data sharing and exchange in order to support complex analytics with minimal integration overhead. *SDF* is currently implemented in *SciDB* using user-defined operators, providing two connection models, *master-to-master* and *cluster-to-master*, for a shared-nothing architecture.

CCS CONCEPTS

• **Information systems → Federated databases**;

KEYWORDS

Scientific Data; Federated Database System; SciDB

1 INTRODUCTION

High precision and resolution sensors generate diverse types and a high volume of data for accurate and sophisticated analysis. In order to manage an increasing volume of data in various types effectively, data are often organized in multiple databases which can have different ownership and properties. Data can be partitioned and distributed geographically in multiple databases across separate servers, considering their features and purposes of organizations.

When distributed data are analyzed together, it is necessary to access such data via multiple simultaneous connections or collected in a single location. If databases are not able to be interconnected with each other, users themselves have to manually establish the connections and integrate data in their application. Also, to collect data in a single location, users should carry out heavy work which includes downloading and loading data. Since it takes a significantly long time to load raw data with those many cumbersome steps, the work is inefficient and inconvenient. Furthermore, users may encounter a challenge when the schema for data may not be identical. Even with a global schema, translating schema is required to integrate data, which incurs extraneous costs. Hence, it is not easy to find a simple solution that offers a remedy for complex queries or analysis using distributed data in multiple databases.

In this demo, we present *SDF*, *Scientific Database in Federation*, which facilitates data sharing and exchange to process complex queries with minimal integration overhead. *SDF* is currently implemented in *SciDB* [1] by user-defined operators (UDO). SciDB is based on a multi-dimensional array model that is more appropriate to manage scientific data for data mining or an arithmetic computation (*e.g.*, matrix computation) than the relational model [2]. UDO is a plug-in provided by SciDB and helps *SDF* federate multiple databases with relatively simple installation. It is aimed at using SciDB query languages (both AQL and AFL) to allow *SDF* access remote databases at the query level. Unlike BigDAWG [3], *SDF* is not a middleware solution to federate databases.

For scientific analysis, users pose a query via the SciDB client interface. However, SciDB does not support complex analytics using data stored in separate databases since it is only able to access a single database. Adopting the principle of a federated database system, *SDF* abstracts away details about the process of combining and integrating databases, which considerably appeals to scientists who need to pose sophisticated analytics but do not have expertise in database systems. For instance, scientists need not declare a global schema in advance for federated query processing. Instead, a schema in a remote database is generated during the query execution time. *SDF* accepts the primary ownership of systems (*i.e.*, autonomy) and gives a federated view as a single database from multiple databases. Moreover, *SDF* preserves unique system features including numerous analytics libraries while accessing remote databases. The main advantage is the ability to combine data from multiple databases in a single query statement and cooperate separate databases together for complex analysis.

Since SciDB has a shared-nothing architecture, *SDF* provides two connection models, *master-to-master* and *cluster-to-master*, to establish a connection between SciDB clusters. Especially, in the cluster-to-master model, a remote query is converted into localized queries, which leverages the performance such as an aggregate query. *SDF* allows SciDB to operate within different back-end storage or clusters. It helps perform complex analytics and aggregate data, as granting a data warehouse to be used without any modification.

SSDBM '18, July 9–11, 2018, Bozen-Bolzano, Italy
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6505-5/18/07…$15.00
https://doi.org/10.1145/3221269.3222332

This work was partly supported by a grant (K-16-L03-C01-S03) funded by the ministry of science, ICT, and future planning, Korea and PF Class Heterogeneous High Performance Computer Development (NRF-2016M3C4A7952587).

Section 2 describes the background knowledge of the federated database system. In Section 3, we present the system architecture of *SDF* and federated query processing. Section 4 describes our demonstration scenarios. Finally, we conclude our work in Section 5.

