Enabling Sovereign Information Sharing Using Web Services

Rakesh Agrawal Dmitri Asonov Ramakrishnan Srikant

IBM Almaden Research Center, San Jose, CA 95120

ABSTRACT

Sovereign information sharing allows autonomous entities to compute queries across their databases in such a way that nothing apart from the result is revealed. We describe an implementation of this model using web services infrastructure. Each site participating in sovereign sharing offers a data service that allows database operations to be applied on the tables they own. Of particular interest is the provision for binary operations such as relational joins. Applications are developed by combining these data services. We present performance measurements that show the promise of a new breed of practical applications based on the paradigm of sovereign information integration.

1. INTRODUCTION

Current information integration approaches, as exemplified by centralized data warehouses and mediator-based data federations, are based on the assumption that the data in each database can be revealed completely to the other databases. A new model, which we shall henceforth refer to as *sovereign information sharing*, was proposed in [8] to fulfill the requirements for integrating information across autonomous entities. In this model, the execution of a query over the union of relations from two different databases does not reveal any information apart from the result of the query to either entity.[1] Applications include information exchange between security agencies, intellectual property licensing, crime prevention, and medical research.

We describe a practical realization of sovereign information sharing on top of the web services infrastructure. Data resides in DB2 v.8.1. database systems, installed on 2.4GHz/ 512MB RAM Intel workstations, connected by a 100Mbit LAN network. Web services run on top of the IBM WebSphere Application Server v.5.0 and the IBM private UDDI registry installed on one of the machines. We use Apache AXIS v.1.1. SOAP library for messaging.

The basic capability provided by sovereign information sharing is a set of crypographic protocols. The algorithmic description of the protocols for set intersection, intersection size, equijoin, and join size is available in [8]. Providing these protocols as web services offered by the autonomous data sources enables the development of novel applications, unencumbered by platform and implementation language considerations.

2. SOVEREIGN INFORMATION SHARING AS A GRID OF DATA SERVICES

In our architecture, each participating site offers two web services: one receives requests to execute the specified protocol with the specified remote party and another serves the requests of remote parties to engage in a protocol. For example, Figure 1 shows the computation of the intersection size of tables R and S belonging to parties A and B respectively.

2.1 Binary Operations over Web Services

The current tools for developing database-oriented web services [2, 5, 6] have been designed for exposing operations on a database belonging to a single party. For example, the Web Services Object Runtime Framework for DB2 (WORF) [5] allows predefined queries or stored procedures to be published as web services. Query parameters are passed from the client application to the web service as part of the request for service. Figure 2a shows a WORF generated web service for a client application. In this example, the client is checking the existence of a specific SSN in the remote database.

However, these tools are not designed to create web services that can take local tables as arguments. They are inadequate for developing data services involving operations spanning over tables of two or more parties, which we refer to as binary operations.

Being able to perform such operations is necessary for implementing the sovereign information sharing protocols.[2] Figure 2b shows an example, where a client application can intersect its table with a remote table by providing the local table as an argument to the remote web service.

[1]For example, suppose the entity A has a set $R = \{b, u, v, y\}$ and the entity B has a set $S = \{a, u, v, x\}$. As the result of sovereign intersection $|R \cap S|$, A and B will get to know the result $\{u, v\}$, but A will not know that B also has $\{a, x\}$, and B will not know that A also has $\{b, y\}$.

[2]There is extensive distributed database literature [9, 10] on execution strategies for implementing operations across databases distributed over multiple parties. It will be interesting to explore how this rich body of work can be applied to web services.

SIGMOD 2004 , June 13-18, 2004, Paris, France.

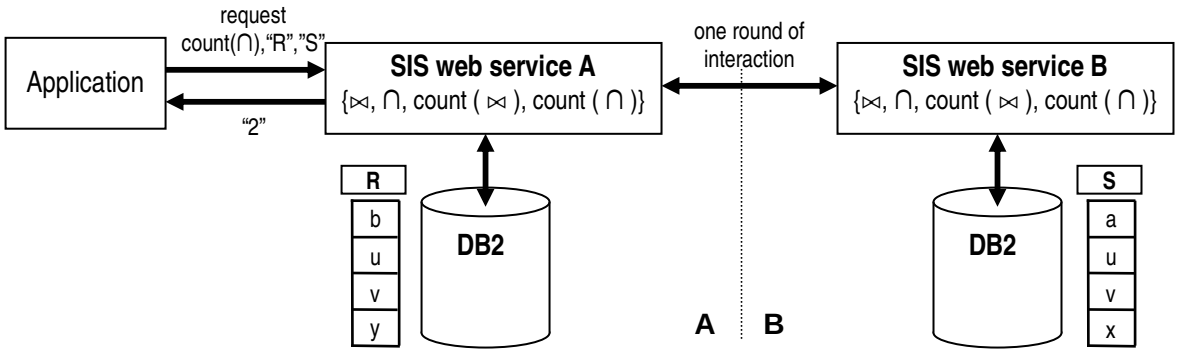

Figure 1: Sovereign Information Sharing (SIS) architecture.

