DISTRIBUTED DATABASES

INTRODUCTION

The development of network and data communication technology has resulted in a trend of decentralized processing in modern computer applications, which includes distributed database management. Naturally, the decentralized approach reflects the distributed organizational structure, allows the improved availability and reliability of data, and allows improved performance and easier system expansion.

We can define a distributed database (1,2) as a collection of data that belong logically to a single database but are stored physically in several databases over the sites of a network. Two important aspects in the definition of a distributed database exist. First, a distributed database is distributed physically within several databases called local databases on different sites of a network; this aspect distinguishes a distributed database from a centralized database. Second, a user may have an illusion that a distributed database is a single database (i.e., a virtual database called a global database); this aspect distinguishes a distributed database from a set of networked databases.

The fact that a distributed database is spread physically over several local databases, yet it is viewed logically as a whole brings challenging tasks for a distributed database management system (DDBMS), a software that is used to manage distributed databases. A distributed database system (DBS) consists of a DDBMS and the distributed databases that it manages. The key issue of a DDBS is the support of transparency. With transparency, users may access and update a distributed database through a single global schema by using an ordinary query language such as SQL in the same way as they do to a centralized database. Three fundamental tasks must be supported by a DDBS: distributed database design, distributed query processing, and distributed transaction management.

Apart from data distribution, heterogeneity and autonomy are two other aspects of a DDBS. In terms of heterogeneity, a DDBS may be classified as homogeneous or heterogeneous. A homogeneous DDBS has identical local DBMSs on all sites, whereas a heterogeneous DDBS allows differences in their local DBMSs. Sometimes, local DBMSs of a heterogeneous DDBMS may be of different types: relational, hierarchical, network, and object-oriented. In terms of autonomy, a DDBMS with high autonomy of local DBMSs is called a federated DBS (FDBS) or a mutidatabase system (3).

Date (4) has listed twelve rules for a DDBMS. They are as follows:

1. Local autonomy. The sites in a distributed system should be autonomous. In this context, autonomy indicates that local data is locally owned, local operations remain purely local, and all operations at a given site are controlled by that site.
2. No reliance on a central site. There should be no single site without which the system cannot operate.
3. Continuous operation. Ideally, a need should never exist for a planned system shutdown.
4. Location independence. The user should be able to access all data from any site as if it were stored at the user's site, regardless of where it is stored physically.
5. Fragmentation independence. The user should be able to access the data, regardless of how it is fragmented.
6. Replication independence. The user should be unaware that data has been replicated. Thus, the user should not be able to access a particular copy of a data item directly, nor should the user have to update all copies of a data item.
7. Distributed query processing. The system should be capable of processing queries that reference data at more than one site.
8. Distributed transaction processing. The system ensures that both transactions at local and global levels conform to ACID properties (i.e., atomicity, consistency, isolation, and durability).
9. Hardware independence. It should be possible to run the DDBMS on a variety of hardware platforms.
10. Operating system independence. It should be possible to run the DDBMS on a variety of operating systems.
11. Network independence. It should be possible to run the DDBMS on a variety of disparate communication networks.
12. Database independence. It should be possible to have a DDBMS that consists of different local DBMSs. In other words, the system should support heterogeneity.

Recently, we have observed the rapid development of Internet technology and the use of XML as a standard for data formatting and exchange on the Internet. The effective management and integration of huge amounts of XML data resources on the Internet brings new topics for distributed database management.

The rest of this paper is organized as follows: Three fundamental tasks of a DDBMS (i.e., distributed database design, distributed query processing, and distributed transaction management) are introduced in the first 3 Sections, respectively. In the next Section, we discuss the problems in FDBSs. In the final Section, we discuss distributed database related research topics on the Web and XML.
DISTRIBUTED DATABASE ARCHITECTURE AND DESIGN

The ANSI/SPARC three-level architecture provides a reference architecture for a centralized database. This architecture can be extended for a distributed database as shown in Fig. 1(5). Given that the global virtual database of a distributed database is used by end users, whereas local databases of the distributed database are actually used to store and manage real data, the architecture only has global external schemas and local internal schemas. The task of distributed database design is to map a global conceptual schema into a set of local conceptual schemas. Three steps can be followed, which result in three schemas, fragmentation schema, allocation schema, and local mapping schema. The fragmentation schema describes how global relations are partitioned into subrelations called fragments. The allocation schema describes on which site(s) a fragment is placed, which takes into account any replication. The local mapping schema maps fragments of allocation schema into relations in local conceptual schema. The local conceptual schema at each site defines the entire local database at the site, and the local internal schema is a physical level representation of a local database.

