Providing Database as a Service

Hakan Hacıgümüş
Department of Information and Computer Science
University of California
Irvine, CA 92697, USA
hakanh@acm.org

Bala Iyer
IBM Silicon Valley Lab.
San Jose, CA 95141, USA
balaiyer@us.ibm.com

Sharad Mehrotra
Department of Information and Computer Science
University of California
Irvine, CA 92697, USA
sharad@ics.uci.edu

Abstract

In this paper, we explore a new paradigm for data management in which a third party service provider hosts "database as a service" providing its customers seamless mechanisms to create, store, and access their databases at the host site. Such a model alleviates the need for organizations to purchase expensive hardware and software, deal with software upgrades, and hire professionals for administrative and maintenance tasks which are taken over by the service provider. We have developed and deployed a database service on the Internet, called NetDB2, which is in constant use. In a sense, data management model supported by NetDB2 provides an effective mechanism for organizations to purchase data management as a service, thereby freeing them to concentrate on their core businesses. Among the primary challenges introduced by "database as a service" are additional overhead of remote access to data, an infrastructure to guarantee data privacy, and user interface design for such a service. These issues are investigated in the study. We identify data privacy as a particularly vital problem and propose alternative solutions based on data encryption. This paper is meant as a challenges paper for the database community to explore a rich set of research issues that arise in developing such a service.

1. Introduction

Advances in the networking technologies have triggered one of the key industry responses, the "software as a service" initiative, also referred to as the application service provider (ASP) model. In this paper, we explore the "*database as service*" paradigm and the challenges introduced by that.

Today, efficient data processing is a fundamental and vital issue for almost every scientific, academic, or business organization. Therefore the organizations end up installing and managing database management systems to satisfy different data processing needs. Although it is possible to purchase the necessary hardware, deploy database products, establish network connectivity, and hire the professional people who run the system, as a traditional solution, this solution has been getting increasingly expensive and impractical as the database systems and problems become larger and more complicated.

As it is described above, the traditional solution entails different costs. It might be arguable that hardware, software, and network costs are decreasing constantly. People costs, however, generally, do not decrease. In the future, it is likely that computing solution costs will be dominated by people costs [13]. There is need for database backup, database restore, and database reorganization to reclaim space or to restore preferable arrangement of data. Migration from one database version to the next, without impacting solution availability, is an art still in its infancy [5]. Parts of a database solution, if not the entire solution usually become unavailable during version change. An organization that provides database service has an opportunity to do these tasks and offer a value proposition provided it is efficient.

The new paradigm challenges the traditional model of data management followed by current organizations. Database service provider provides seamless mechanisms for organizations to create, store, and access their databases. Moreover, the entire responsibility of database management, i.e., database backup, administration, restoration, database reorganization to reclaim space or to restore preferable arrangement of data, migration from one database version to the next without impacting availability will befall such an organization. Users wishing to access data will now access it using the hardware and software at the service provider instead of their own organization's computing infrastructure. The application would not be impacted by outages due to software, hardware and network-

Proceedings of the 18th International Conference on Data Engineering (ICDE'02)
1063-6382/02 $17.00 © 2002 IEEE

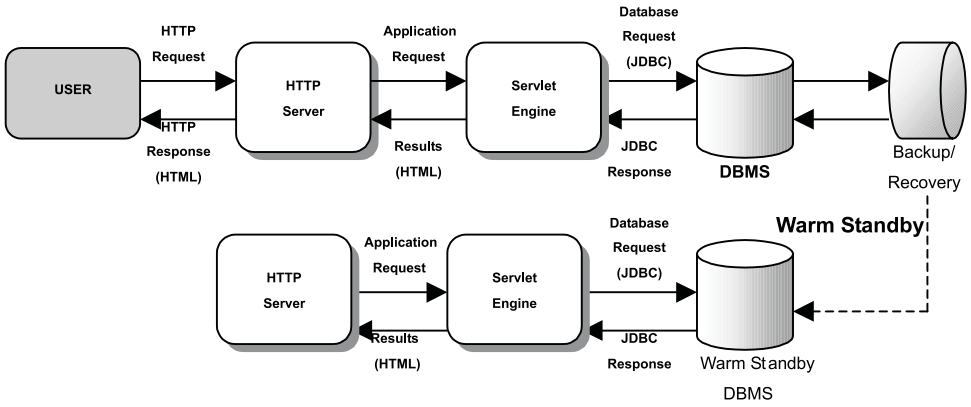

Figure 1. System architecture of NetDB2

ing changes or failures at the database service provider's site. This would alleviate the problem of purchasing, installing, maintaining and updating the software and administrating the system. Instead of doing these, the organization will only use the ready system maintained by the service provider for its database needs.

The technological aspects of developing database as a service lead to new research challenges. First and foremost is the issue of *data privacy*. In the database service provider model, user data needs to reside on the premises of the database service provider. Most corporations view their data as a very valuable asset. The service provider would need to provide sufficient security measures to guard the data privacy. We propose data encryption as the solution to this problem. Detailed investigation of this solution is presented in Section 5.

