

Prototyping DIME, A Tightly Coupled Heterogeneous Distributed Database System

Haengrae Cho*

Yoo Sung Kim⁺

Songchun Moon*

* Dept. of Info. and Comm. Engineering
Korea Advanced Institute of Science & Technology
207-43, Cheongryang, Seoul, Korea

⁺ Dept. of Computer Science
INHA University
Inchon 402-751, Korea

Abstract

In this paper, we design and implement a prototypical heterogeneous distributed database system, named DIME (Distributed Information Management). DIME has the following salient properties. First, DIME allows both global retrieval operations and global update operations where different concurrency control schemes are used in different local database systems (LDBSs). Second, DIME implements international standard protocols on the distributed transaction processing and the remote database access. Last, DIME provides distribution transparency, and thus users can generate not only single site queries (including remote site queries) but also inter-site queries without considering data distribution in LDBSs.

1: Introduction

A practical approach to constructing very large distributed database systems is to integrate existing local database systems (LDBSs) into one global system, i.e., a federation. In case heterogeneity is involved, each LDBS may use different data model, database language, concurrency control, and/or recovery scheme. Therefore, a mapping (or composition) mechanism needs to be added to each LDBS in a heterogeneous distributed database system (HDDBS) [1-3]. When the already existing LDBSs are integrated into a federation, it is not desirable to modify the LDBSs for some practical reasons. In this respect, design autonomy, in terms of the original design of data model and of whatever protocols that each LDBS has, must be preserved in HDDBSs. That is, maintaining design autonomy is a strong requirement for HDDBSs.

For the past two decades, a number of HDDBSs, such as MULTIBASE [9], Proteus [1], ADDS [2], and DATAPLEX [3], have been implemented. The major aim of previous HDDBSs is to resolve the heterogeneities of local data models and local database languages. However, those systems, except DATAPLEX, do not provide global update operations, automatic updating of data stored at a number of LDBSs. In those systems, updating data stored at different LDBSs is either impossible or allowed to execute only in a restricted manner as off-line updates. Furthermore, all of the systems including DATAPLEX cannot deal with the heterogeneity of local concurrency control schemes (LCCSs). For example, DATAPLEX restricts the LCCS only to the two-phase locking.

In this paper, we describe the design and implementation of a tightly-coupled HDDBS, named distributed information management (DIME) in which heterogeneity among LCCSs exists but 100 percent of site autonomy is guaranteed. DIME has the following salient properties.

- Unlike the previous HDDBSs, DIME allows both global update operations and global retrieval operations where different LCCSs are used in different LDBSs. This is achieved by a *unified concurrency control scheme* [7] which is developed by authors.
- DIME implements international standard protocols on the distributed transaction processing [6] and the remote database access [5]. With the standards, DIME can alleviate the complexity of integrating different LDBSs.
- DIME provides distribution transparency to assist users access data. Users can generate both single site queries (including remote site query) and inter-site queries without considering data distri-