2 FEDERATED DATABASE SYSTEM

A federated database system [6] virtually maps multiple databases into a single database for data sharing and exchange. The system is composed of disparate databases that can be described as virtual databases since it migrates physically or geographically distributed databases into a single database interconnected via networks. For operating tasks with (heterogeneous) data, a federated schema is essential to use or access data belonging to a remote database. The principle of the system has been adopted in several popular open-source database systems [4, 5].

The federated database system is able to abstract away integrating processes, enabling access to a remote database without tuning database system engines. By this abstraction, this system can provide an interface for standard queries to store and retrieve data. Since a result can be composed of sub-query results, the system should process a decomposable query to submit it to federated DBMSes. Given that the federated database system consists of several various systems that use different query languages, it needs an intermediate translator to translate the different languages into their proper languages or vice versa.

The federated database system maintains the authority of autonomous systems, each of which can cooperate with the high-level integration. It accepts the primary ownership of each system rather than uses centralized control. That is, it does not require any global schemas for all local schemas and does not dedicate itself to access databases. It enables to access data in remote databases using a global interface as a local database and need not reconstruct the appropriate information whenever the access is requested. The architecture of this system is suitable for taking self-sufficient and stand-alone DBMSes while maintaining the autonomy of them. The access to a remote database is possible without altering functions of an application and impacting on other applications.

3 DESIGN OF SDF

Basically, data sharing and exchange with disparate databases are infeasible. To access a remote database, a DBMS should support integrating operations with minimizing overhead. We adopt the principle of a federated database system to provide a view as a single database from multiple databases. We add the novel feature to SciDB, called *SDF*, *Scientific Database in Federation*. *SDF* federates multiple databases by decomposing a federated query into sub-queries and posing a remote query to a remote SciDB. This federated query follows the process of a regular query. When posing a query to a local SciDB, users utilize array information such as array and attribute names. Similarly, to pose a federated query, remote array information with a remote database name which indicates a target remote SciDB is used.

There are two benefits of *SDF*. First, a remote query statement can be simpler. The vanilla SciDB would demand the information of a remote database to establish a connection. In *SDF*, however, once the name of a remote database is registered, its information is provided. Second, it is convenient to access distributed data simultaneously by a single query. For example, suppose in the example given below, `cloud` is a local array and `seawifs` is in a `remote` database, the following query

join(cloud, connector('remote', 'seawifs'))

is able to be processed. The *connector* is a key operator of *SDF* and enables a local SciDB to access remote databases. Since the abstraction of federated processing makes it unnecessary to modify a system engine every time different databases are federated, *SDF* can provide a scalable database view.

3.1 System Architecture

For the federated query processing, a query is decomposed into sub-queries. For example, the above example query is decomposed into *connector* and *join*. After receiving a remote query from a local SciDB, the remote SciDB processes the query and sends a *foreign array* to a local SciDB. The foreign array is referred to as a result of a remote query processed by a remote SciDB. Then, the local SciDB maps the foreign array to a local array and processes the decomposed query (*e.g.*, *join* in above example). The architecture of *SDF* is in Figure 1, and we describe seven modules as follows.

Modules

SDF has seven modules to process a federated query, implemented by user-defined operators (UDO). It is relatively easy to add new algorithms or operations to SciDB using UDO. Compared with developing a new feature in a database engine, installation of UDO is not required to re-build SciDB.

Address Manager manages information of remote databases such as an IP address and port number. The information is stored in PostgreSQL because it is suitable for frequent inserts and updates rather than SciDB. A SciDB cluster consists of several *instance*s that are independent workers for query processing, but only the master instance has this module.

Connector establishes a connection between local and remote SciDB clusters. The connection should be maintained until the transmission of a foreign array is complete.

Executor sends a query to a remote SciDB via *Connector*. If an array does not exist or the syntax of a remote query is incorrect, the remote SciDB returns error messages. Otherwise, the query is processed, and the remote SciDB returns a schema and a foreign array. While *Executor* submits a remote query, additional parameters informing a query type (requesting either a schema or an array) piggyback on the query. If an array is requested, the query brings parameters such as an instance ID and the number of instances.