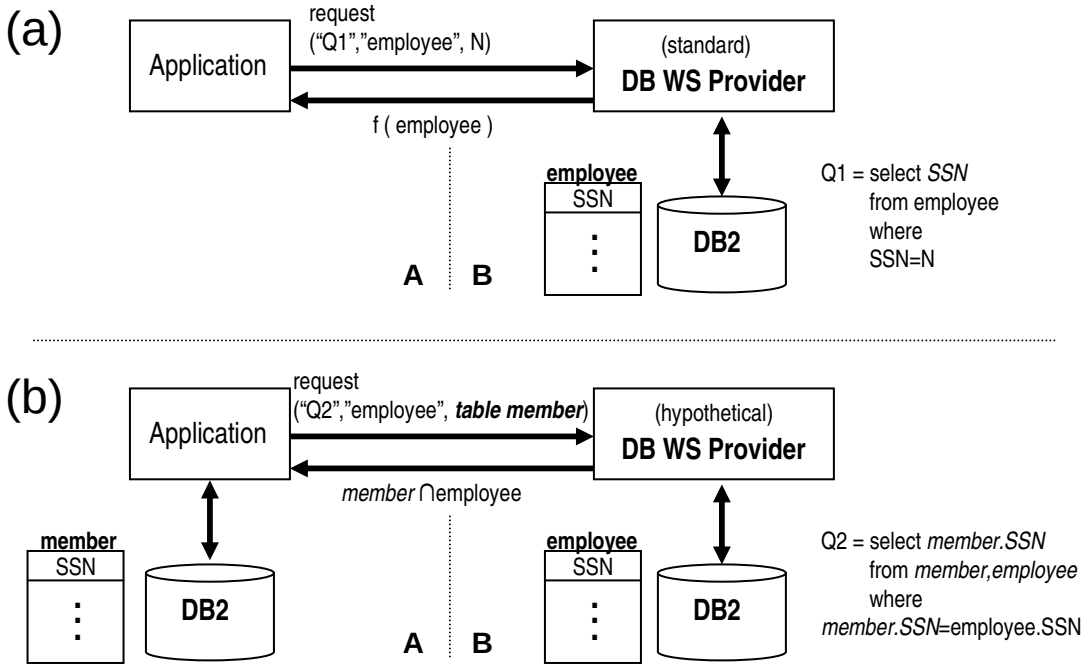

Figure 2: (a) A web service utilizing only a local table (unary operation). (b) A hypothetical web service taking a remote table as an argument (binary operation).

2.2 Providing Tables as Arguments

Unlike scalar values, there is no established standard for encoding relational tables in a SOAP message. We circumvent the problem by employing the attachment facility in the SOAP specification [1]. This option allows a client applications to attach any data (in our case, an exported local table after applying necessary local predicates) to SOAP message and let the web service fielding the request extract this data. The web service imports the table into the local database before calling the query.

2.3 Implementing Sovereign Protocols

For concreteness, we will discuss the implementation of the intersection size protocol. It will be easy to see how the implementation ideas extend directly to other operations.

Assume for simplicity that the tables R and S, belonging to autonomous parties A and B, have one column each. The following is the essence of the intersection size protocol (see [8] for details), wherein party A learns nothing about S except the intersection size $|R \cap S|$ (and $|S|$), and B learns nothing about R (except $|R|$):

1. A encrypts table R into $E_a(R)$ using the key a, and sends it to B.
2. B encrypts table S into $E_b(S)$ using the key b. It also encrypts $E_a(R)$, received from A, into $E_b(E_a(R))$.
3. B sends tables $E_b(S)$ and $E_b(E_a(R))$ to A.
4. A decrypts $E_b(E_a(R))$ into $E_b(R)$.
5. A computes $|E_b(R) \cap E_b(S)|$, which equals $|R \cap S|$.

Note that we use a commutative encryption in the above protocol.