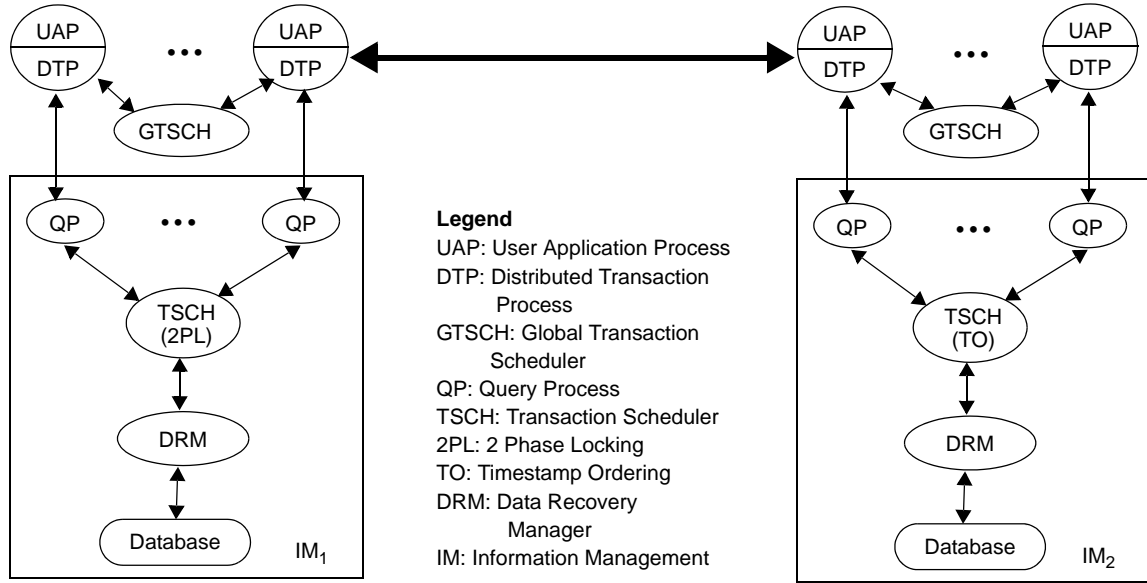


Figure 1: System architecture of DIME

bution in LDBSs. This results from an efficient handling of the distributed catalogs.

Because the main thrust of DIME is to support a global concurrency control scheme where the heterogeneity of LCCSs exists, DIME does not consider heterogeneities of local data models. DIME can only be applied to LDBSs each of which support relational data model with SQL interface. Furthermore, we assume that there are no *data incompatibilities* [17] between LDBSs. This results in simplifying the schema integration procedures that are not main focuses of current version of DIME.

The remainder of this paper is organized as follows. In Section 2, we describe the system architecture of DIME. The global query processing of DIME is presented in Section 3 and the distributed transaction processing of DIME is described in Section 4. In Section 5, we present a global concurrency control scheme where two-phase locking scheme and timestamp ordering scheme are used as LCCSs. The experimental results of DIME is presented in Section 6. Finally, Section 7 concludes with a summary and a future plan of DIME.

2: System architecture of DIME

DIME is implemented on top of an LDBS called information management (IM) which is the relational

DBMS developed in Korea by KAIST [10]. DIME uses SQL for the global user interface since SQL is the international standard of a database language [11]. To improve the extensibility of DIME, the communication mechanism between LDBSs is implemented with ISO standard protocol, such as the remote database access (RDA) [5] and the distributed transaction processing (OSITP¹) [6]. Figure 1 depicts the system architecture of DIME.

User application process (UAP) parses a user query and decomposes the query into a set of subqueries. UAP also maintains a *distributed catalog*. With the distributed catalog, UAP is able to generate the optimized execution plan of user query. The distributed catalog is regarded as a system database which contains location and type information of every data object stored in the distributed database.

Distributed transaction process (DTP) acts the role of guaranteeing atomic commitment of global transactions. To guarantee the atomic commitment of global transactions, two-phase commitment protocol is implemented according to the OSITP protocol. In DIME, communication between LDBSs is accomplished according to the RDA protocol. Since the RDA standard specifies the functionality of a database server within a distributed open systems envi-

1. As an abbreviation of the distributed transaction processing, we use not DTP but OSITP, which implies transaction processing on OSI. DTP will be used as an abbreviation of the *distributed transaction process* which is a main component of DIME.

ronment and specifies the communication services and protocols for accessing its capabilities, it is desirable to adopt the RDA protocol as a communication mechanism between LDBSs.

Global transaction scheduler (GTSCH) checks whether the serializability can be preserved at its underlying LDBS. To check the serializability, GTSCH certifies every transaction submitted to each LDBS. By forcing every transaction to be executed in the same order at every LDBS, GTSCH guarantees the global serializability of HDDBS. To preserve the local site autonomy, GTSCH must be attached on top of each LDBS without modifying the LDBS. To resolve the heterogeneity between LCCSs while guaranteeing both global serializability and complete local site autonomy, a unified concurrency control scheme [7] is implemented.

As an LDBS of DIME, we use IM. Since IM has been developed by authors, it was easy to modify IM so that it has a different structure. This results in cost-effective construction of multiple versions of IM, which can be integrated in a testing environment for DIME. In order to construct a testing environment where different LCCSs are used in different LDBSs, we implement two versions of IM: one uses two-phase locking and the other uses timestamp ordering. For each version of IM, other components (query process and data recovery manager) are same.

3: Global query processing

UAP is an interface between users and DIME, and plays a role to analyze and optimize user's SQL queries. Users access the database by the unit of a global query. A global query may be a single site query (including a remote site query) or a inter-site join query. When the global query is an inter-site join query, it accesses a set of data items stored at several LDBSs. The global query in this case is decomposed into a set of subqueries each of which will be executed at one LDBS. Those subqueries are then transmitted and executed at the relevant LDBSs. Finally, the execution results are returned to the origin site in which the global query is invoked. The global query processing procedure is depicted in Figure 2.