Second key challenge is that of *performance*. Since the interaction between the users and the database service provider takes place in a different medium, the network, than it does in traditional databases, there are potential overheads introduced by this architecture. Therefore the sources of performance degradation and its significance should be determined.

Another challenge facing the database service provider model is that of an appropriate *user interface*. Clearly, the interface must be easy to use; yet it needs to be powerful enough to allow ease in building applications.

We have developed and deployed a database service on the Internet, called NetDB2, an experimental network-based application service provider (ASP) system. It has been operational over a year and used by number of universities to help teaching database courses at different locations. NetDB2 provides database services including tools for application development, creating and loading tables, and performing queries and transactions to the users over the Internet.

In the system, data and all of the necessary database products are located on the server site. A user makes a connection to the system through the Internet and performs the database queries and other relevant tasks over the data through a web browser or an application programming interface such as JDBC [6]. The design principle of the system is to absorb complexity and workload on the server site as much as possible. The goal is to keep the client side lightweight, possibly requiring only a web browser to access the system. This makes the system portable and readily available from any location without any installation and configuration at the client side. By using a web browser based connection and web interface, the user has a chance to access and use the whole set of database products, which are professionally managed, without worrying about the system administration, maintenance, upgrading the system etc.

The rest of the paper is organized as follows. Section 2 presents NetDB2's system architecture. Section 3 describes user interface design of NetDB2. Section 4 discusses additional overheads due to the World Wide Web access to NetDB2 system and presents experimental results based on TPC-H benchmark queries. In Section 5 we describe our solution to data privacy problem and provide experimental results for different alternatives proposed in the study. We conclude the paper in Section 6.

2. System Architecture

The basic NetDB2 system is implemented as a three-tier architecture, namely; the presentation layer, the application layer, the data management layer (Figure 1). There are two benefits of separating NetDB2 into layers. The first is the insulation of software components of one layer from another,

