OSS Data Integration using Virtual Database

Naoki Take, Manabu Nishio, and Hikaru Seshake[1]
NTT Network Service Systems Laboratories
3-9-11, Midori-Cho, Musashino-Shi,
Tokyo, 180-8585 Japan
E-mail: {take.naoki, nishio.manabu, seshake.hikaru[1]}@lab.ntt.co.jp

***Abstract*—Due to network scale expansion and shortening of service lifecycles in recent years, operation support systems (OSSs) have been developed and deployed individually in a short time. However, such OSSs are isolated from each other, which makes it difficult to share data among them. To solve this problem, we propose an OSS data integration approach using a virtual database and discuss its feasibility with respect to functionality and performance. In terms of functionality, we discuss the feasibility of our approach without functions that use event detections. In terms of performance, we argue that overhead will not be a problem in reading and writing data with our approach.**

Keywords—operation support systems; data integration; virtual database;

I. INTRODUCTION

Integration of disparate data that have distributed, autonomous, and heterogeneous data sources, often referred to as data integration [1] or information integration [2], is a crucial topic both commercially and academically. The heterogeneity and dispersiveness of data sources are due to various reasons depending on the situation. One example is a large-scale scientific project where data sets are being produced independently by multiple researchers [1]. Another example is large companies or government agencies that have developed many information systems independently to satisfy the needs of local organizational units [2]. These systems were designed, built, and optimized to solve local needs, so there is little regard for using data throughout the entire enterprise.

There is a similar problem regarding network operation. Due to the rapid expansion of network size and the short lifecycles of services over a network in recent years, operation support systems (OSSs) have been developed and deployed in a short time, resulting in the isolation of each OSS, which makes it difficult to share information among those systems.

As a solution to the integration problems mentioned above, an industry called Enterprise Information Integration (EII) has been growing since the beginning of late 1990's [1]. EII products provide tools for integrating data from multiple data sources without first needing to load all the data into a central storage.

EII products are based on a system called the Federated Database System (FDBS) [3]. An FDBS is "a collection of cooperating database systems that are autonomous and possibly heterogeneous" [3], and is used to integrate multiple distributed databases. In an FDBS, a global view is created over existing databases so that users can treat them as a single database [2]. An FDBS, or simply a federated database, is often called a "virtual database (VDB)" (e.g. [4]), so we use this term in this paper.

We discuss the feasibility of applying data integration based on the VDB technique to OSSs. When considering the application of integration to OSSs, we assume that two application scopes in data integration. The most common one integrates only data located in databases, files, and so on. The other is more advanced and integrates all the data including network elements (NEs) as data sources. We suggest the latter scope of integration because it has a positive effect on OSS development, as discussed in Section 2.

This paper is organized as follows. Section 2 explains the reason we propose integration including NEs. Section 3 describes related approaches and compares them with our proposed approach. Sections 4 and 5 explain and discuss the results of feasibility evaluations with respect to functionality and performance, respectively.

II. SCOPE OF INTEGRATION

When considering the application of data integration to OSSs, we have to consider the most important feature of OSSs; that there are NEs in the system. In this section, we focus on this feature to discuss the application scope in data integration.

A. Scopes of Integration

When we model an OSS application as having its own database and managed NEs as data sources, the current isolated systems are represented as in Fig. 1(a). When applying data integration, we assume that there are two cases according to the scopes of integration. One case integrates only data located in databases (Fig. 1(b)). The other integrates all the data including NEs when assuming NEs as data sources (Fig. 1(c)).

B. Proposal

We propose the latter scope of integration because we can integrate the interfaces to access data. Many OSSs today are designed to have their own databases and store information obtained from the NEs they manage. This means OSSs store the copies of the original data. We often conduct operations, such as referring, updating, appending, and deleting, on those data. On the other hand, we sometimes conduct the same

[1] Present affiliation: NTT Comware Corporation
Address: seshake.hikaru@nttcom.co.jp