Figure 3 shows how this protocol is implemented. Party B has advertised that it is willing to intersect its table S

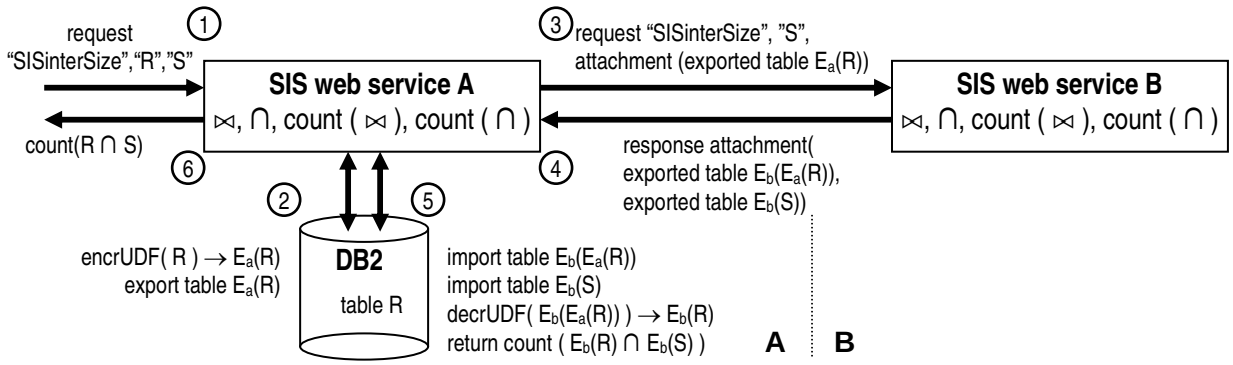

Figure 3: Implementation of the sovereign intersection protocol.

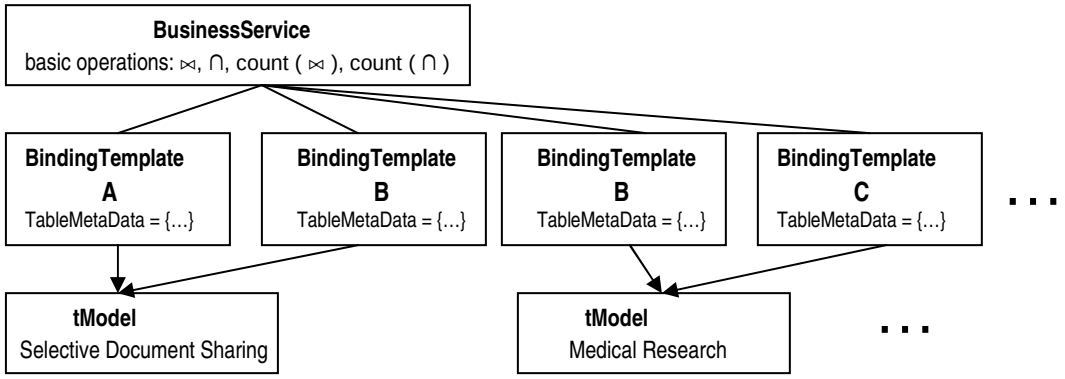

Figure 4: A UDDI registry is employed to allow for dynamic service discovery.

with authorized parties.[3] The following sequence of steps takes place (Figure 3):

1. An application wishing to compute the size of the intersection of R and S (and that has proper authorizations to do so) issues the request to the intersection size service at A.
2. The web service at A encrypts each value in the table R.
3. It sends the intersection size request to the web service at B, with the table of encrypted values as an attachment.
4. After B has finished its part of the protocol, the web service at A receives response from the web service at B, with two encrypted tables attached.
5. The web service at A removes its encryption from the values in table R received from the remote web service.
6. Finally, the web service at A determines the intersection size of tables R and S and returns this result to the application.

2.4 Resource Discovery

Before calling a web service, an application must be able to:

- find a party that offers the required web service (if the party is not known at build time);
- determine the arguments (the names of the tables, columns, and necessary predicates) that must be provided to obtain the desired result.

We employ a UDDI registry [7] for this purpose. A UDDI registry offers a way of storing and searching cataloged descriptions of web services. The UDDI standard categorizes the types of the entities stored in a registry into "BusinessService", "BindingTemplate", and "tModel". We use them as follows (Figure 4):

- "BusinessService" represents a SIS web service, having a set of functions corresponding to the the supported SIS operations, such as intersection size.
- Each instance of the type "tModel" represents a sequence of operations that constitute a specific application model. Since SIS operations (BusinessService) can be used for many purposes, several tModels can be linked to a BusinessService.
- Each instance of the type "BindingTemplate" links a BusinessService and a tModel, and is specific to the party. It provides an endpoint access address of the instance of the BusinessService of that party, and stores the required metadata (including a set of table names and possible predicates) that are available for the given tModel at the given party.