Fragmentation

Two types of fragmentation exist: horizontal and vertical. A horizontal fragment is a subset of tuples, and a vertical fragment is a subset of attributes of a global relation. Two rules must be followed during fragmentation: completeness—all the data of a global relation must be mapped into the fragments and reconstruction—a global relation must be able to construct from its fragments. For horizontal fragmentation, an additional rule must be followed, which is disjointness—no overlapping exits between any two fragments. For vertical fragmentation, the disjointness rule is allowed to be violated because a replicated attribute is required to reconstruct the global relation. The following is global schema with three global relations:

\[
\text{SALESPERSON}(\text{sid}, \text{name}, \text{commission}, \text{branch})
\]

\[
\text{CUSTOMER}(\text{cid}, \text{name}, \text{address})
\]

\[
\text{ORDER}(\text{oid}, \text{ordate}, \text{totamt}, \text{cid}, \text{sid})
\]

A global relation can be fragmented horizontally into fragments by using a selection operation \(\sigma\) and be reconstructed from its fragments by a union operation \(\cup\). For example, assume that every salesperson works in either Sydney or Melbourne but not in both branches, then we have the following:

\[
\text{SALESPERSON}_{\text{SYD}} = \sigma_{\text{branch} = \text{SYD}} \text{SALESPERSON}
\]

\[
\text{SALESPERSON}_{\text{MEL}} = \sigma_{\text{branch} = \text{MEL}} \text{SALESPERSON}
\]

\[
\text{SALESPERSON} = \text{SALESPERSON}_{\text{SYD}} \cup \text{SALESPERSON}_{\text{MEL}}
\]

\[
\text{SALESPERSON}_{\text{SYD}} \cap \text{SALESPERSON}_{\text{MEL}} = \emptyset
\]

A global relation can also be fragmented horizontally into fragments that depend on the horizontal fragmentation of
another global relation (derived horizontal fragmentation) by using a semi-join operation $\text{SJ}$. For example,

$\text{ORDER}_{\text{SYD}} = \text{ORDER} \bowtie \text{SALESPERSON}_{\text{SYD}}$

$\text{ORDER}_{\text{MEL}} = \text{ORDER} \bowtie \text{SALESPERSON}_{\text{MEL}}$

$\text{ORDER} = \text{ORDER}_{\text{SYD}} \cup \text{ORDER}_{\text{MEL}}$

$\text{ORDER}_{\text{SYD}} \cap \text{ORDER}_{\text{MEL}} = \emptyset$

A global relation can be fragmented vertically into fragments by using a projection operation $\pi$ and be reconstructed from its fragments by a natural join operation $\text{NJ}$. For example,

$\text{SALESPERSON}_{\text{COMM}} = \pi_{\text{sid}, \text{commission}} \text{SALESPERSON}$

$\text{SALESPERSON}_{\text{DETAIL}} = \pi_{\text{sid}, \text{name}, \text{branch}} \text{SALESPERSON}$

$\text{SALESPERSON}_{\text{SYD}} = \text{SALESPERSON}_{\text{COMM}} \bowtie \text{NJ} \text{SALESPERSON}_{\text{DETAIL}}$

Mixed horizontal and vertical fragmentations may be used for a global relation. Sometimes, a global relation may not need to be fragmented.

**Allocation**

After the fragmentation step, we have a set of fragments. The problem of allocation is how to distribute this set of fragments to the set of sites such that the distribution is optimal to a predefined set of dominant applications, which can be modeled as a set of retrieval and update references to fragments. Two basic alternatives to allocate fragments exist: nonredundant or redundant. The former places each fragment into a single site, whereas the latter may place a fragment into multiple sites. The general allocation problem is NP-hard (6). Therefore, the proposed solutions are based on heuristics.

**Levels of Transparencies**

From Fig. 1, a DDBMS may provide transparencies at different levels, which depend on users’ requirement. The highest level of transparency is the fragmentation transparency. At this level, users do not need to know that global relations are fragmented and where the fragments are placed. Therefore, global schema is used for any retrieval and update requests. The middle level of transparency is the location transparency. At this level, users must use fragments specified in the fragmentation schema for any retrieval and update requests. However, users do not need to know the locations of these fragments and how many copies of these fragments exist. The local mapping transparency is the lowest level of transparency. At this level, users must use the allocation schema to specify not only the fragment, but also which copy of the fragment on a given site. The only thing users may not know is how the fragment is represented in the local conceptual schema.