Decomposing a global query into a set of subqueries requires a lot of information for optimizing its performance. A distributed catalog is regarded as a system database that contains the required informa-

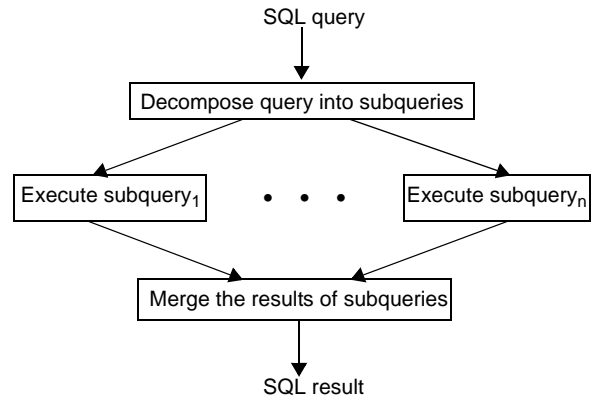


Figure 2: Global query processing in DIME

tion. In DIME, the distributed catalog maintains location map information and relationship information of data objects stored in a database. Figure 3 represents the structure of the distributed catalog of DIME.

GLREL					
relname	home	creator	relid	hostname	status

GLATT				
colname	coltype	offset	length	relptr

GLDB		GLALIAS		
dbname	relptr	colname	user	relptr

Figure 3: The structure of a distributed catalog

In Figure 3, GLREL maintains the site information of global relations. In GLREL, the attribute home represents the site in which the relation has been created, while the attribute hostname represents the site in which the relation is located currently. The attribute status implies that whether the relation is located in local site or remote site. GLATT maintains the attribute information of global relations. GLDB stores the location map information of global databases. GLALIAS stores the mapping information between the full name of a global relation and the alias name. In a distributed database system, each data object must have the system-wide unique name. To guarantee the uniqueness of naming in DIME, the site name in the computer network is attached to the original name of data object. For convenient referencing of data objects, the shortening alias name facility employed in R* [15] is also provided in DIME.

Distributed catalogs can be allocated in many different ways. Three basic alternatives are the central-

ized catalogs, the fully replicated catalogs, and the partitioned catalogs [16]. In case of centralized catalogs, the total catalogs are stored exactly at a single central site, and thus it suffers from the communication bottleneck and the system unreliability. In case of replicated catalogs, the total catalogs are replicated entirely at every site, and thus the cost of updating the catalog information must be very high. In case of partitioned catalogs, each site maintains its own catalogs for data objects stored at that site, and thus the cost of querying the catalog information at a remote site must be high.

In DIME, the distributed catalogs are allocated with a *hybrid manner* of fully replicated catalogs and partitioned catalogs. DIME partitions its component LDBSs to *disjoint groups* according to their access patterns. Tightly interacting LDBSs are in a same group. A distributed catalog which contains information of its member LDBSs is fully replicated to the member LDBSs of the group. In order to locate the data objects on different groups, DIME uses a broadcasting mechanism employed in partitioned catalogs [16]. This result in reducing the cost of updating the catalog information, because the catalog is replicated only to the member LDBSs. Furthermore, since inter-group data accesses are inherently rare, most of query operations on the catalog can be performed without inter-site communication.

Now we describe a schema integration procedure of DIME. The procedure is simple because every LDBS supports only a relational data model and there are no data incompatibilities between LDBSs. Every relation has unique name and its level of abstraction is similar. When a new LDBS tries to participate in DIME, the following two steps are performed.

- (1) A new distributed catalog that contains information of the new LDBS is created.
- (2) The contents of the distributed catalog are published to every member site of the group. The group information is provided by a DBA. For every member site, its distributed catalog is updated so that the catalog includes information of the new LDBS. After every member site updates its catalog, one of the member site (normally predefined) returns the resulting catalog to the original site. The original site can now update its distributed catalog. Therefore, every member site of a group can have the same distributed catalogs.

4: Distributed transaction processing

A transaction is an atomic unit of work and contains a sequence of queries. The transaction processing model of DIME follows OSITP model [6]. A transaction has four properties: atomicity, consistency, isolation, and durability. In order to meet the properties, OSITP protocol specifies the notion of a transaction tree, the two-phase commitment (2PC) protocol, and recovery mechanisms. However, in OSITP protocol, the mechanisms to assure compatibility of LCCSs are not specified. The only restriction of OSITP protocol on the LCCSs is that a global deadlock must be detected or avoided. Therefore, we develop a new global concurrency control scheme and extend OSITP protocol to cooperate the scheme.