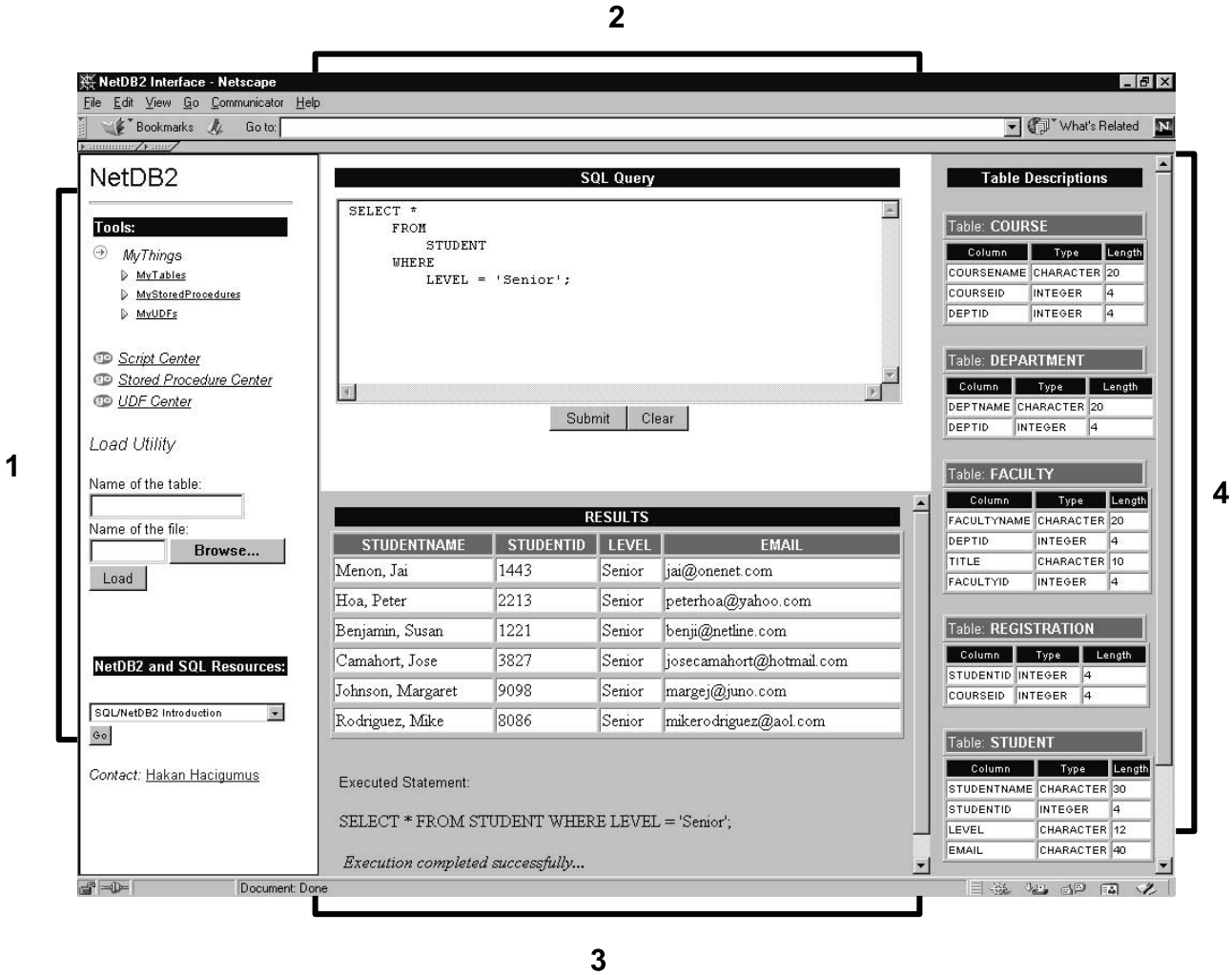

Figure 2. A snapshot from NetDB2 client screen

the other is the separation of concerns that helps achieve better interoperability and higher scalability.

NetDB2's presentation layer consists of the end user's web browser and NetDB2's HTTP server. The end user's browser is responsible for displaying the user interface and the HTTP server manages the communication between the browser and the application. The application executes server side logic, which generates the user interface.

The application layer consists of Java servlets managed by a servlet engine. Java was selected out of the desire for platform portability. In response to user interaction, HTML pages are generated and handed over to the presentation layer. This layer is also responsible for user authentication, session management by using session ids, and database connection management. Section 3.2 describes how the basic architecture was extended to support user defined interfaces.

The data management layer consists of a database manager and a backup/recovery server. The servlet engine communicates with the database using the JDBC protocol [6]. The database server and the backup/recovery server communicate, on a set schedule, through a private and secure high-speed network, without human involvement. On a set schedule backed up data is automatically restored to a warm standby NetDB2 system, with take over capability.

3. User Interface

3.1. NetDB2's Visual Interface

NetDB2 provides a web interface, which makes the system accessible from any computer running a web browser via Internet. Through NetDB2's user interface, one can create/remove tables, views, triggers, indexes, abstract data types, SQL queries, generate and call user defined functions

and stored procedures, creating and deleting indexes, etc. In a sense, this interface supports portions of database application development that can be pushed into the database.

A screen snapshot of NetDB2's visual interface is given in Figure 2. (This view is obtained after (1) querying metadata for table names and (2) after submitting a select query.) The first screen seen by the user logging on to NetDB2's visual interface is divided into four parts. The left portion of the screen (Region 1 in the Figure 2.) lists available tools and documentation. The middle region of the screen is used to type in queries and obtain their results. The upper half (Region 2 in the Figure 2.) is reserved for entering a SQL query, and the lower half (Region 3 in the Figure 2.) for results. Users commonly refer to metadata, tools and documentation during the use of the service. Whenever metadata is queried from here, the metadata queried is displayed on the right portion of the screen (Region 4 in the Figure 2.).

The user interface mainly provides the following functionality; *metadata* information for users database, the *Script Center*, which allows users to send more than one SQL queries, the *Stored Procedure Center*, which is used to create stored procedures, the *User Defined Function Center*, which is used to extend the built-in functions supplied with the database manager, *Load Utility*, which is used to upload (bulk) data from the end users computer through the network and insert into tables specified by them.

3.2. NetDB2's User Defined Interface

Another class of NetDB2 users is the users who provide access to their data on NetDB2 to *end-users*. We refer to such users as *application service providers*. They wish to present their own user interfaces to the end-users instead of NetDB2's user interface. NetDB2 also supports these users with its triangulation architecture given in Figure 3.

Application service providers are required to register the URL of their user interface with NetDB2. When an end-user logs into NetDB2, the end-user is redirected to the registered URL and a NetDB2 session is initiated for the end-user. Session management responsibility is provided by NetDB2, on behalf of the application service provider, simplifying application service providers programming. NetDB2 gives the application service provider the ability to set the redirection URL dynamically by reading it from an updateable table. Queries may be sent to NetDB2 by using supported programmatic interfaces, like JDBC [6], or by passing query parameters through URL.

It is also possible to allow NetDB2 end-users to be redirected to application service providers interfaces without logging into NetDB2. In this case, however, session management and authentication would need to be handled by the application service provider. Database calls are coded in the same way described above.

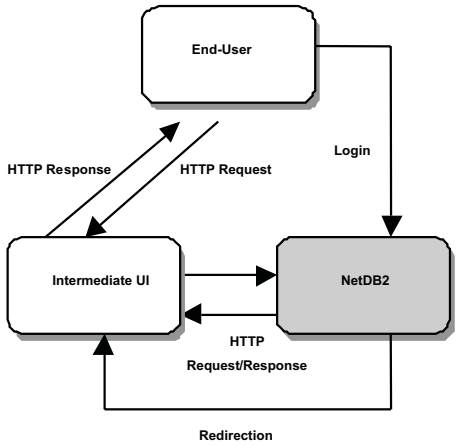

Figure 3. NetDB2's triangulation architecture for user defined interface

4. Performance Considerations for Service Delivery Penalty

In our implementation of database as a service, users access the database over the Internet and the results of their queries are sent back over the Internet. Therefore, we evaluated the overhead introduced by the extra infrastructure that converted database into a service. This overhead will be referred to as the *service delivery penalty*. We studied NetDB2 performance using the industry standard TPC-H benchmark [15] on scale factor of 0.1, 1, and 10 databases.