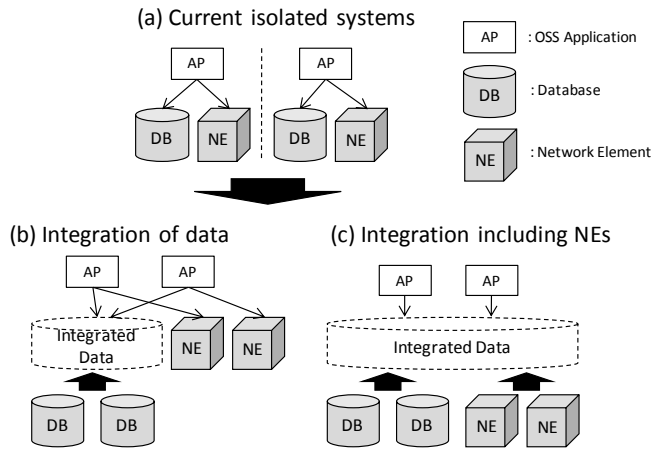

Fig. 1. Scopes of data integration

operations directly on the NEs. In the former case, SQL is widely used as an interface to access data, while an interface, such as SNMP, is widely used in the latter case.

Both cases have different operational targets but ultimately result in the same operation. Therefore, we argue that it would be effective to integrate the interfaces to access data to enable easier development of OSSs.

Of course, there are operations that do not seem to be targeted at the copied data such as command execution for NEs and event detection in NEs. We discuss this problem in Section 4.

III. Related Techniques

In this section, we describe related techniques and compare them with our proposed approach. We state this topic from two viewpoints: integration approaches and interfaces. The integration approaches viewpoint focuses discussion on methods of data integration, and the interfaces viewpoint focuses discussion on network operation and compares interfaces to access NEs.

A. Integration Approaches

There are two main approaches in data integration: physical (materialized) integration and virtual integration (e.g. [5]). In this section, we describe the features of each approach and discuss our proposed integration approach.

1) Physical Integration

a) Manual Integration

There have been many situations that require integration, such as a merger of multiple banks and related database integrations. In these situations, the data schema is often re-designed, a new database is reconstructed, and data are manually imported into the database. This is a primitive but the most fundamental way to integrate data. However, it is difficult to do and requires much time and money. In addition, it requires strong governance in the related organizations.

b) Data Warehousing

A data warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making" [6]. A data warehouse collects and stores data from multiple, distributed, and heterogeneous operational databases, organizes the data so they are consistent and easy to read, and keeps 'old' data for historical analysis so that users can conduct analysis [7]. In this approach, data must be first loaded into large storage using extract, transform, and load (ETL) tools, so we need additional facility investment. In addition, since the purpose of integration is only analysis, there is no support for data updating.

2) Virtual Integration

The other approach is virtual integration, which does not use additional central storage, and integrates the data virtually. A VDB is one of the most common technologies for this approach. We describe the architecture that enables the implementation of a VDB and related technology.

a) Virtual Database

A typical architecture of a VDB is shown in Fig. 2 (modified from [8]). It consists of a series of query processors, such as the query reformulation, query optimization, and query execution, and a set of wrappers. There are two types of data schemata in a VDB. One is the exported source schema, which is directly imported from a data source into the VDB. The other is the mediated schema, to which users can detect and send queries. The query processors reform a query in the mediated schema to that in the source schema, optimize the distributed query execution, and execute the optimized plan. A wrapper is a program that is specific to every data source whose task is to translate the queries in the source schema into actual queries in the data sources. The wrapper also translates the answers from the data sources into a form that can be further processed by the query processors.

b) SQL/MED

SQL/MED (management of external data) [9] is a SQL standard defined by ISO/IEC 9075-9:2008 that determines how a database management system can integrate data stored outside the database, and it is usually treated as a technique for implementing a VDB. External data can be data not only in relational databases (RDBs) but also data in NoSQL databases, files, and data managed by web services. PostgreSQL, a widely used open source database management system (DBMS), has supported this standard since version 9.1, so we can use it as an implementation of VDB. However, it has no support for updating queries at this time [10], so we do not consider this standard in this paper.