When a transaction is submitted to the UAP, DIME executes the following steps. The corresponding communication protocol is illustrated in Figure 4.

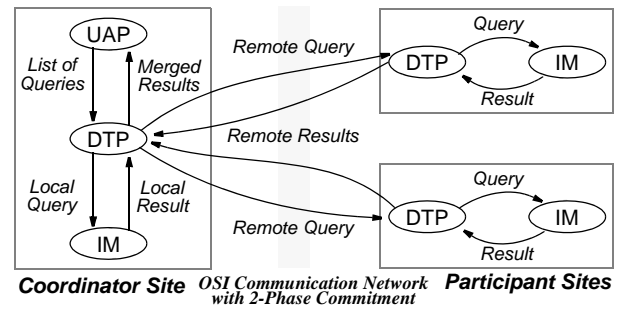


Figure 4: Communication protocol

- (1) A transaction is submitted to the UAP and it is decomposed into a set of subtransactions according to the location map information of the distributed catalog in the local site. The UAP submits the set of subtransactions to the DTP.
- (2) The DTP transfers each subtransaction to the site which stores data required by the subtransaction. In that site, a new DTP is generated to execute the subtransaction. Here, the DTP that sends the subtransaction becomes the *coordinator* process and the recipient DTPs become *participant* processes.
- (3) To check if a subtransaction does not violate the global serializability at participant sites, each participant DTP sends the certification request for the subtransaction to the GTSCH. Only the subtransaction which passes its certification is transmitted for execution at the underlying LDBS.
- (4) The execution results of each subtransaction are

returned to the coordinator DTP and are merged to generate the final SQL result.

- (5) If the subtransactions are aborted in step (3) or (4), the coordinator DTP is notified. The coordinator DTP then aborts the global transaction and its subtransactions. Otherwise, the coordinator DTP and participant DTPs commit the global transaction in an atomic way with the standard 2PC protocol [6].

DTP is implemented with OSITP protocol and RDA protocol. In the OSI reference model, since OSITP protocol and RDA protocol are classified into the application layer, we implement OSITP protocol and RDA protocol according to "Application Layer Structure" [12], where the structure of application layer in OSI reference model is specified.

As an implementation environment of the DTP, we chose *ISODE 7.0* (ISO Development Environment: Release 7.0) [13], which is a public software for the developer of the OSI standard protocols. Since ISODE implements the lower layers of OSI reference model and facilitates developing the protocols of application layer, it is an essential component in developing distributed applications like a distributed DBMS.

In order to implement OSITP protocol and RDA protocol with ISODE, we must define OSITP and RDA services with ASN.1 grammar [14]. ASN.1 is the description language used to define a set of primitive data types and provides a facility to construct new elements with their own typing inherent in the structure. For each services of RDA and DTP, three kinds of objects must be defined with ASN.1 grammar: arguments, results, and errors. ISODE reads the definitions and produces the corresponding C-stubs and definitions for use with run-time environment.

Once the descriptions of OSITP services and RDA services are compiled by ISODE, we must define processing rules for client and those for server. Processing rules for client include association establishment between client and server, operation invocation, association release, and error handling. Processing rules for server include association management, operation response, and error handling. For each services, we implement the processing rules with the corresponding C-stubs generated by ISODE.

Currently, DIME supports only a part of OSITP services and RDA services. The services supported in DIME are *RDA dialogue management*, *RDA database language service*, and *distributed transactions with two-*

phase commitment. With those services, DIME can preserve the atomicity of distributed transactions and execute queries on remote databases. While there are a few services which DIME does not support yet, we believe that the three services are the indispensable requirements of a distributed database system. The remaining services of OSITP and RDA are planned to be supported in the next version of DIME.

5: Global concurrency control scheme

In this section, we describe a basic idea of global concurrency control scheme which is developed by authors [7] and its implementation in DIME. Unlike the previous concurrency control schemes, our scheme guarantees local site autonomy while the global serializability is preserved in HDDBSs. Current version of the global concurrency control scheme allows two-phase locking scheme (2PL) and timestamp ordering scheme (TO) to be used as an LCCS.