The TPC-H queries were first run directly on the DB2's database manager from the server machine console. In this configuration, there is no network overhead. Next we re-ran the TPC-H benchmark queries, this time using the NetDB2 interface, and over the Internet. The response time measurements showed that for the overall benchmark the service delivery penalty, i.e., the extra percent response time overhead for running TPC-H is 28% for 0.1 scale database, 8% for scale 1 database, and 1% for scale 10 database. The number of rows returned to the user grows less than linearly with database size (scale), 10726 rows for 0.1 scale, 70815 for 1 scale, 541220 for 10 scale. The fractional overhead due to the web decreases with TPC-H scale factor. The results are shown in Figure 4.

The TPC-H experiments were conducted on an IBM Netfinity 5500 server with dual Pentium III 600 MHz processors with 512 MB RAM. Software components used were IBM DB2 v7.1, IBM WebSphere Application Server v3.5, IBM JDK v 1.1.1.8 and Microsoft Windows NT 4.0. The database service was made available on the Internet over a 10-Mb link.

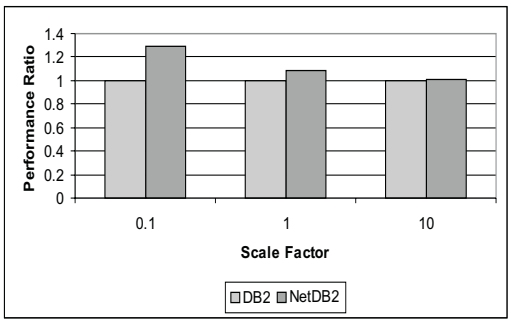

Figure 4. Relative performance comparison for DB2 versus NetDB2

5. Data Privacy

Privacy on the Internet is an issue that is of significant interest. There are two fundamental issues: 1) Privacy of data during transmission and 2) Privacy of stored data. The first issue, privacy during network transmission, has been studied widely in the Internet area and addressed by the Secure Socket Layer protocol (SSL) [8] and Transport Layer Security (TSL) protocol [4]. The second issue, privacy of stored data in relational databases is less studied and of greater relevance to database as a service model. If database as a service is to be successful, and customer data is to reside on the site of the database service provider, then the service provider needs to find a way to preserve the privacy of the user data. There needs to be security measure in place so that even if the data is stolen, the thief cannot make sense of it.

Encryption is the perfect technique to solve this problem. Prior work [7] [2] does not address the critical issue of performance. But in this work, for the first time, we have addressed and evaluated the most critical issue for the success of encryption in databases, performance. To achieve that, we have analyzed different solution alternatives.

There are two dimensions to encryption support in databases. One is the granularity of data to be encrypted or decrypted. The field, the row and the page, typically 4KB, are the alternatives. The field may appear to be the best choice, because it would minimize the number of bytes encrypted. However, as we have discovered, practical methods of embedding encryption within relational databases entail a significant start up cost for an encryption operation. Row or the page level encryption amortizes this cost over larger data. The second dimension is software versus hardware level implementation of encryption algorithms. Our results show that the choice makes significant impact on the performance.

5.1. Software level encryption

We considered two encryption algorithms: a) RSA [10] and b) Blowfish [11]. We conducted experiments using both these algorithms and found that the performance of the Blowfish algorithm we implemented in Java is better than the RSA implementation available to us. We report on our experience with the Blowfish algorithm. Blowfish is fast, compact, and simple, compared to other well-known encryption algorithms such as DES [12]. Detailed description of the algorithm is given in [12]. Blowfish is a 64-bit block cipher, which means that data is encrypted and decrypted in 64-bit chunks. This has implication on short data. Even 8-bit data, when encrypted by the algorithm will result in 64 bits.

Blowfish implementation was registered into the database as a user defined function (UDF) (also known as foreign function). Once it was registered, it could be used to encrypt the data in one or more fields - whenever data was inserted into the chosen fields, the values are encrypted before being stored. On read access, the stored data is decrypted before being operated upon.

For example, if we were to encrypt the column `discount` of a table called `lineitem` using the user defined function called "*encrypt*", and decrypt it by the user defined function "*decrypt*" one would use the following SQL command to insert data into the table `lineitem`:

```
insert into lineitem (discount)
values (encrypt(10,key))
```

The statement to select the encrypted field is given next:

```
select decrypt(discount,key)
from lineitem
where custid = 300
```

In this approach the creator of the encrypted data supplies the *key*, and the database provides the encryption function. Only those users who are given the key can decrypt the data using the decryption algorithm. Since the key is owned by the creator, and not stored at the site of the database service provider, unauthorized person who may get hold of disk files can not get hold of the key. In fact, even employees of the database service provider do not have access to the encryption key. The full security provided by the encryption algorithm is inherited by the data in the database. Note that we used the generic function name *encrypt* and *decrypt* in the query. In fact, we could implement the two functions with any encryption algorithm. Also note that users of our database service can easily specify and use encryption algorithms of their choice, using the facilities provided by our database service.