3) Proposed Approach

As described above, the physical integration approach has a problem of high cost since it requires expensive additional central storage or manual data integration. Compared to this, the virtual integration approach does not require additional storage and is relatively cost-effective. For this reason, we selected the virtual integration approach as our integration approach. On the other hand, the physical integration approach is superior with respect to performance since we can directly access the physical data in the local repository. In this paper, we verify this disadvantage in performance of the virtual

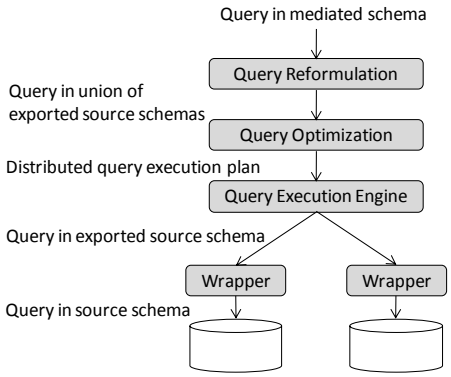

Fig. 2. Typical architecture of VDB (modified from [8])

integration approach in Section 5.

B. Interfaces

In Section 2, we described that the main advantage of integration including NEs is that we can integrate the interfaces to access all the data handled by OSSs. Basically, since a VDB is based on the relational DBMS (RDBMS), the main interface to the data is SQL, which is commonly used as the language for handling data in relational databases. SQL is the main interface to the data because it is basically based on RDB systems. We also have to take into account that there are several techniques for achieving interface integration, other than with a VDB, such as SNMP and Web services. We compared these techniques.

1) SNMP

SNMP is widely used as an interface to NEs and refers to the management information base (MIB) they have. The MIB is based on the original idea of the OSI Network Management Model specified by ISO 7498-4 / ITU-T X.700 [11]. This standard does not define the implementation of the MIB but defines only the logical structure of information (Fig. 3), and is often referred to as a VDB [13].

As described above, the original concepts of management are similar between SNMP and a VDB. In fact, there are standards, such as RDBMS-MIB [14], that enable integrated access to RDBMS.

One important difference between them is the style of programming. In SNMP, we basically need to combine the information obtained individually by writing codes. Compared to this, SQL can integrate information by itself since it is originally a language that combines multiple tables and creates a new information view. This means that the application codes will be relatively simple in the VDB approach since developers only have to write SQL declaratively.

Another difference is the interfaces they support. SNMP supports data reading (SNMP get), data writing/updating (SNMP set), and event notification/detection (SNMP trap) while SQL supports data reading (SELECT) and data writing/updating (UPDATE, INSERT, DELETE). Event notification/detection is an important role in managing NEs, so

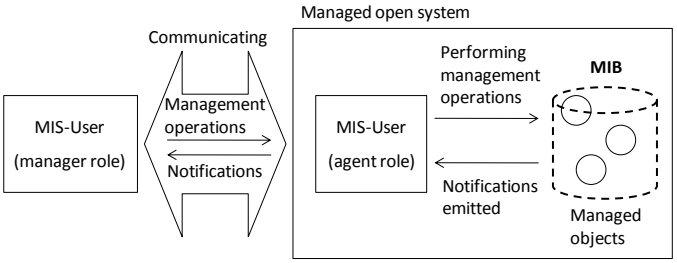

Fig. 3. Schematic of MIB (modified from [12])

the lack of this role would be a problem with the VDB approach. We discuss this issue in Section 4.

2) Web Services

Web services are also used as the interface for NEs such as TMF MTOSI [15]. Assuming NEs and databases are web services, we can also integrate their interfaces.

Similar to the discussion of SNMP in the previous section, the main difference between the web service approach and the VDB approach is the style of programming. In the web service approach, similar to the SNMP approach, we need to combine specific information by writing codes. Therefore, there is the same advantages and disadvantages of SQL compared with web services as in the discussion of SNMP.

3) Proposed Approach

As described in the SNMP section, SQL enables easier OSS development, especially in combining different data, because developers do not need to write many codes. Of course, there is a problem in that the VDB approach seems to be unfamiliar and complicated for developers who mainly develop applications and rarely write SQL. This problem can be solved by clearly distinguishing developers' roles into "application developer" and "database developer", and delegating development using SQL to a "database developer". We believe that this encourages the sharing of functions and eases the development as a whole. Therefore, we argue that the VDB approach is the best way to integrate OSS data and SQL is the best interface for it.