[3]We assume that the authentication and authorization are done using the standard facilities [3].

Exponentiation Implementation	msec
Standard Java (BigInteger package)	32
MS Visual C++ (Crypto++ library)	65
DB2 UDF using standard Java (BigInteger)	33

Table 1: Exponentiation implementations (1024 bit).

Figure 4 gives an example of the three parties A, B, and C, where A and B support selective document sharing application (tModel), and B and C support medical research application.

3. EVALUATION

This section presents performance measurements from our implementation of the sovereign protocols.

3.1 Performance

Sovereign protocols require some communication between the two parties, some number of database records to be encrypted and decrypted, and a few standard database operations to be applied (e.g. step 5 of the intersection protocol given in Section 2.3). Both computation and communication complexities are linear in number of data records [8], which is borne out by the experimental results.

Figure 5 shows the execution times for the four operations. We have broken the execution time into the components mentioned above. The performance numbers are presented for tables having 10,000, 50,000, and 100,000 records each.

The cost of the protocols may appear high to the reader. We would like to emphasize that although we are using the familiar names, the functionality provided by, say the sovereign join, is quite different from the standard relational join. Comparing the execution times of the sovereign operations with the standard relational operations is, therefore, inappropriate. The test we must apply is whether the performance numbers are within the operating range for creating useful applications. We will describe two applications we are building using these protocols where the answer is in affirmative.

Figure 5 shows that the encryption is the dominant cost. We, therefore, further study the cost of encryption.

3.2 Encryption

The encryption and decryption functions used in our protocols are based on exponentiation. For an input x, a modulus p, and encryption and decryption keys e, d (all four numbers are 1024-bit long), the encryption and decryption functions are as follows [8]:

$$encr(x, e, p) = x^e \bmod p; \quad decr(x, d, p) = x^d \bmod p.$$

We employ the user defined function (UDF) facility in DB2 to implement encryption and decryption of column values, which allows us to perform these operations within the database, saving the cost of moving the data from the database to application level and back. UDFs also allow for parallel execution of encryption over multiple processors, although our measurements do not take advantage of this feature.

Table 1 gives the performance comparison of the following implementations of exponentiation:

- MS C++ implementation (Crypto++ library),
- Java native (BigInteger) implementation,
- DB2 UDF implementation based on standard Java (BigInteger).

Surprisingly, the Java implementation of exponentiation outperforms the Visual C++ implementation. This superiority could be due to different exponentiation algorithms used in Java and C++ libraries. The algorithm used is a crucial determinant of performance because different algorithms make different number of calls to the modular multiplication of large numbers.

Note that encrypting records inside the database is competitive compared to the exponentiation with Java in the main memory. The reason again is that most of the encryption time is spent by the processor doing multiplication of large numbers.

3.3 Making Encryption Faster

Software Approaches We tried some custom implementations of exponentiation that used preprocessing based on the fixed exponent (e) or the fixed base (x) to reduce cost. Unfortunately, the fixed-exponent implementation turned out to be slower than the Java native implementation of exponentiation.

The fixed-base implementation optimizes the computation for the case when the same value is encrypted multiple times with different keys. Therefore, it is not useful for such applications as list intersection, where each value is encrypted only once. However, for other applications such as document sharing, the encryption is performed multiple times for the same input table. In such cases, the fixed-base implementation of the exponentiation can be beneficial.

Hardware Approaches One possible way of increasing the encryption speed would be to use crypto accelerators, such as COTS accelerators for SSL [4]. These cards are fairly inexpensive (about US $2000). Based on the specifications, we expect them to speed up the exponentiation (and hence the cost of protocol) by at least an order of magnitude.

The other source for performance enhancement would be parallelism, as a record can be encrypted/decrypted independent of other records in the database. We therefore can get linear speedup.

4. APPLICATIONS

To test the effectiveness of our infrastructure we are building two applications.

The first application is *list intersection* that organizations may employ to find which items in their lists are the same, without disclosing the rest of the lists. For instance, security agencies can determine the suspects they have in common without disclosing the names of other suspects. This application is easy to implement on top of our infrastructure. We need a single call to the intersection service.