In our scheme, the *pessimistic approach* is used for concurrency control in HDDBSs. To correctly detect and resolve conflicts between transactions submitted to LDBSs, the *agent* of global concurrency controller is attached on top of each LDBS as a preprocessor of the LDBS. Not only global subtransactions but also local transactions are certified whether the transaction violates the global serializability. The transactions submitted to the agent of global concurrency controller are coordinated on the basis of LCCS of its underlying LDBS. That is, local transactions that are submitted to the LDBS in which 2PL (or TO) is used are scheduled by 2PL (or TO). The subtransaction of global transaction G_j is scheduled on the basis of LCCSs that is used at the following LDBSs: the LDBS to which G_j is submitted and the LDBS in which the subtransaction is executed. Therefore, the indirect conflicts due to local transactions can be detected by the agent of each LDBS. The detailed descriptions of the global concurrency control scheme including its correctness proof are shown in [7].

Now we present the implementation of the global concurrency control in DIME. In DIME, the GTSCH corresponds to the agent of global concurrency controller. Since the GTSCH is implemented as a preprocessor on top of LDBSs and the communication between the DTP and the GTSCH is performed by the unit of an SQL statement, *the physical address of data object accessed by transactions cannot be known at the*

GTSCH. Therefore, as the granularity of global concurrency control, we use the *predicates* of an SQL statement rather than the physical address of data object, like page ID and record ID. A predicate specifies the condition of data to be accessed. Among SQL statements, the SELECT statement is considered as a read operation. Other statements, like INSERT, DELETE, and UPDATE are considered as write operations.

In the predicate locking [18], the logic expression of a given query is used as the lock granularity. In other words, rather than locking an individual object, a lock request can specify a subset of the database to which the lock applies. In general, a transaction, T , could issue a lock like the following: $\langle T, [\text{read_lock} \mid \text{write_lock}], \text{predicate} \rangle$. Here the predicate is any WHERE clause from an SQL statement. Two predicate locks $\langle t, \text{lock_mode}, p \rangle$ and $\langle t', \text{lock_mode}', p' \rangle$ are *compatible* if one of the following conditions holds: (1) $t = t'$, (2) both modes are READ, or (3) no object satisfies both predicates.

The major problem of predicate locking is high execution cost of checking whether two logic expressions are conflict. Note that the overhead for checking conflict between two arbitrary logic expressions is known to be NP-complete. In DIME, we modify the original predicate locking so that the overhead for checking conflict becomes rather cheap. The basic idea is to restrict the degree of concurrency supported by GTSCH. Specifically, we append the following three rules to the original predicate locking.

(Rule 1) If an expression includes a logic operator “or”, then it is conflict to others.

(Rule 2) If there are no common attributes in two expressions, then they are conflict.

(Rule 3) If an expression includes “ $a \text{ op } b$ ” where a and b are attributes of relations and op is an arithmetic operator, such as ‘<’, ‘>’, ‘=’, and so on, then it is conflict to others.

With Rule 1, it is not required to scan lengthy expressions each of which includes a number of conditions connected by one or more “or” operators. With Rule 2 and Rule 3, we can check compatibility of two expressions without accessing database to which the lock applies.

The GTSCH maintains two levels of information: relation level and operation level. As *relation-level*

information (RLI), the largest timestamp of committed transactions that had read and written the relation is maintained. RLI is used to check whether an incoming transaction obeys the timestamp ordering rule against committed transactions. As *operation-level information* (OLI), the timestamps of transactions and operation types (e.g. read or write) are maintained. OLIs are used to check whether the incoming transaction obeys the timestamp ordering rule or two-phase locking rule against active transactions. OLI of a transaction is upgraded to RLI when the transaction commits.

A transaction is scheduled with the type and the timestamp of the transaction. The transaction has a *locking-type* when a LDBS to which the transaction is submitted uses 2PL. Otherwise, the transaction has *timestamping-type*. When the GTSCH receives a transaction that tries to access a relation, the GTSCH checks whether the transaction is allowed to be executed without violating global serializability, using RLI and OLI. This validation is performed by the following steps.