IV. Feasibility Evaluation (Functional)

In the previous section, we proposed our integration approach, which includes NEs as the data sources and stated that there are operations that seem to be difficult to express as data operations, such as command or test execution against NEs and event detection in NEs. We evaluated the feasibilities of those operations through a VDB.

A. Command Execution

Functions that "execute" something seem to be a problem because there is no concept of execution in the data domain. However, we argue that some "execute" functions are naturally translated into data operations and can be executed through the VDB.

We give an example of an execution of a "ping" command (Fig. 4). First, a client application issues the following query:

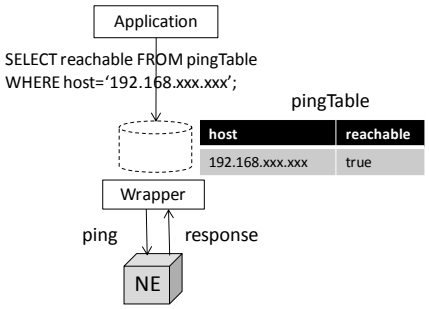

Fig. 4. Example of executing "ping" through VDB

```
SELECT reachable
    FROM pingTable
    WHERE host = '192.168.xxx.xxx';
```

where "pingTable" is a table in the VDB that has "host" and "reachable" columns. The information of the query is then sent to the wrapper for "ping", which we created, and executes the ping command to "192.168.xxx.xxx". This argument of the command was originally from the WHERE clause in the first query. When the response comes back from the host, the wrapper translates it into the form of a table in the VDB. The client can now determine whether it has reachability to a host "192.168.xxx.xxx" as an answer to the query above.

The "ping" command can be executed through the VDB since the wrapper can be written to translate anything, and the arguments that are necessary for the command execution can be provided as a WHERE clause. What is important here is that the SQL expression of that command is easy to understand. This is thought due to the fact that the client application's goal is to determine the information of reachability, not execute any commands.

Theoretically, this is not only the case of "ping" but also of any other operation which is called the "request-reply" process. Therefore, the problem is converted into a semantic one, such as "Which is suitable for this command, SELECT, UPDATE, INSERT, or DELETE?".

B. Event Detection

Basically, functions that use event detection, such as fault monitoring and congestion control, are difficult to implement using databases because they do not allow autonomous change of data; in other words, they do not have interfaces to allow queries from the data side. We discuss methods for event detection through a VDB.

One of the methods is based on the idea that a message from an NE is a response to a SELECT query the client application issued in advance. Issuing a SELECT query in advance allows the receiver application of an NE message to find the event that is occurring.

However, the implementation of this method may be complicated and many functions in the wrapper are required such as receiving the notification, suspending the SELECT query, and cueing the query. In addition, applications require another thread while the other thread waits for an event so other operations will not be blocked. Furthermore, this architecture has duplicate event receiving parts in the wrapper and application.

As described above, event detection seems difficult with the current functions of a VDB. This indicates that a function that enables event detection, such as transaction-monitor products, is required for our approach.

V. Feasibility Evaluation (Performance)

As described in Section 3, the virtual integration approach results in processing overhead compared with the physical integration approach. We conducted a feasibility evaluation from the viewpoint of performance. We first conducted a basic experiment to examine the general behavior of the VDB. We then investigated the feasibility of our approach in an actual situation using the data from the basic experiment.

A. Experimental Methodology and Environment

1) Environment and Methodology

Using an open source product called Teiid [16] as an implementation of a VDB, we measured the turnaround time of each data operation. Teiid is provided as an application on the JBoss application server (JBoss AS) together with Teiid Designer, a development environment of VDBs, running on Eclipse. The software configuration of this experiment and the hardware specifications are shown in Fig. 5. In this configuration, the VDB server, "data source" physical database server (PostgreSQL), and client applications are all in the same physical machine. To access the VDB from the client application, we used "Teiid JDBC" and to access PostgreSQL we used JDBC for PostgreSQL. The fetch size of JDBC (PostgreSQL, Teiid) was configured to 10,000, and the auto commit mode was set to false.