Schema Manager receives the schema of a foreign array from a remote SciDB, which makes it unnecessary to declare the schema ahead of time. Especially, the schema of an original array can be transformed if a query (*e.g.*, aggregate) create a new array different from an original array. The schema is used to create a virtual array in a local database.

Receiver receives a foreign array from a remote database. The transmission unit can be either a cell or a chunk. A cell contains one or more attributes, and a chunk is a group of cells and a physical unit of I/O.

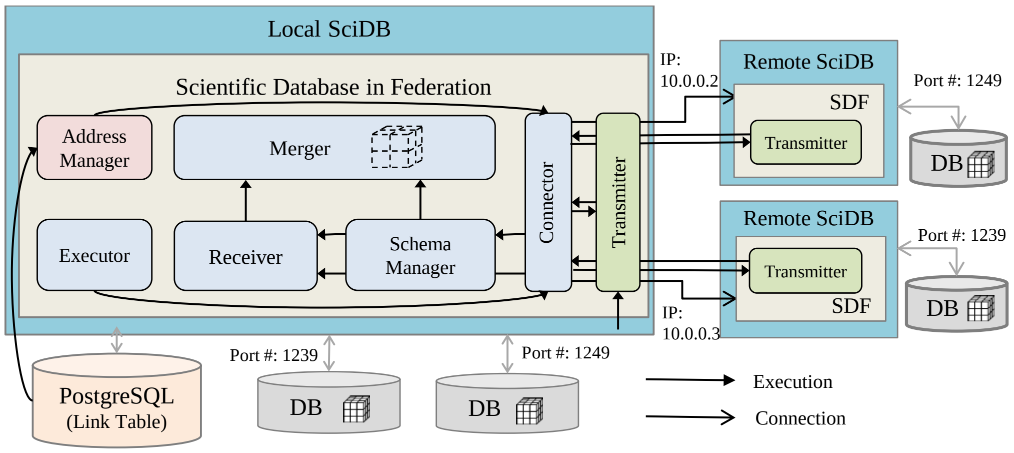

Figure 1: *SDF* architecture: local and remote SciDB clusters are connected for federated query processing. Each SciDB can be connected using IP addresses, and databases are distinguished by their unique port number.

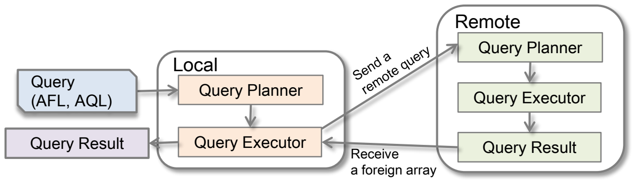

Figure 2: Processing sequence to process a federated query.

Merger produces a virtual array using a schema from *Schema Manager* and an array from *Receiver*. While receiving a foreign array, *Receiver* directly passes the received chunks to *Merger* to create a virtual array in a local database.

Transmitter sends a schema or a foreign array to a local SciDB. A local SciDB should also use *Transmitter* when sharing arrays between local databases. In a cluster-to-master model described in Section 3.3, *Transmitter* filters out cells, not involved in a foreign array using chunk ID, and sends a foreign array to instances corresponding to instance ID.

3.2 Federated Query Processing

There are a few steps for message exchange to process a federated query. To access a remote database, a local SciDB must create a link to a remote SciDB. As an IP address is required to establish a connection to a remote SciDB and port numbers distinguish multiple databases, any user who attempts to access other databases should obtain them in advance. *SDF* provides an alias (*i.e.*, a database name) to retrieve the information from PostgreSQL where SciDB stores its metadata.