5.2. Hardware level encryption

Specialized encryption hardware, the IBM S/390 Cryptographic Coprocessor, is available under IBM OS/390 environment with Integrated Cryptographic Service Facility (ICSF) libraries. IBM DB2 for OS/390 provides a facility called "*editproc*" (or edit routine), which can be associated with a database table. An edit routine is invoked for a **whole row** of the database table, whenever the row is accessed by the DBMS.

We registered an encryption/decryption edit routine for the tables. When a read/write request arrives for a row in one of these tables, the edit routine invokes encryption/decryption algorithm, which is implemented in hardware, for whole row. We used the DES [3] algorithm option for encryption hardware.

5.3. Encryption scheme alternatives

We did not consider all possible combinations of different encryption approaches, namely; software and hardware level encryption, and different data granularity. We started with software encryption at field level. As it is presented in the performance results, for some particular cases, such as TPC-H Query #1, this introduced prohibitively large overhead on query response time. Note that, one selected field was encrypted to evaluate this combination. Having these results, we predicted that software encryption at row level, where all of the fields are encrypted, and page level encryption will introduce even much higher performance overheads. Because of this, we directed our experiments to hardware level encryption alternatives.

In hardware encryption, we did not consider field level encryption. The main reason is the expansion in the original data size due to the nature of block cipher encryption algorithms. This behavior is described in software level encryption section. This problem is not severe when the input data is typically 80-120 bytes row as generally the size of a row is relatively larger than a size of a field. For example, in TPC-H benchmark database, there are number of fields, which are defined as one byte character data, which becomes 8 bytes when processed by the encryption algorithm.

5.4. Encryption Penalty

If we compare the response time for a query on unencrypted data with the response time for the same query over the same data, but with some or all of it encrypted, the response time over encrypted data will increase due to both the cost of decryption as well as routine and/or hardware invocations in DB2. This increase is referred to as the *encryption penalty*. To measure the encryption penalty, first, we used all the queries in the TPC-H benchmark for software level encryption in the way described above.

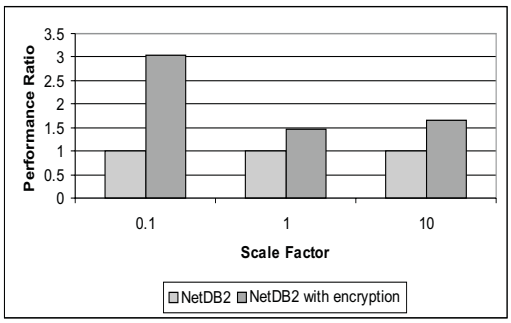

Figure 5. Relative performance comparison for NetDB2 versus NetDB2 with encryption (Query #1 excluded)

We found software field level encryption to be particularly CPU intensive. When we encrypted all fields of all tables of the TPC-H workload, the encryption penalty was extremely large. An observation according to recent studies is that, different fields have different sensitivity [16]. It is possible for NetDB2 to support encryption only on selected fields of selected tables. To illustrate this, we picked one field for the TPC-H schema and encrypted it. In particular we picked a field from the largest table in the database, `l_discount` from the `lineitem` table, and encrypted it. As described in an earlier section, encryption was defined as a foreign function. We used the Blowfish algorithm. Except for one query, Query #1, the overheads, although still large, show that it is possible to selectively support encryption. The performance results for running all the queries in the TPC-H benchmark, except Query #1, is given in Figure 5. Query #1, which first showed a dramatic increase in response time, is studied in more detail.

Query #1 from the TPC-H suite is given in Figure 6. The query refers to the majority of rows of a single table, `lineitem`, and computes multiple aggregates. Three of the aggregate computations require the decryption of the encrypted field `l_discount`. Decryption took place three times for each row of the table that passes the selection condition. Since the same field that is decrypted thrice; we looked for a way to eliminate the two redundant decryptions. We were able to rewrite the query using a feature of DB2 that temporarily materializes a specific part of a query. We used the feature in a way to reduce to only one decryption function execution per row selected by the query. The rewritten query is given in Figure 7.