**DISTRIBUTED QUERY PROCESSING AND OPTIMIZATION**

In a distributed database, a global query is expressed as references to global relations defined in the global schema. A DDBMS must transform this global query into several subqueries; each subquery executes on a local database and then combines the results of subqueries to form the result of the global query (7). The set of subqueries, the queries for combining results of subqueries, and the order for executing these queries constitute a distributed query execution plan. For a global query, many such query execution plans exist. The task of the distributed query optimization is to find an optimal plan such that either minimum total cost or minimum response time for executing a global query is achieved. Unfortunately, finding such an optimal plan has been proved a NP-hard problem; therefore, most of the proposed solutions are based on heuristics.

Distributed query optimization is much more complicated than its centralized counterpart. For centralized systems, the primary factor for the cost of a particular execution plan is the cost of local processing. In a distributed system, more factors must be considered.

1. The distribution of data. As global relations are fragmented and allocated, possibly with more than one copy, to several sites as local relations. Much space exists to choose which copy of a fragment to use for a global query. This process is called materialization.

2. Communication cost. As data is spread over different sites of a network, data transmission between sites is inevitable. For wide area networks, the speed of data transmission is much slower than that of disk access. As such, communication cost becomes a dominant factor toward the measurement of cost of a global query.

3. Potential parallelism. As subqueries are executed by local DBMSs, it is possible to parallelize the processing of these subqueries, provided they are not dependant in the execution order. As such, performance gain can be achieved by parallelism. Exploring parallelism is especially important if minimum response time is selected as the criterion of optimization.

In addition, accurate database profiles are important to estimate the cost of operations and the size of intermediate relations. A database profile contains statistic information of the databases, such as the size of relations and attributes, the data distribution information, the selectivity of operations such as selection and join, and so on.

Several optimization strategies have been used for distributed query optimization. These strategies include transformation, semi-join based, and join based strategies.

**Query Transformation**

Rules are available that can be applied to a query (as a expression of relational algebra) to rewrite it into an equivalent expression, (1,8). Let $U$ and $B$ stand for unary and binary algebraic operations, respectively. We may have the following algebraic laws for some relational algebraic operations where $R$, $S$, and $T$ are relations.

- Commutativity: $U_1 U_2 R \leftrightarrow U_2 U_1 R$, $R B S \leftrightarrow S B R$.
- Associativity: $R (S B T) \leftrightarrow (R B S) T$.
- Idempotence: $U R \leftrightarrow U_1 U_2 R$. 

\[\begin{align*}
\text{ORDER}_{\text{SYD}} = \text{ORDER} \bowtie \text{SALESPERSON}_{\text{SYD}} \\
\text{ORDER}_{\text{MEL}} = \text{ORDER} \bowtie \text{SALESPERSON}_{\text{MEL}} \\
\text{ORDER} = \text{ORDER}_{\text{SYD}} \cup \text{ORDER}_{\text{MEL}} \\
\text{ORDER}_{\text{SYD}} \cap \text{ORDER}_{\text{MEL}} = \emptyset \\
\text{SALESPERSON}_{\text{COMM}} = \pi_{\text{sid}, \text{commission}} \text{SALESPERSON} \\
\text{SALESPERSON}_{\text{DETAIL}} = \pi_{\text{sid}, \text{name}, \text{branch}} \text{SALESPERSON} \\
\text{SALESPERSON}_{\text{SYD}} = \text{SALESPERSON}_{\text{COMM}} \bowtie \text{NJ} \text{SALESPERSON}_{\text{DETAIL}} \\
\text{ORDERSYD} = \text{ORDER} \bowtie \text{SALESPERSON}_{\text{SYD}} \\
\text{ORDERMEL} = \text{ORDER} \bowtie \text{SALESPERSON}_{\text{MEL}} \\
\text{ORDER} = \text{ORDER}_{\text{SYD}} \cup \text{ORDER}_{\text{MEL}} \\
\text{ORDER}_{\text{SYD}} \cap \text{ORDER}_{\text{MEL}} = \emptyset \\
\text{SALESPERSON}_{\text{COMM}} = \pi_{\text{sid}, \text{commission}} \text{SALESPERSON} \\
\text{SALESPERSON}_{\text{DETAIL}} = \pi_{\text{sid}, \text{name}, \text{branch}} \text{SALESPERSON} \\
\text{SALESPERSON}_{\text{SYD}} = \text{SALESPERSON}_{\text{COMM}} \bowtie \text{NJ} \text{SALESPERSON}_{\text{DETAIL}} \\
\end{align*}\]
A query can be represented as an operator tree where the leaf nodes of the tree are relations and nonleaf nodes are operators. Obviously, the operators closest to leaf nodes will be executed first, and the root operator will be executed last. The objective of query optimization based on equivalent transformation is to find an operator tree with the minimum cost for execution. In centralized databases, heuristics can be used for better performance of queries (e.g., use idempotence of selection and projection to generate appropriate selections and projections for each relation, push selections and projections down in the tree as far as possible).