After the connection is established, a local SciDB can access a remote database. Since the local SciDB does not have the schema of a foreign array, it requests the schema before receiving the foreign array. As the schema is received in the query execution time, it need not be declared in advance; thus, the local SciDB can avoid translating a result to a local schema. Then, local SciDB submits a remote query to a remote SciDB as Figure 2. After entirely receiving the foreign array, the local SciDB sends a message that the whole process has completely executed. Then, the connection between a local SciDB and remote SciDB is closed.

3.3 Connection Model

Since SciDB runs in a cluster environment, establishing a connection between clusters should be provided. *SDF* provides two connection models, *master-to-master* and *cluster-to-master*, for shared-nothing architecture. A cluster-to-master model has a more complicated process but better performance than a master-to-master model.

A **master-to-master** model is that a master of a local SciDB connects with a master of a remote SciDB. As all modules are in a local master, a local master sends a remote query to a remote master and receives a foreign array alone. Then, the master distributes data corresponding instance ID. A federated query can be processed without setting the equivalent number of instances of federated SciDB clusters because the connection is only established between two masters.

In contrast to the master-to-master model, a **cluster-to-master** model can directly distribute a foreign array to local instances. This model also does not require the identical number of instances between local and remote SciDB clusters. In order to process a federated query, *Address Manager* retrieves the information of a remote SciDB and its database, the same as the process of master-to-master model. Afterward, it broadcasts the information to local SciDB instances. When they receive it, *Connector* connects all instances to a remote master. Then, the instances transform the query to localized queries and pose them to a remote master. Internally, the number of instances and each instance ID piggyback on a localized query.

In a remote SciDB, while queries are processed, *Transmitter* prepares sub-arrays to send them to corresponding instances. It uses a foreign array schema to calculate the hash value which determines the target instance of each chunk. It is critical when an aggregate query is processed. With the query, the schema may be re-created, which leads to changing not only hash values but also a subset of cells. This subset is used to create a sub-array for a corresponding

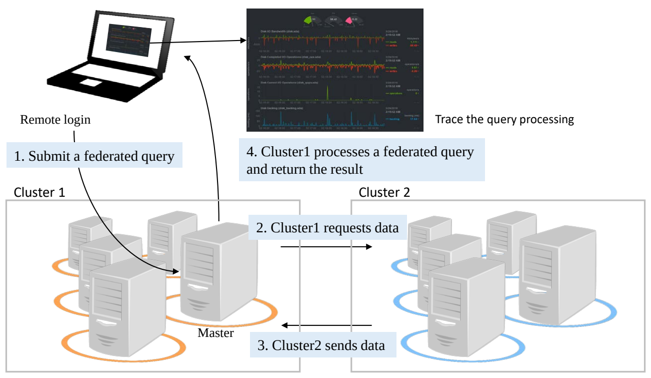
Remote login
1. Submit a federated query
4. Cluster1 processes a federated query and return the result
Trace the query processing
Cluster 1
Cluster 2
2. Cluster1 requests data
3. Cluster2 sends data
Master

Figure 3: The scenario of the demonstration

instance. If the value is computed from an initial schema, the result must be incorrect. That is, *Transmitter* sifts cells to send data to target instances correctly.

4 DEMONSTRATION SCENARIO

We demonstrate *SDF* that federates multiple databases and provides an integrated view. In our showcase, attendees can submit federated queries, either our example or specified queries by themselves.

Environment. The cluster consists of ten nodes, five nodes for a local SciDB cluster and the other five nodes for a remote SciDB cluster. Each node has Intel i7-4770 3.40GHz CPU and 16GB RAM. Linux kernel version is 3.16.0, and SciDB version is 14.12.

Datasets. We will use SeaWiFS L3 Chlorophyll (in short, CHL) and MODIS remote-sensing reflectance (in short, RRS). The data have three dimensions (*longitude, latitude, time*) and one attribute (*chlorophyll*). Each cell is an 8-day average of *chlorophyll* of the earth with the resolution of 9km × 9km (CHL) and 4km × 4km (RRS). The chunk size is 1000 (*longitude*) × 1000 (*latitude*) × 1 (*time*), one million cells in a chunk. We will use two types of RRS for experiments: RRS (412), RRS (443). These two have the same dimensionality, but different spectral bands.