Rewriting the query improves the response time of Query #1 by a factor of 3.8 approximately. We measured the improvement for TPC-H database sizes of 0.1, 1 and 10. The improvement ratios are given in Figure 8. Improvements are due to a factor of 3 reduction in decryption calls

```
select
    l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - decrypt(l_discount,key))) as sum_disc_price,
    sum(l_extendedprice * (1 - decrypt(l_discount,key)) * (1 + l_tax)) as sum_charge, avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price, avg(decrypt(l_discount,key)) as avg_disc, count(*) as count_order
from lineitem
where
    l_shipdate <= date ('1998-12-01') - 90 day
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus
```

Figure 6. First version of Query #1 with encryption

```
declare global temporary table session.dd (lr char(1),ll char(1), lq float, le float, d_discount float, lt float, ls date)
on commit preserve rows not logged;

insert into session.dd
select l_returnflag, l_linestatus, l_quantity, l_extendedprice, decrypt(l_discount,key), l_tax, l_shipdate
from lineitem
where l_shipdate <= date ('1998-12-01') - 90 day;

select
lr, ll, sum(lq) as sum_qty, sum(le) as sum_base_price, sum(le * (1 - d_discount)) as sum_disc_price,
sum(le * (1 - d_discount) * (1 + lt)) as sum_charge, avg(lq) as avg_qty, avg(le) as avg_price, avg(d_discount) as avg_disc,
count(*) as count_order
from session.dd
group by lr,ll
order by lr,ll;
```

Figure 7. Modified version of Query #1 with encryption

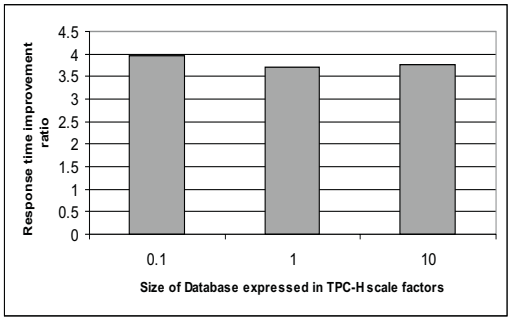

Figure 8. Query #1 response time improvement ratio due to rewrite

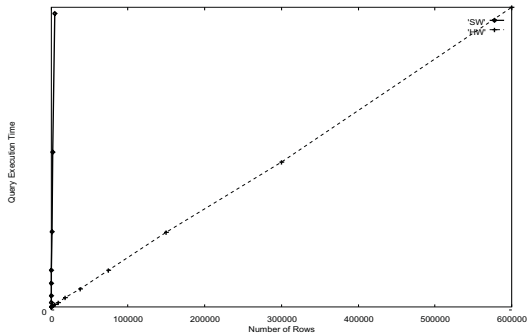

Figure 9. Comparison between software and hardware encryption for Query #1

and changes in the query plan. A formal method for this process is given in Section 5.6.

Since Query #1 shows a large increase in the first form of software encryption, we applied our hardware level encryption techniques on this query to measure and compare the performance of the alternatives. We followed the system setup described above and created TPC-H data on IBM OS/390 environment. Figure 9 shows the comparison between software level encryption and hardware level encryption. Figure 9 shows that as the number of rows increases, query execution time grows very sharply in software level encryption. (It is even hard to observe the curve for software encryption in the graph.) On the other hand, hardware level encryption shows almost perfectly linear increase, enabling to process all of the rows stored in the input table, which has more than 600.000 rows. Another very important point to note here is, all of the fields are encrypted in the hardware level encryption case, whereas only one field

(l_discount) is encrypted in software encryption case. Therefore the difference would be far larger if we had experimented the software level encryption by encrypting all of the fields in the table.

5.5. Page Level Encryption

As it was shown in the previous section, we obtained significant improvement in query response time by applying hardware level encryption techniques. The cost of encryption/decryption consists of start up cost, which involves function and hardware invocation, and encryption/decryption algorithm execution cost, which is depended on the size of the input data. This implies that the start up cost is paid every time a row is processed by encryption, since we use row level encryption. To eliminate the affect of start up cost, we investigated another technique, page level encryption. In this case the granularity of the data is a page. Whenever a page is accessed, it is encrypted/decrypted as it is performed for row level encryption.

Since there was no straightforward way to implement page level encryption, we evaluate the performance of page level encryption using an estimation model. To simulate page level encryption, we make each record of the lineitem table one page long by expanding the l_comment field and fixed the number of rows in the new table to number of pages occupied by the original lineitem table.

Hence, whenever a row is accessed in the new table, one page amount of data is processed. Since we keep the schema of the table same, except l_comment field, which is not used in the query, we are able to run the same query on the new table to compare the query response times. Figure 10 shows relative comparison among the original (non encrypted) query response time, row level hardware encrypted query, and estimate for page level hardware encrypted query. It is shown that, page level encryption introduces even more improvement on row level hardware encryption. The relative difference between non-encrypted query and page level encrypted query response time is 38%.

5.6. Query rewriting to improve software encryption

It is possible to automate the rewrite of a query to use temporary materialized views. A method for common subexpression elimination (CSE) needs to be applied to expensive user defined functions for a query. Common subexpression detection and elimination are well known in compiler optimization [1] [9]. An occurrence of an expression is a common subexpression (CS) if there is another occurrence

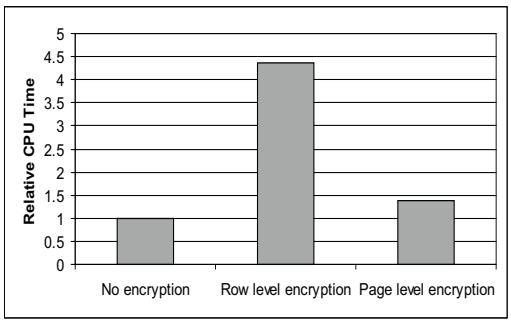

Figure 10. CPU time for no encryption, row level encryption, and page level encryption for Query #1

of the expression whose evaluation always precedes this one in execution order and if the operands of the expression remain unchanged between the two evaluations [9].