In distributed databases, a global relation may be fragmented horizontally and/or vertically into fragments, and real data is stored in local relations that represent physical copies of those fragments. Therefore, a global relation in a global query must be replaced as union (horizontal fragmentation) and/or natural join (vertical fragmentation) of fragments. It is always beneficial to reduce the size of fragments before they are transmitted to other sites. Sometimes, while pushing selection down the tree to union of horizontal fragments, a contradictory qualification may be achieved for some fragments, which means no result will be obtained from those fragments. Consequently, those fragments can be removed from the query. Similarly, while pushing projection down the tree to join of vertical fragments, those fragments that do not contain the projected attributes can be removed from the query. To distribute joins that appear in the global query, unions that represent collections of fragments must be pushed up beyond the joins that we want to distribute. Although a join between two global relations with one horizontally fragmented depends the other, only joins between correspondent fragments are needed.

**Semi-Join Strategy**

A join operation is even more expensive in distributed databases than in centralized ones when data transmission cost is the dominant factor for query optimization. To reduce the cost of a join across sites, it is ideal to reduce the size of operand relations fully. A semi-join operator can be a reducer in most cases. The theory of semi-joins is well defined by Bernstein and Chiu (9).

Let R and S be two relations, in which A and B are the join attributes that belong to R and S, respectively, and SJ stand for semi-join, then \( R \ SJ_{A=B} S \) is a subset of tuples of R, constituted by those tuples that give a contribution to the join of R with S. The benefit of this is that the tuples that are not concerned with join will be filtered out before the real join operation.

A semi-join program for a join between R and S can be done by one of the following strategies:

1. \( R \ JN_{A=B} (S \ SJ_{B=A} \pi_A R) \).
2. \( S \ JN_{B=A} (R \ SJ_{A-B} \pi_B S) \).
3. \( (R \ SJ_{A-B} \pi_B S) \ JN_{A-B} (S \ SJ_{B=A} \pi_A R) \).

If R and S are from different sites and we take the first strategy, then the following program can be used to implement \( R \ JN_{A-B} S \).

1. Send \( \pi_A R \) to the site of S.
2. Compute \( S' = S \ SJ_{B=A} \pi_A R \) at the site of S.
3. Send \( S' \) to the site of R.
4. Compute \( R \ JN_{A-B} S' \) at the site of R.

The cost of the above semi-join program is the cost of step 1 and step 3, whereas the cost of a join-based algorithm is that of transferring relation S. The semi-join approach is better if \( \text{size}(\pi_A R) + \text{size}(S \ SJ_{B=A} \pi_A R) < \text{size}(S) \), i.e.,

\[
\text{size}(\pi_A R) < \text{size}(S) - \text{size}(S \ SJ_{B=A} \pi_A R)
\]

Notice that the right side of the above inequity is the reduced tuples of S. The semi-join approach is better if the semi-join acts as a sufficient reducer (i.e., if a few tuples of S participate in the join). The join approach is better if most of the tuples of S participate in the join, because the step 1 of semi-join requires additional cost.

The semi-join can be useful to reduce the size of the operand relations involved in a multiple-join query. The size of an operand relation may be reduced by more than one semi-join. For example, R in a multiple join query of operand relations R, S, and T can be reduced by \( R' = R \ SJ (S \ SJ T) \). Such a sequence of semi-joins is called a semi-join program for R. For an operand relation, several potential semi-join programs exist. One of these programs is optimal and it is called the full reducer. Given that the number of semi-join programs is exponential in the number of operand relations, the cost of the full reducer program is sometimes greater than the benefit. In the DDBMS prototype SDD-1, a semi-join based algorithm has been proposed (10) based on a hill-climbing algorithm for centralized query optimization.

**Join-Based Strategy**

In a DDBS in which data transmission cost is much more expensive than local processing cost, the use of semi-joins can improve the performance of a query significantly. If we also consider the cost of local processing to evaluate alternative execution plans, then the direct use of joins as a query processing tactic is often more convenient than the use of semi-joins. For example, \( R' \) query optimization algorithm (11) uses joins rather than semi-joins. It uses a compilation approach in which an exhaustive search of all alternative execution plans is performed to choose one with the least cost. Both data transmission and local processing costs are considered in Ref. 11.

**DISTRIBUTED TRANSACTION MANAGEMENT**

The objectives of distributed transaction management are the same as those of centralized transaction management [i.e., the guarantee of ACID properties (12–14)]:

- **Distributivity:** \( U(R \ B \ S) \rightarrow U(R) \ B \ U(S) \).
- **Factorization:** \( U(R) \ B \ U(S) \rightarrow U(R \ B \ S) \).
Atomicity requires that either all or none of the transaction’s operations be performed. In other words, if a transaction fails to commit, its partial results cannot remain in the database.