Scenarios. Our first scenario is the comparison between the vanilla SciDB and *SDF*. Figure 3 shows our demonstration scenario. Attendees access our cluster via SSH, connecting the master node (SciDB coordinator). They can submit federated queries using *iquery*. According to our scenario, Cluster1 requests data to Cluster2 by submitting a remote query to Cluster2 in which requested data are involved. Cluster2 processes the query and sends a foreign array to Cluster1. Then, SciDB in Cluster1 generates a *virtual* array and then returns the result to users. As users cannot see these processes, we will trace the query processing and visualize logs in real-time.

Table 1 is the preliminary result that will also be presented in our showcase. It shows the query processing time of *SDF* and the vanilla SciDB. Since the vanilla SciDB has a limit to process queries which require accessing distributed data in different databases simultaneously, we used two queries, *scan* and *aggregate*, that access a database. A scan query reads all cells, and an aggregate query computes an average of *chlorophyll* over the period and groups

Query	Scan	Aggregate
Vanilla SciDB	467	6,192
SDF	413	3,380

Table 1: The processing time (second) of two queries.

the result by longitude and latitude. In this evaluation, we used the cluster-to-master model of *SDF*. *SDF* is slightly faster than the vanilla SciDB using the scan query because, in the vanilla SciDB, the master of a local SciDB receives an entire array alone. An increase of the data size also grows the query processing time linearly. Also, *SDF* achieves the 1.8x speedup on the aggregate query. Network overhead for sending a foreign array rarely affects the processing time because aggregation reduces the data size. In this query, the size is reduced by 0.2% of the original data size. Rather, loading the entire array into memory and computing for aggregation affect its performance, accounting for most of the query processing time. Local SciDB instances pose localized queries, which makes concurrent computation in a remote SciDB.

The second scenario is that we will describe two connection models. As is described in Section 3.3, the performance is different because of their distinguished mechanisms. To evaluate them, we will use two queries, *merge* and *join*. The queries access and concatenate two arrays. A join operator concatenates attributes by each dimension, but a merge operator concatenates dimensions of two arrays. Without *SDF*, two queries are infeasible to be processed if two arrays are stored in separate databases.

5 CONCLUSION

In order to process complicated analysis using data in separate databases, database federation is necessary. In this demo, we present *SDF*, built in SciDB, which adopts the principle of a federated database system for a shared-nothing architecture using UDO. *SDF* abstracts away integrating processes, preserving system features and the primary authority of each SciDB; it facilitates the data sharing and exchange within multiple databases, providing an interconnect interface. It will be one of the big data solutions for numerous consolidated resources and help enhance scientific analytics.

REFERENCES

[1] Paul G Brown. 2010. Overview of SciDB: Large Scale Array Storage, Processing and Analysis. In *Proceedings of the 2010 ACM SIGMOD Conference*. ACM, Indianapolis, indiana, USA, 963–968.
[2] Philippe Cudre-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Samuel Madden, Michael Stonebraker, Stanley B Zdonik, and Paul G Brown. 2010. SS-DB: A Standard Science DBMS Benchmark. Extremely Large Databases Conference 2010.
[3] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The BigDAWG Polystore System. *ACM Sigmod Record* 44, 2 (2015), 11–16.
[4] MySQL. 2017. Federated Storage Engine of MySQL. https://dev.mysql.com/-doc/refman/5.7/en/federated-storage-engine.html. (2017).
[5] PostgreSQL. 2017. PostgreSQL 9.6.1 Documentation. https://www.postgresql.org/-docs/9.6/static/postgres-fdw.html. (2017).
[6] Amit P Sheth and James A Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys (CSUR)* 22, 3 (1990), 183–236.