Common UDF subexpressions may be detected during query parsing. To illustrate, we give a simplified subset of SQL grammar in Appendix. For simplicity, lets assume that UDFs only take base columns as parameters, and not other UDFs. To detect CSs, we use a two-dimensional array, which stores UDF calls in the query text. Each row of the array stores the name of the UDF, the operand, and the number of times it is used. Whenever a new UDF is parsed out, the array is looked up to determine if the same UDF has been previously called with same operand. If it has, then the use count is increased, else the UDF, its operand are inserted as a new UDF into the array. At the end, if we detect some UDFs were called more then once with the same parameter, we treat them as a common subexpression and they are subject to our query rewriting procedure described below.

To form the materialized view (V) and the rewritten query (Q'), which replaces the original query (Q), we adapted the notation given in [14]. Here we give the notation for a simple select query with user defined functions, aggregate functions, GROUP BY, and ORDER BY clauses.

R_i denotes the tables in the database and $\tilde{A}_i$ denotes set of attributes of relation R_i, where $i = 1, \ldots, n$. $Tables(Q)$ denotes the set of tables with their columns in FROM clause where $Tables(Q) = \{R_1(\tilde{A}_1), \ldots, R_n(\tilde{A}_n)\}$. $Cols(Q)$ denotes $\tilde{A}_1 \cup \ldots \cup \tilde{A}_n$. $Sel(Q)$ denotes the set of *base* columns in SELECT clause. We define three groups of columns mentioned in SELECT clause: 1) *non-aggregation columns* are denoted by $ColSel(Q)$. 2) *aggregation columns* are written as $AGG(A)$, where AGG is one of the aggregation functions of SQL. The set of aggregation columns is denoted by $AggSel(Q)$. 3) *UDF columns* are in the form of $UDF(A)$, where UDF is the name of any valid user defined function. The set of UDF columns is denoted by $UdfSel(Q)$.

Similarly $Groups(Q)$ denotes the set of the columns in

GROUP BY clause of the query Q, and $Orders(Q)$ denotes the set of the columns in ORDER BY clause of the query Q, where $Groups(Q)$,$Orders(Q) \subseteq Cols(Q)$.

The conditions in WHERE clause of query Q is denoted by $Conds(Q)$.

We define three different types of column mapping from query Q to definition of view V. Each of these will correspond to mapping for $ColSel(Q)$, $AggSel(Q)$, and $UdfSel(Q)$. We assume that, all user defined functions are expensive functions in terms of system resources, therefore they are subject to CSE. We use same set of column mappings to form query Q'.

1. Mapping ϕ_S is defined as; $A \rightarrow B$, $A \in Sel(Q)$ and $B \in Sel(V)$; i.e., each element of non-aggregation columns of the query Q will directly be mapped.

2. Mapping ϕ_A is defined as; $AGG_A(A) \rightarrow B$, $A \in AggSel(Q)$ and $B \in ColSel(V)$, where A is the operand of aggregate function AGG_A; i.e., aggregation functions are removed in mapping.

3. Mapping ϕ_U is defined as; $UDF_A(A) \rightarrow UDF_B(B)$, $A \in UdfSel(Q)$ and $B \in UdfSel(V)$, where A is the operand of user defined function UDF_A; i.e., user defined functions are directly mapped.

The definition of the temporary materialized view will contain all of the mappings from $ColSel(Q)$, $AggSel(Q)$, and $UdfSel(Q)$ to view definition. We call this as maximal coverage. An alternative approach would be defining some subset of this mapping and defining a join block, which joins the view and the necessary elements of $Tables(Q)$. The algorithm steps are given in Figure 11. In the notation, B_A, $B_A \in Cols(V)$, denotes the corresponding mapping for column A, $A \in Cols(Q)$.

6. Conclusion

In this paper, we introduced NetDB2, an internet-based database service built on top of DB2 that provides users with tools for application development, creating and loading tables, and performing queries and transactions.