We used a physical database and files as data sources in this evaluation since they are considered to be the most common data stores in network operation. We compared the turnaround time of accessing the data sources between 1) through the VDB and 2) directly and evaluated that difference (overhead).

2) Data Specifications

We used tables that had 1000, 10,000, 100,000, 1,000,000, and 10,000,000 rows in the "data source" database (physical database). The data specifications, simply simulating the traffic or resource usage logs, are shown in Fig. 6. This is an example of a data size of 10,000,000. Other data sizes were created by adjusting the maximum number of "target_id" columns. Specifying "mtime" and "target_id" can specify one record. We did not set any primary keys. As a VDB schema, we used the same schema as table "traffic" in the physical database as a source model and did not use the view model (In Teiid, the mediated schema is called "view model", and the exported source schema is called "source model"). We used the same data schema as in the physical database as for the file, but in CSV format. We used only one file, not multiple files, for each data size.

3) Data Operation

We conducted an all-records search ("SELECT * FROM traffic") and one-record search ("SELECT * FROM traffic

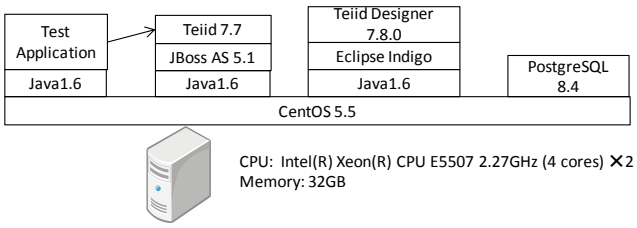

Fig. 5. Software configuration and hardware specifications for experiment

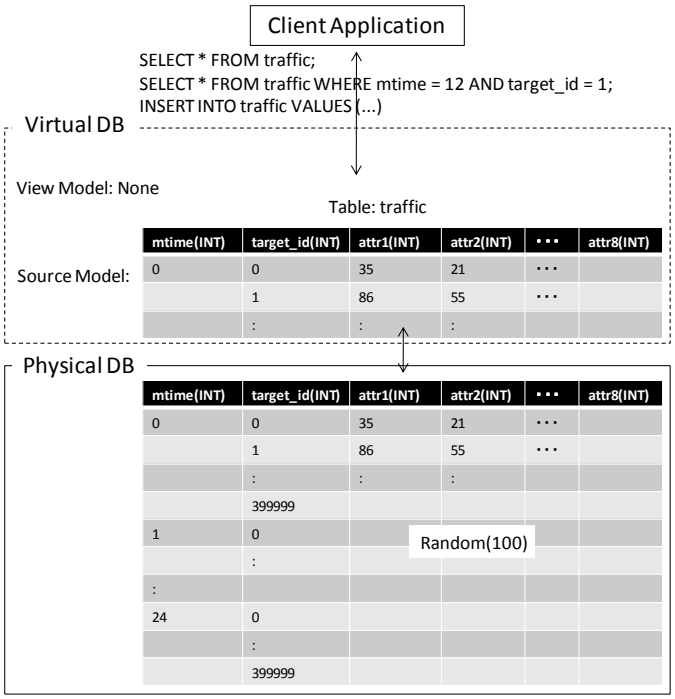

Fig. 6. Data specifications

WHERE mtime = 12 AND target_id = 1") as reading operations. We conducted evaluations of INSERT queries ("INSERT INTO traffic VALUES(…)") as writing operations. It is believed that INSERT takes too much time in actual situations, so we performed another operation that directly writes data to a file and compared the result.

B. Results

1) Reading

The measurement results for data reading are shown in Fig. 7. Fig. 7(a) shows the sum of the execution and the fetching times for all-records search and Fig. 7(b) shows the execution time for one-record search. The execution time was measured as the turnaround time of JDBC command "executeQuery", and the fetching time was measured as the turnaround time of the entire loop, each of which had one "next" command (Fig. 8). The solid line shows the results when we use a file as a data source through a VDB, and the dotted line shows the same but the data source is a database. The dashed line shows the results when we directly access a physical database. While the overhead was fixed under 100 ms in the one-record search, it increased as the number of records in the physical database increased in the all-records search (note that the vertical axis is log-scale). This result indicates that the VDB executes and