Consistency requires that a transaction to be correct. In other words, if a transaction is executed alone, it takes the database from one consistent state to another. When more than one transaction is executed concurrently, the database management system must ensure the consistency of the database.

Isolation requires that an incomplete transaction cannot reveal its results to other transactions before its commitment. This function can avoid the problem of cascading abort (i.e., the necessity to abort all the transactions that observed the partial results of a transaction that was later aborted).

Durability means that once a transaction has been committed, all the changes made by this transaction must not be lost even in the presence of system failures.

Two types of transactions we need to consider in a distributed database system are local and global transactions. A local transaction may access and update data in only one local database, whereas a global transaction may access and update data in several local databases. Thus, a global transaction consists of a set of subtransactions, each of which involves data residing on one site. A transaction manager at each site ensures ACID properties of local transactions as well as subtransactions at that site. For global transactions, the task is much more complicated, because several sites may be participating in execution. The concurrent global transactions must be serializable and recoverable in the distributed database system. In consequence, each subtransaction of a global transaction must be either performed in its entirety or not performed at all.

Serializability in a Distributed Database

It is well understood that the maintenance of the consistency of each single database does not guarantee the consistency of the entire distributed database. It follows, for example, from the fact that serializability of executions of the subtransactions on each single site is only a necessary (but not sufficient) condition for the serializability of the global transactions. To ensure the serializability of distributed transactions, a condition stronger than the serializability of single schedule for individual sites is required.

In the case of distributed databases, it is relatively easy to formulate a general requirement for correctness of global transactions. The behavior of a distributed database system is the same as a centralized system but with distributed resources. The execution of the distributed transactions is correct if their schedule is serializable in the whole system. The equivalent conditions are as follows:

- Each local schedule is serializable.
- The subtransactions of a global transaction must have a compatible serializable order at all participating sites.

The last condition indicates that for any two global transactions Gi and Gj, their subtransactions must be scheduled in the same order at all the sites on which these subtransactions have conflicting operations. Precisely, if Gi and Gj belong to Gi and Gj, respectively, and the local serializable order is Gi at site k, then all the subtransactions of Gi must precede the subtransactions of Gj at all sites where they are in conflict.

Various concurrency control algorithms such as two phase locking (2PL) (15,16) and timestamp ordering approaches (17,18) have been extended to distributed database systems. Because the transaction management in a distributed database system is implemented by several identical local transaction managers, the local transaction managers cooperate with each other for the synchronization of global transactions. If the timestamp ordering technique is used, a global timestamp is assigned to each subtransaction and the order of timestamps is used as the serialization order of global transactions. If a 2PL algorithm is used in the distributed database system, the locks of a global transaction cannot be released at all local sites until all the required locks are granted. In distributed systems, the data item might be replicated. The updates to replicas must be atomic (i.e., the replicas must be consistent at different sites). The following rules may be used to lock with n replicas:

- Writers need to lock all n replicas, readers need to lock one replica.
- Writers need to lock all m replicas (m > n/2), readers need to lock n – m + 1 replicas.
- All updates directed first to a primary copy replica (one copy has been selected as the primary copy for updates first and then the updates will be propagated to other copies).

Any one of the above rules will guarantee consistency among the duplicates.

Atomicity of Distributed Transactions

In a centralized system, transactions can either be processed successfully or be aborted with no effects left on the database in the case of failures. Normally, the failures cause loss of volatile or nonvolatile storage data. In a distributed system, however, additional types of failure may occur.

For example, network failures or communication failures may cause network partition, and the messages sent from one site may not reach the destination site. If a partial execution of a global transaction at a partitioned site existed in a network, it would not be easy to implement the atomicity of a distributed transaction. To achieve an atomic commitment of a global transaction, it must be ensured that all of its subtransactions at different sites are capable and available to commit. Thus, an agreement protocol must be used among the distributed sites. The most popular atomic commitment protocol is the two phase commitment (2PC) protocol.
In the basic 2PC, the site where a global transaction is issued serves as a coordinator. The participating sites that execute the subtransactions must commit or abort the transaction unanimously. The coordinator is responsible to make the final decision to terminate each subtransaction. The first phase of 2PC is to request from all participants the information on the execution state of subtransactions. The participants report to the coordinator, who collects the answers and makes the decision. In the second phase, that decision is sent to all participants. In detail, the 2PC protocol proceeds in two phases for a global transaction Ti (1).

Phase 1. Obtaining a Decision.