Database as a service model introduces many significant challenges primary of which are the additional overhead of remote access to data (service delivery penalty), an infrastructure to guarantee data privacy, and user interface design for such a service. We have addressed these issues. Our experiments using the TPC-H benchmark showed that the network overhead is tolerable. Data privacy can be achieved by using a suitable encryption algorithm. We proposed, implemented, and evaluated different encryption schemes. First, software level encryption techniques investigated. Field level encryption is implemented and evaluated. In this scheme selected number of fields of the given

```
// Define the temporary materialized view
1  for each A_i ∈ Sel(Q), 1 ≤ i ≤ |Sel(Q)|
       include B_i = φ_S(A_i) into definition of V
// Form insert statement
2  let ColSel(V) = {}, AggSel(V) = {}, UdfSel(V) = {}
3  for each A ∈ ColSel(Q)
       ColSel(V) = ColSel(Q) ∪ B,B = φ_S(A)
4  for each A ∈ AggSel(Q)
       ColSel(V) = ColSel(Q) ∪ B,B = φ_A(A)
5  for each A ∈ UdfSel(Q)
       UdfSel(V) = UdfSel(Q) ∪ B,B = φ_U(A)
6  Tables(V) = Tables(Q)
7  Conds(V) = Conds(Q)
// Form query Q'
8  Replace each A ∈ ColSel(Q) ∪ Groups(Q) ∪ Orders(Q)
       by B_A
9  Replace each A ∈ AggSel(Q) by AGG_A(B_A)
10 Replace each A ∈ UdfSel(Q) by B_A
11 Tables(Q') = {V}
```

Figure 11. Algorithm steps for query rewriting

table are encrypted. We showed that the query evaluation time can significantly be reduced by rewriting the queries. A formal method is also provided to make this process automatic. As a second step, we investigated hardware level encryption techniques. At this level, row level encryption scheme is implemented and evaluated. In row level encryption, all defined fields are encrypted as a whole. We showed the drastic decrease in query execution times from software level encryption. To obtain a more possible improvement, another encryption scheme, page level hardware encryption is suggested. We constructed an estimation model to evaluate the overheads of page level encryption. It was shown that even more improvement is possible by using page level encryption, which reduces the relative encryption overhead to 38%. We believe, from our experience, *database as a service* is a viable model and has a good chance of emerging as a successful commercial offering for some applications.

Acknowledgements

We thank Dante Aubert, Girma Bizuneh, Thomas Burke, Glen Deen, Joseph Demuth, Mohan Desouza, Linda Distel, Nick Donofrio, Anne Gardner, Satish Gupta, Don Haderle, Katharine Harris, Anant Jhingran, Kirk Jordan, Michael Kelley, Gopal Krishnan, Charles Lickel, Bruce McAlister, Diane Moebus, Inderpal Narang, Robert Pederslie, Tony Rall, and Guraraj Rao for their generous support.

References

[1] A. Aho, S. Johnson, and J. Ullman. Code generation for expressions with subexpressions. *Journal of ACM*, Jan., 1977.

[2] G. Davida, D. Wells, and J. Kam. A database encryption system with subkeys. *ACM Transactions on Database Systems*, 6(2), 1981.
[3] DES. Data encryption standard. *FIPS PUB 46, Federal Information Processing Standards Publication*, 1977.
[4] T. Dierks and C. Allen. The TSL protocol. *Internet Draft*, Nov., 1997.
[5] D. Gupta, P. Jalote, and G. Barua. A formal framework for on-line software version change. *IEEE Transactions on Software Engineering*, 22(2):120–131, 1996.
[6] G. Hamilton and R. Cattell. JDBC: A Java SQL API. *http://splash.javasoft.com/jdbc/*.
[7] J. He and M. Wang. Encryption in relational database management systems. In *Proc. Fourteenth Annual IFIP WG 11.3 Working Conference on Database Security (DBSec'00), Schoorl, The Netherlands*, 2000.
[8] P. Karlton, A. Freier, and P. Kocher. The SSL protocol v3.0. *Internet Draft*, Nov., 1996.
[9] S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers, 1997.
[10] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
[11] B. Schneier. Description of a new variable-length key, block cipher (blowfish), fast software encryption. In *Cambridge Security Workshop Proceedings*, pages 191–204, 1994.
[12] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 1996.
[13] I. Sommerville. *Software Engineering*. Addison-Wesley, 6th Edition, 2001.
[14] D. Srivastava, S. Dar, H. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. 22nd VLDB Conference, India*, 1996.
[15] TPC-H. *Benchmark Specification*. http://www.tpc.org.
[16] A. F. Westin. Freebies and privacy: What net users think. Technical report, Opinion Research Corporation, http://www.privacyexchange.org/iss/surveys/sr990714.html, 1999.

Appendix: Simplified Grammar for the SQL Subset

```
<query> ::=
  SELECT <select_list>
  FROM <range_list>
  WHERE <predicate>
  [ GROUP BY <attribute_list> ]
  [ ORDER BY <attribute_list> ]
<select_list> ::= <term>{"," <term>}
<range_list> ::= <relation_name>
                 {"," <relation_name>}
<attribute_list> ::=
  <attribute_name>{","<attribute_name>}
<predicate> ::= <comparison_predicate>
<comparison_predicate> ::=
  <term> <comp_op> <term>
<term> ::= <literal>|<attribute_name>|
           <aggr_func>|<user_def_func>
<aggr_func> ::=
  AVG|MAX|MIN|SUM|COUNT
  "(" <attribute_name>
  |{AVG|MAX|MIN|SUM|COUNT}|
  {<user_def_func>} ")"
<user_def_func> ::=
  <literal> "(" <attribute_name>|
  {<user_def_func>} ")"

  <relation_name>,<attribute_name>,<comp_op>
```

are defined as in relational calculus.