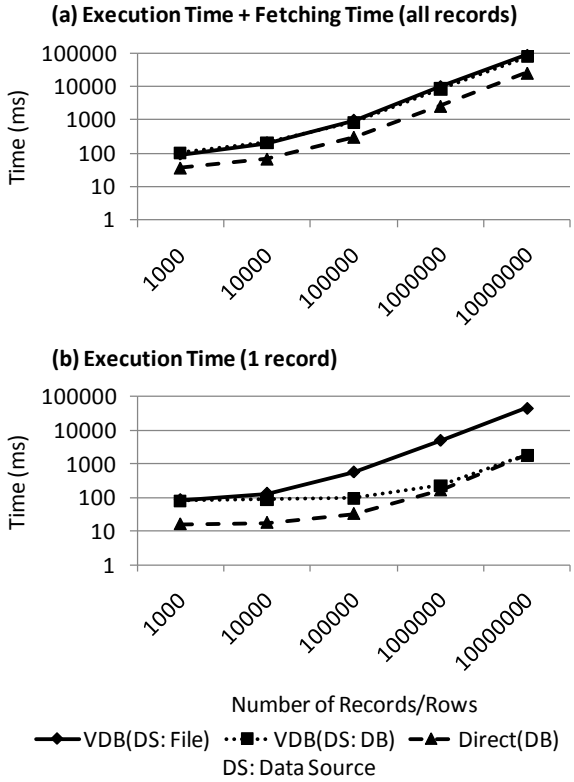

Fig. 7. (a) Execution and fetching times for all-records search and (b) execution times for one-record search

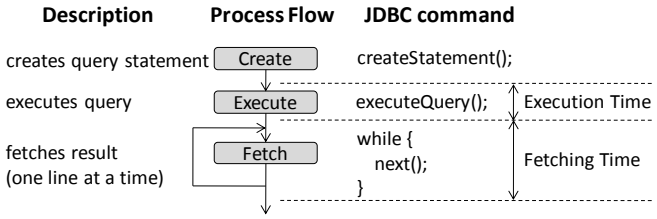

Fig. 8. Definition of execution and fetching times

fetches all the records first in the source database, then conducts some translations where the execution time is proportional to the number of records.

2) Writing

Fig. 9 shows the execution time when we write the number of rows shown in the horizontal axis. The solid line and the dotted line indicate the results of INSERTs through a VDB and directly, respectively. The dashed line indicates the results when we write data in the CSV file format directly, not through the VDB. Naturally, writing data in the file format is much faster (more than 100 times faster) than into databases by INSERTs, even not through VDB.

C. Discussion

1) Reading

One of the largest amounts of data that OSSs handle is usage data such as resource usage logs. We investigated the

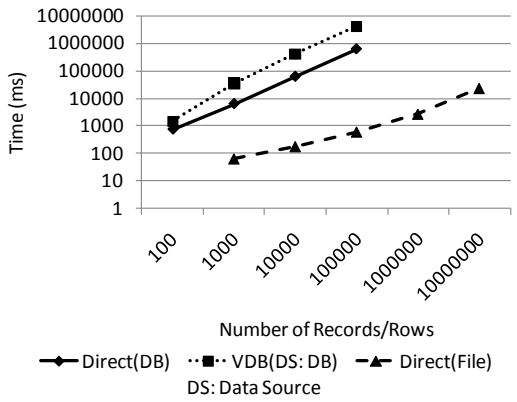

Fig. 9. Turnaround time for data writing

function that displays usage data to an operator as an actual situation. We assumed that the number of managed devices was 50,000 and an OSS was receiving data from these devices every 5 minutes, resulting in about 14,000,000 records of data created per day.

Fig. 7 indicates that while there was fixed overhead in the one-record search, it increased as the number of rows increased in the all-records search. As a result, it took more than 1 minute for 10,000,000 records, which is not so different from the data size we estimated from the assumption above.

Taking more than 1 minute to display usage data would be frustrating to an operator, but there seems to be no actual situation for an operator to display such a large amount of data at once. The valid number of records to display at once is about 500 at most. We confirmed that the execution time for this range of data size is less than 2 seconds in another experiment (the issued query is "SELECT * FROM traffic WHERE mtime=12 AND id < n", where n is the number of expected answer records), and this seems to be acceptable.