1. Coordinator asks all participants to prepare to commit transaction Ti:
   a. add [prepare Ti] record to the log
   b. send [prepare Ti] message to each participant
2. When a participant receives [prepare Ti] message it determines if it can commit the transaction:
   a. if Ti has failed locally, respond with [abort Ti]
   b. if Ti can be committed, send [ready Ti] message to the coordinator.
3. Coordinator collects responses:
   a. all respond ready, decision is commit
   b. at least one response is abort, decision is abort
   c. at least one fails to respond within time-out period, decision is abort.

Phase 2. Recording the Decision in the Database.

1. Coordinator adds a decision record ([abort Ti] or [commit Ti]) in its log.
2. Coordinator sends a message to each participant informing it of the decision (commit or abort).
3. Participant takes appropriate action locally and replies done to the coordinator.

The first phase is that the coordinator initiates the protocol by sending a prepare-to-commit request to all participating sites. The prepare state is recorded in the log and the coordinator is waiting for the answers. A participant will reply with a ready-to-commit message and record the ready state at the local site if it has finished the operations of the subtransaction successfully. Otherwise, an abort message will be sent to the coordinator and the subtransaction will be rolled back accordingly.

The second phase is that the coordinator decides whether to commit or abort the global transaction based on the answers from the participants. If all sites answered ready-to-commit, then the global transaction is to be committed. The final decision-to-commit is issued to all participants. If any site replies with an abort message to the coordinator, the global transaction must be aborted at all the sites. The final decision-to-abort is sent to all the participants who voted the ready message. The global transaction information can be removed from the log when the coordinator has received the completed message from all the participants.

The basic idea of 2PC is to make an agreement among all the participants with respect to committing or aborting all the subtransactions. The atomic property of global transaction is then preserved in a distributed environment.

The 2PC protocol is subject to the blocking problem in the presence of site or communication failures. For example, suppose that a failure occurs after a site has reported ready-to-commit for a transaction, and a global commitment message has not yet reached this site. This site would not be able to decide whether the transaction should be committed or aborted after the site is recovered from the failure. Three phase commitment (3PC) protocol (19) has later been introduced to avoid the blocking problem. But, 3PC is too expensive.

The 2PC protocol is used not only in distributed databases, but also in parallel databases for transactions, which contain subtransactions to be executed in different partitions of a parallel database (20).

FEDERATED DATABASE SYSTEMS

A federated database system (FDBS) is a collection of cooperating but autonomous database systems called component DBSs that are integrated to various degrees (3). The software that provides controlled and coordinated manipulation of the component DBSs is called a federated database management system (FDBMS). A component DBS in an FDBS can participate in more than one federation. A multidatabase system (MDBS) differs from an FDBS in that only a single federation schema is defined for a multidatabase. The DBMS of a component DBS, or component DBMS, can be a centralized or distributed DBMS or another FDBMS. Several significant aspects of an FDBS are as follows:

1. Local autonomy. Component DBSs are often under separate and independent control. Those who control a database are often willing to let others share the data only if they retain control. A component DBS can continue its local operations and can participate in a federation at the same time. Normally, no difference to a component DBS exists between a local application or a global applications at federated levels.

2. Heterogeneity. Usually, component DBMSs are different; they can differ in such aspects as data models, query languages, and transaction management capabilities.

3. Pre-existing distribution. Usually, multiple component DBSs are built before an FDBS is built. Therefore, discrepancy in semantics and conflicts may exist among those component databases.

Schema Architecture and Design

Figure 2 shows a five-level schema architecture of a FDBS proposed by Sheth (3).
Local Schema. A local schema is the conceptual schema of a component DBS. A local schema is expressed in the native data model of the component DBMS; and hence, different local schemas may be expressed in different data models.

Component Schema: A component schema is derived by translating local schemas into a data model called the canonical or common data model (CDM) of the FDBS. Two reasons for defining component schemas in a CDM are 1) they describe the divergent local schemas using a single representation and 2) semantics that are missing in a local schema can be added to its component schema.

Export Schema. Not all data of a component DBS may be available to the federation and its users. An export schema represents a subset of a component schema that is available to the FDBS. It may include access control information regarding its use by specific federation users. The purpose of defining export schemas is to facilitate control and management of association autonomy.

Federated Schema. A federated schema is an integration of multiple export schemas. A federated schema also includes the information on data distribution that is generated when integrating export schemas. Some systems use a separate schema called a distribution schema or an allocation schema to contain this information. Multiple federated schemas may exist in an FDBS, one for each class of federation users. A class of federation users is a group of users and/or applications who perform a related set of activities.

External Schema. A subschema or a view defined over a federated schema primarily for a pragmatic reason of not having to define too many federated schemas or to tailor a federated schema for smaller groups of federation users than that of a federated schema.