- (1) If the type of incoming transaction coincides with the type of transactions that LCCS handles, the transaction is validated with the scheduling rule of LCCS as the following two substeps.
 - (1.1) If the type of LCCS is of **timestamp ordering**, the incoming transaction is validated on the basis of TO rule with RLI in order to guarantee that the transaction does not violate the rule against committed transactions. The transaction which succeeds in the above validation is further validated on the basis of TO rule with OLI in order to guarantee that the transaction does not violate the rule against active transactions.
 - (1.2) If the type of LCCS is of **locking**, the incoming transaction is validated on the basis of 2PL rule with the wound-wait protocol by using only OLI to guarantee that the transaction is not involved in any conflict or any deadlock with the other active transactions.
- (2) If the type of incoming transaction is different from that of LCCS, the transaction is certified on the basis of both TO rule and the 2PL rule with the wound-wait protocol. This certification is accomplished by performing both 1.1 and 1.2 above.

In DIME, to integrate existing LDBSs into a

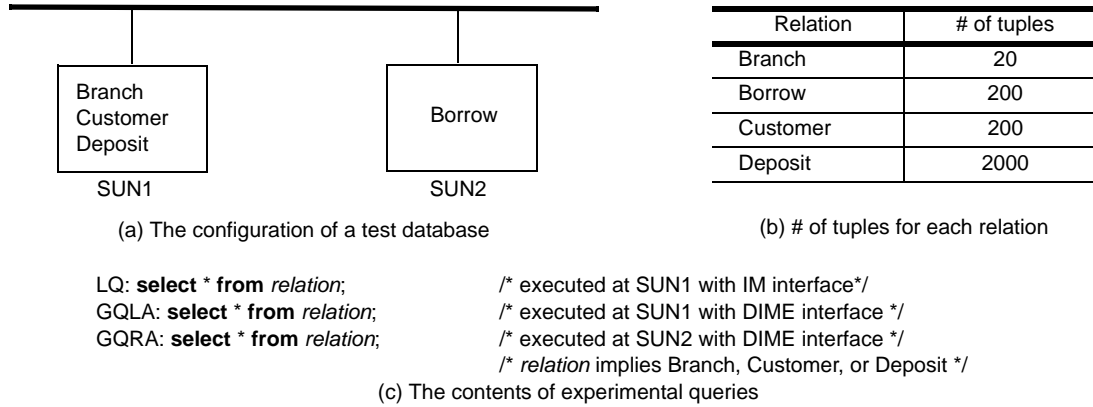


Figure 5: A test database and experimental queries

HDDBS, we only need to attach an GTSCH on top of each LDBS. That is, the heterogeneity between different LCCSs can be resolved by the global concurrency control scheme at each GTSCH. So DIME can guarantee the design autonomy of each local site. Since only the transaction that is known to be correct in serializability test is executed at the LDBS and it is terminated according to 2PC, there is no abortion of the local transaction that has already committed. Therefore, local commitment autonomy is also guaranteed in DIME. Finally, since local execution information is collected at GTSCH and the communication for coordinating global transaction is accomplished between the GTSCH and DTP, DIME can also guarantee the local communication autonomy.

6: Experimental results

In this section, we describe some of the tests that have been completed along with what we have learned from these tests. These tests include three types of queries: local query (LQ), global query with local access only (GQLA), and global query with remote access (GQRA). The queries are performed on two SUN SPARC2 workstations connected through Ethernet protocols with 10Mbps data transfer rate. Figure 5 shows the contents of test databases and the experimental queries.

This set of tests enables us to investigate the *interprocess communication overhead* and the *distribution overhead* of DIME. Table 1 shows the results of these tests with three sample relations, each of which has 20 tuples, 200 tuples, and 2000 tuples, respectively.

By comparing the performance of GQLA to that of

Table 1 Execution time of experimental queries

	Branch (20 tuples)	Customer (200 tuples)	Deposit (2000 tuples)
LQ	0.57s	1.26s	8.26s
GQLA	2.16s	4.37s	33.21s
GQRA	2.27s	4.50s	35.48s

LQ executed at local DBMS (IM), we can investigate the interprocess communication (IPC) overhead of DIME. Table 1 shows that the average execution time of GQLA is about fourth times as long as that of LQ. It results from the *similarities* between user application process (UAP) of DIME and query processor (QP) of IM. That is, the query processing steps are duplicated on UAP and QP, and additional messages should be transferred between them. One possible solution to this disadvantage is to eliminate UAP and to extend the role of QP such that QP can execute global queries. Since QP executes every type of queries, query processing can be performed efficiently and message overhead can be reduced. This modified system architecture, however, suffers from poor extensibility, because the system architecture is strongly dependent on the local DBMS, IM. Note that one of the objectives of DIME is to implement an open architectural distributed database system. This requires that the system architecture of DIME should be independent of local DBMS, even though there may be some performance degradation. So, we are currently developing an efficient implementation technique to reduce the performance degradation, while preserving the system architecture of DIME.