As described above, we discussed the feasibility of the functions that read and display data to operators using a VDB.

2) Writing

Similar to the assumption discussed in the previous section, we investigated a function that stores usage data in a data source. According to Fig. 9, the turnaround time for INSERTs was about 40 ms per record when we used the VDB. This is derived from a simple calculation of taking more than 100 hours to store 10,000,000 records, and we could not finish the operation in one day. Not surprisingly, writing data into files is more than 100 times faster. According to Fig. 7, however, reading one record from a large file that has 10,000,000 rows is more than 20 times slower compared to reading data from databases and takes about 45 seconds. Therefore, we should use files when we write large data and should use databases when we read data from a large data set.

There are several methods for implementing this. One is importing files into databases using the "import" command ("COPY FROM" in PostgreSQL). We also measured the file importing time into a database, and obtained results of about 5 seconds for 1,000,000 rows and 54 seconds for 10,000,000 rows, which is fast enough for actual use. Another method is using external tools such as rsyslog [17], which is an advanced version of syslog and can store syslog messages into databases.

VI. CONCLUSION

We proposed an OSS data integration approach that includes NEs as the data sources using virtual database technique and discussed its feasibility. From the viewpoint of functionality, functions such as command execution are possible by wrappers that translate the queries for the mediated schema into any operation, while functions such as event detection are difficult in a current database manner. From the viewpoint of performance, we explained feasibility in reading data, especially when displaying data to operators. As a data source, we also explained that files are suitable for writing large data while databases are suitable for reading.

REFERENCES

[1] A. Halevy, A. Rajaraman, and J. Ordille. "Data integration: The teenage years," In VLDB, pp. 9-16, 2006.

[2] R. E. Giachetti, "A framework to review the information integration of the enterprise," International Journal of Production Research, vol. 42, no. 6, pp. 1147-1166, 2004.

[3] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," ACM Computing Surveys, vol. 22, no. 3, pp. 183-236, 1990.

[4] J. Berlin and A. Motro, "Autoplex: Automated discovery of content for virtual databases," In Cooperative Information Systems, Springer Berlin Heidelberg, pp. 108-122, 2001.

[5] R. Hull and G. Zhou, "A framework for supporting data integration using the materialized and virtual approaches," SIGMOD, pp.481-492, 1996.

[6] W. H. Inmon, Building the Data Warehouse, John Wiley, 1992.

[7] S. Kelly, Data warehousing in action, John Wiley & Sons, 1997.

[8] A. Y. Levy, "The information manifold approach to data integration," IEEE Intelligent Systems, 13, 1998.

[9] ISO/IEC 9075-9:2008 : "Information technology —Database languages — SQL — Part 9: Management of External Data （SQL/MED）," International Organization for Standardization, 2008.

[10] "CREATE FOREIGN DATA WRAPPER", PostgreSQL 9.2.4 Documentation, http://www.postgresql.org/docs/9.2/static/sql-createforeigndatawrapper.html, accessed 2013-05-14.

[11] ITU-T Recommendation X.700, "Management Framework Definition for Open Systems Interconnection(OSI) for ITU-T Applications.", 1992.

[12] ITU-T Recommendation X.701, "Information technology - Open Systems Interconnection- Systems Management Overview.", 1992.

[13] J. Sathyan, Fundamentals of EMS, NMS and OSS/BSS, Auerbach Publications, 2010.

[14] D. Brower, B. Purvy, A. Daniel, M. Sinykin, and J. Smith, RFC 1697, "Relational Database Management System (RDBMS) Management Information Base (MIB) using SMIv2," IETF, August 1994.

[15] "Standardized Interfaces MTOSI", Telemanagement Forum, http://www.tmforum.org/MTOSI/2319/home.html, accessed 2013-05-14.

[16] "Teiid - JBoss Community", http://www.jboss.org/teiid/, accessed 2013-05-14.

[17] "The enhanced syslogd for Linux and Unix rsyslog", http://www.rsyslog.com/, accessed 2013-05-14.