As component databases normally pre-exist in a FDBS, we can take a bottom-up design approach for federated databases. This approach is in contrast to the top-down design approach discussed in a previous section. The major tasks of the bottom-up design are schema translation and schema integration.

Schema Translation. As local schemas of different component databases may be defined in different data models, the specification of a CDM for defining federated schemas is required. Relational data model and object-oriented data model are often chosen as a CDM. Mapping rules must be studied between data models (e.g., relational model, DBTG or network model, hierarchical model, object-oriented model, and more recently XML data model).

Schema Integration. After schema translation, component schemas are generated for component databases. After that, export schemas are generated from component schemas for integration to different federated schemas. Four steps can be followed for schema integration.

Pre-Integration. Pre-integration is required to establish the rules of the integration process before actual integration occurs. For example, candidate keys in each schema must be identified; equivalent domains of attributes must be described in terms of mappings from one representation to another.

Comparison. During this phase, both the naming and the structural conflicts are identified. Naming conflicts include synonym (two identical entities or attributes with different names) and homonym (two different entities or attributes with the same name). Structural
conflicts include 1) **type conflicts**: the same object is represented by an attribute in one schema and by an entity in another, 2) **dependency conflicts**: different relationship types are used to represent the same thing in different schemas (1:m vs m:n), 3) **key conflicts**: different candidate keys are available and different primary keys are selected in different schemas, and 4) **behavioral conflicts** are implied by the modeling mechanism (e.g., deletion of the last employee causes the dissolution of the department).

- Conformation. Conformation is the resolution of the conflicts that are determined at the comparison phase.
- Merging and Restructuring. All schemas must be merged into a single database schema and then restructured to create the best federated schema.

### Global Query Processing and Optimization

In a loosely coupled FDBS, the FDBMS can support little or no query optimization. In a tightly coupled FDBS, the FDBMS can perform extensive query optimization. Query processing involves converting a query against a federated schema into several queries against the export schemas and executing these queries. Query processing in an FDBMS is similar to that in a distributed DBMS. In an FDBMS, however, several additional complexities may be introduced because of heterogeneity and autonomy. The cost of performing an operation may be different in different component DBSs. The component DBMSs may differ in their abilities to perform local query optimizations. The system and database operations provided by each of the component DBMSs and the FDBMSs may be different. Landers and Rosenberg (21) discuss optimization problems and solutions adopted for some of the above issues in Multibase.

### Global Transaction Management

Supporting global transaction management in an environment with multiple heterogeneous and autonomous component DBSs is very difficult. The challenge is to permit concurrently global updates to the underlying databases without violating their autonomy. Two types of transactions to be managed exist: **global transactions** submitted to the FDBMS by federation users and **local transactions** submitted directly to a component DBMS by local users. The basic problem in supporting global concurrency control is that the FDBMS does not know about local transactions because a component DBMS is autonomous. That is, local wait-for relationships are known only to the transaction manager of the component DBMS. Without knowledge about local as well as global transactions, it is highly unlikely that efficient global concurrency control can be provided. Because of the existence of local transactions, it is very difficult to recognize when the execution order differs from the serialization order at any site (22). Additional complications occur when different component DBMSs and the FDBMS support different concurrency control mechanisms (23). Georgakopoulos et al. (24) proposed to incorporate additional data manipulation operations on **tickets** in the subtransactions of each global transaction and show that if these operations create direct conflicts between subtransactions at each participating component DBS, indirect conflicts can be resolved even if the FDBS is not aware of their existence. However, all the published solutions often make unrealistic and pessimistic assumptions, or support a low level of concurrency or sacrifice autonomy to obtain higher concurrency. It is unlikely that a theoretically elegant solution exists that provides conflict serializability without sacrificing performance (i.e., concurrency and/or response time) and availability.

Work on weaker consistency criteria (25) and advanced transaction models (26) provide techniques to specify and to execute transactions that provide ACID properties selectively. A concept of **S-Transactions** (27) is proposed for semantic transactions suited for a banking environment that consists of a network of highly autonomous systems. It may be desirable to devise solutions that do not meet the conflict serializability criteria but that are practical and meet a desired level of consistency. Du and Elmargamid (22) propose a weaker consistency criterion called **Quasi Serializability** that works if no value dependencies (e.g., referential integrity constraints) exist across databases. Garcia-Molina and Salem (28) propose a concept of **Sagas** that provides semantic atomicity but does not serialize execution of global transactions.