By comparing the performance of GQLA to that of GQRA, we can investigate the distribution overhead of DIME. Table 1 shows the execution time of GQRA is longer than that of GQLA about five percent. This implies that the additional overhead of GQRA on GQLA is the networking overhead, and it is trivial when the data transfer rate of the network is high.

7: Conclusions

We have described the status of the current DIME prototype. DIME is a heterogeneous distributed database system that supports a global SQL interface with distribution transparency, a global concurrency control scheme with full local site autonomy, and standard communication protocols. All the features described in this paper have been implemented and are operational. Specifically, DIME has been in operation on SUN SPARC2 workstations under SUN operating systems 4.1.x (a variant of UNIX) and IBM PC 486 machines under System V operating systems, connected through Ethernet protocols since February 1992.

To include more functionality in DIME, we are now in the stage of including deadlock resolution scheme and recovery scheme. Furthermore, we have a plan to integrate commercial relational DBMSs, such as INGRES, ORACLE, and Informix, into DIME. We will then derive a product from the prototype. This calls for a complete evaluation of the prototype, its interface, its architecture, its specific algorithms, and the quality of code. This evaluation will be done on the functional side by writing a number of applications using the system, and on the performance side by measuring and benchmarking it. Then we will rewrite whatever percentage of the code is necessary to build a robust and efficient prototype.

Acknowledgment

The authors should thank Won-Young Kim, Tae-Kyung Byun, and Ho-Woong Hwang for their assistance in the implementation of DIME's user application process and distributed transaction process.

References

1. M. Atkinson, et al., "The Proteus Distributed Database System," Proc. of the 3rd British National Conference on

- Databases, Leeds, U. K., July 1984, pp. 225-245.
2. Y. Breitbart and L. Tieman, "ADDS: Heterogeneous Distributed Database System," Distributed Data Sharing Systems, North-Holland, 1985, pp. 7-24.
3. C. Chung, "DATAPLEX: An Access to Heterogeneous Distributed Databases," CACM, Vol. 33, No. 1, Jan. 1990, pp. 70-80.
4. M. Deen and et al., "Implementation of a Prototype for Preci*," The Computer Journal, Vol. 30, No. 2, 1987, pp. 157-162.
5. ISO/CD 9579, *Information Processing Systems - Open System Interconnection - Remote Database Access*, Committed Draft of International Standard, 1991.
6. ISO/DIS 10026, *Information Processing Systems - Open System Interconnection - Distributed Transaction Processing*, Draft International Standard, 1992.
7. Y. Kim, "Atomic Transaction Scheduling in Tightly Coupled Heterogeneous Distributed Databases," Ph.D. Thesis, KAIST, May 1992.
8. V. Krishnamurthy and et al., "A Distributed Database Architecture for An Integrated Manufacturing Facility," Integration of Information Systems: Bridging Heterogeneous Databases, IEEE Press, 1989, pp. 179-190.
9. T. Landers and R. Rosenberg, "An Overview of MULTIBASE," Proc. of the 2nd International Symposium on Distributed Databases, North-Holland, 1982, pp. 153-183.
10. J.K.Ahn, et al., "Implementation of A Relational Database Management System IM," Database Review, Korea Information Science Society, Vol. 7, No. 2, 1991, pp. 21-41.
11. ISO/CD 9075, *Information Processing Systems - Open System Interconnection - Database Language SQL 2*, Committed Draft of International Standard, 1990.
12. ISO 9545, *Information Processing Systems - Open System Interconnection - Application Layer Structure*, International Standard, 1989.
13. M. Rose, *The ISO Development Environment: User's Manual*, May 1990.
14. ISO 8824, *Information Processing Systems - Open System Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*, International Standard, 1987.
15. B.Lindsay, "Object Naming and Catalog Management for a Distributed Database Manager," IBM Research Report RJ2914, San Jose, CA., Aug. 1980.
16. E.K.Hong, "Performance Evaluation of Catalog Architectures in Distributed Database Systems," Ph.D. Thesis, KAIST, Nov. 1990.
17. Y. Breitbart, "Multidatabase Interoperability," SIGMOD RECORD 19(3), Sept. 1990, pp.53-60.
18. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1993.