Selective document sharing is the second application. It uses a sequence of calls to the intersection size service to only find those documents for which the similarity score is high enough to warrant their exchange between the two organizations. Specifically, this application calls the intersection size service for each pair of documents d_A, d_B (one from the local database and one from the remote database) to calculate the similarity between them:

$$f(|d_A \cap d_B|, |d_A|, |d_B|) = |d_A \cap d_B|/(|d_A| + |d_B|)$$

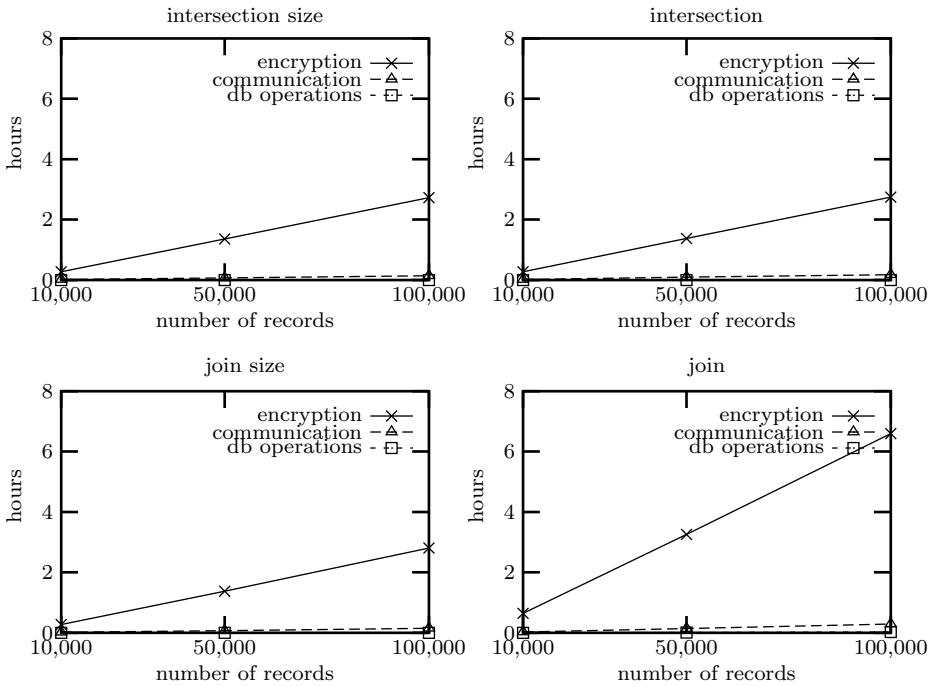

Figure 5: Execution times ($|R| = |S|$ = number of records).

5. SUMMARY

We presented a web services based infrastructure for enabling sovereign information sharing. This infrastructure uses a grid of data services to perform database operations, while ensuring that no information apart from the query result is revealed.

The performance results, while demonstrating the feasibility of using this paradigm for developing practical applications, also provide motivation for further research. The possibilities range from the use of hardware accelerators for speeding exponentiation to developing computationally less expensive encryption to developing new protocols for richer query processing.

Acknowledgements. We wish to thank Roberto Bayardo, Marcus Fontoura, Jussi Myllymaki, Inderpal Narang, Vijayshankar Raman, Berthold Reinwald, and Amit Somani for insightful discussions.

6. REFERENCES

[1] SOAP Messages with Attachments. `www.w3.org/TR/SOAP-attachments`, Dec. 2000.

[2] Database Web Services: An Oracle White Paper. `http://otn.oracle.com/tech/webservices/htdocs/dbwebservices/Database_We%b_Services.pdf`, Nov. 2002.

[3] Security in a Web Services World: A Proposed Architecture and Roadmap. A joint white paper from IBM Corp. and Microsoft Corp. `http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/`, Apr. 2002.

[4] SSL Accelerators, Interactive Buyer's Guide. `http://ibg.networkcomputing.com/ibg/Guide?guide_id=4065`, Nov. 2002.

[5] Web Services Object Runtime Framework for DB2: Implementing DB2 Web Services. `http://www7b.software.ibm.com/dmdd/zones/webservices/worf/DXXSERVL.PDF`, Sept. 2002.

[6] SQL Server 2000 Web Services Toolkit. `http://www.microsoft.com/sql/techinfo/xml/default.asp`, 2003.

[7] UDDI Spec Technical Committee Specification. `http://uddi.org/pubs/uddi_v3.htm`, Oct. 2003.

[8] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of ACM SIGMOD*, June 2003.

[9] P. Bernstein, N. Goodman, E. Wong, C. Reeve, and J. Rothnie. Query processing in a system for distributed databases (SDD-1). *In ACM Trans. Database Systems*, 6(4), 1981.

[10] B. Lindsay, L. Haas, C. Mohan, P. Wilms, and R. Yost. Computation and Communication in R*: A Distributed Database Manager. *In ACM Trans. Database Systems*, 2(1), 1984.