### THE WEB AND DISTRIBUTED DATABASES

The last decade has seen the emergence of the Web as the central forum for data storage and exchange. More recently, XML has been proposed as a standard for data exchange and storage. Compared with the relational data model, the de facto standard for database systems and its structural primitives for building trees of elements with attributes offer much more flexibility in data organization and format. The Web and XML provide many avenues for database researchers. The Web bears a similarity to a bottom-up designed federated database in terms of integrating structured data sources from different websites; however, the Web is much more loosely coupled. In the following, we address two areas that are relevant to distributed databases.

### Information Integration on the Web

Data integration is a pervasive challenge faced in applications that need to query across multiple autonomous and heterogeneous data sources (29). Data integration is crucial in large enterprises that own a multitude of data sources. Integrating information from data resources over the Internet requires creating some form of integrated view to allow for distributed querying. The context of the Internet raises several issues for information integration that are far more difficult than those of multibase systems (3). First, the number of data sources may be very high, which makes view integration and conflict resolution a problem. Second, the space of data resources is very dynamic, so adding or dropping a data source should be done with minimal impact on the integrated view. Third, the data sources may have different computing capabilities, which range from full-featured...
DBMS to simple files. This feature is unlike multidatabase systems, which assume data sources with an SQL-like interface. Finally, data sources may be unstructured or semi-structured, which provides virtually no information for view integration. To address these problems, the database research community has revisited the multidatabase architecture (i.e., the architecture of a FDBS with a single federation schema) with data source wrappers and mediators. For each data source, a wrapper exports some information about its source schema, data, and query capabilities (30). For the whole integration system, a mediator centralizes the information provided by the wrappers in a unified global view of all available data, decomposes global queries into subqueries executable by wrappers on data sources, and gathers the partial results and computes the answer to the global queries. This wrapper–mediator architecture differs from a data warehouse in that integrated global view is not materialized.

Two basic approaches exist in data integration, GAV and LAV (31–33). These two approaches have also been used in the context of data integration on the Web. GAV (Global As View) defines a global schema as a view over a set of source schemas, whereas LAV (Local As View) defines source schemas as views over the global schema. GAV has been used in FDBSs and multidatabase systems, in which the quality depends on how well we have compiled the sources into the global schema through mapping. Whenever a source changes or a new source is added, the global schema must be reconsidered. Query processing can be based on some sort of rewriting. Each element in the user’s query corresponds to a substitution rule just as each element in the global schema corresponds to a query over the source. Query processing is simply expanding the subgoals of the user’s query according to the rule specified in the mediator, and thus the resulting query is likely to be equivalent. LAV has high modularity and reusability. Once the global schema is well designed, changes on a source only affect the definition of the source. The quality depends on how well we have characterized the sources. In LAV systems, queries undergo a more radical process of rewriting because a mediator does not exist. The integration system must execute a search over the space of possible queries to find the best rewrite. The resulting rewrite may not be an equivalent query but maximally contained, and the resulting tuples may be incomplete.

Recently, Semantic Web (34) has attracted great attentions from both research communities and standard organizations. Semantic Web is supposed to be an extension of the Web where the semantics of data is available to and processable by machines. At the core of this new technology are the languages that are used to describe the semantics of XML documents and ontology, such as RDF, DAML+OIL, and OWL. Ontology can be used to define global schemas, which makes information integration on the Web easy.

**Publishing Relational Data on the Web**

Although XML is emerging as the universal format to publish and to exchange data on the Web, most business data is still stored and maintained in relational database systems. As a result, an increasing need exists to publish relational data efficiently as XML documents for Internet-based applications. One approach to publish relational data is to create XML views of the underlying relational data. Through the XML views, users may access the relational databases as though they were accessing XML documents. Once XML views are created over a relational database, queries in an XML query language like XML-QL or XQuery can be issued against these XML views for the purpose of accessing relational databases. SilkRoute (35) is one of the systems that takes this approach. In SilkRoute, XML views of a relational database are defined using a relational to XML transformation language called RXL, and then XML-QL queries are issued against these views. The queries and views are combined together by a query composer and the combined RXL queries are then translated into the corresponding SQL queries. XPERANTO (36) takes a similar approach which uses XQuery for user queries. DTD directed publishing is introduced in Ref. 37 where an attribute translation grammar (ATG) is designed to creating XML views of relational databases. Another approach (38) to publish relational data is to provide virtual XML documents for relational data via an XML schema that is transformed from the underlying relational database schema such that users can access the relational database through the XML schema. In this approach, the process of XML schema generation preserves integrity constraints of the underlying relational schema, which makes a difference compared with the view approach taken by SilkRoute.

**BIBLIOGRAPHY**


CHENGFEI LIU
Swinburne University of Technology
Melbourne, Australia

YANCHUN ZHANG
Victoria University of Technology
Melbourne, Australia

10 DISTRIBUTED DATABASES