Distribution Problems in Distributed Database Systems: Integration and Query Decomposition

MAKOTO TAKIZAWA*

This paper presents the solutions to some of the distribution problems in distributed databases of a bottom-up type. They are a language for integrating local relations into global relations, and an algorithm for decomposing global queries into local queries and executing inter-site joins. The language which is called GSDL is an extension of QUEL so as to express the union of relations. Our query decomposition algorithm purposes to keep directory information as small and static as possible, because their maintenance is most important from the operational viewpoint. To achieve this goal, the scheduling of transmission of local relations is decided dynamically not in an off-line fashion.

1. Introduction

Distributed database systems of a bottom-up type aim at cooperating existing database systems through computer networks. There are two problems in designing them: how to solve the heterogeneity of the existing databases, and how to intergrate databases with different semantics into a logical database system. We call the former a heterogeneity problem and the latter a distribution one [TAKIM78]. The database is composed of several layers [TSICD78] and each layer can be characterized by one data model and language based on it. The databases connected by the network have to communicate with each other through one of their layers. Therefore, they are assumed to be black boxes, each of which provides one schema and language based on a data model.

There are two aspects in considering the heterogeneity of databases. One is the data model and language provided. It is called a syntactic aspect of the database. The other is called a semantic aspect, which represents the meaning of the database. Therefore, the heterogeneity is defined as the differencies of these two aspects. The heterogeneity problem, therefore, is how to solve the differencies of these syntactic aspects, and the distribution one is how to overcome the differencies of these semantic aspects.

Databases in a bottom-up distributed database system are different with respect to both syntactic and semantic aspects. The syntactic aspect, i.e. a data model and language, is considered as a means of description of and access to the semantic aspect. Hence, first, data models and languages have to be homogenized. Then, the semantic aspect described in terms of a common model can be integrated into one logical data description. We call the former homogenization and the latter integration. Our approach called a four-schema structure is based on such designing process [TAKIM 78, 79]. This is similar to five-schema approach by [ADIBM78].

The four-schema structure as shown in Fig. 1.1 consists of four schema-layers, mappings among the layers, and network data directory. There are four schemas, local internal (LIS), local conceptual (LCS), global conceptual (GCS), and external (EXS) schemas. The LIS corresponds to a schema or subschema of an

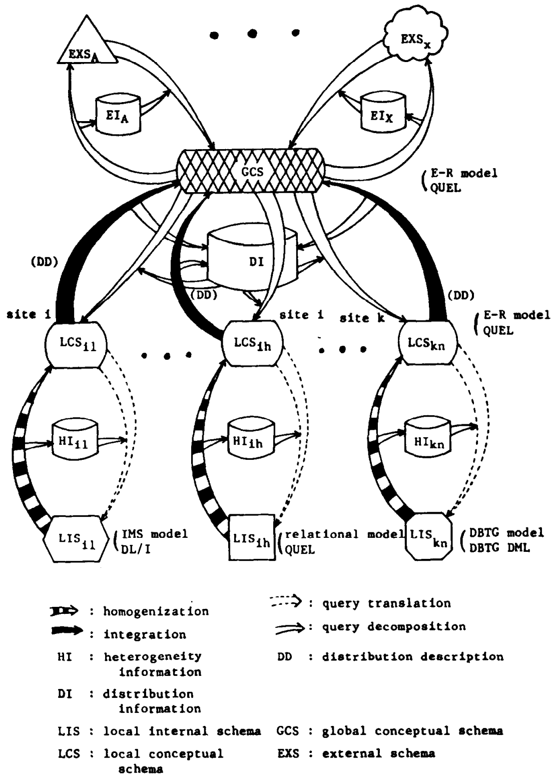

Fig. 1.1 Four-Schema Structure (FSS).

*Development Department, Japan Information Processing Development Center (JIPDEC).

existing DBMS, which describes data usable under network situation. The LCS is a description of the LIS in terms of an E-R model [CHENP76]. The GCS is an E-R model description of data integrated from the LCSs. At both LCS and GCS levels, QUEL [HELDG75] is provided. The EXS is a description of data of interest to an application in terms of a data model suited for it. We do not discuss it, because at the GCS level the distributed databases are virtualized as one database.

The mapping of the LCSs to the GCS is integration and its inverse mapping is a query decomposition. It is a process where a query based on the GCS is decomposed into a sequence of queries based on the LCSs using information (call it distribution information) generated in the integration. Both queries are written in QUEL. The mapping of the LIS into the LCS is homogenization and its inverse is a query translation. It is a process where such decomposed queries are translated into a sequence of DMLs executable on the database using information (call it heterogeneity information) generated in the homogenization [TAKIM79, 80]. The distribution and heterogeneity information together are called the network data directory. The latter exists at its corresponding site, and the former at every site.

A reason for adopting the E-R model as a common model is that it provides the concepts of entities and relationships among them in a simple manner, and can be easily related to a relational model [CODDE70]. A major advantage of the relational model is its simplicity of description and access to data, its semantic problems are pointed out. The more semantics the data model provide, the more complex the description and access become. Therefore, we employ the E-R model as a means of designing the distributed database because of its describability of semantics, and the relational model as a means of describing and accessing data because of its simplicity. This means that the LCSs and GCS are described really in a relational form. Such descriptions are called relational descriptions of schemas, by which the distributed database can be accessed, based on the E-R model description of these schemas.

We have discussed the heterogeneity problem in [TAKIM79, 80]. So, we would like to concentrate on the distribution problem. The integration is presented in Ch. 2. The query decomposition is discussed in Ch. 3–6. Especially, in Ch. 5, our query decomposition algorithm based on dynamic decision is proposed. In Ch. 7, the system architecture for query decomposition is shown.

2. Integration

The integration is a process for defining GCS relations from existing LCS relations distributed over a network. It is also similar to the definition of views [STONM76] in a relational model. In designing a relational database, one universal relation is vertically decomposed into normalized relations. Views can be defined by means of joins as multirelational operations. But in bottom-up designing, relations already exist at each site. This implies that unions are also required. Unfortunately, relational calculus languages like QUEL do not provide such capabilities. QUEL is extended so that the unions can be taken in the definition of the GCS relations.

The extension consists of three kinds of statements, drange, define, and drop. The drange is used to define tuple variables against the LCS relations along with their existing sites:

drange $(x_1, \cdots, x_m)(X_1 : s_1, \cdots, X_m : s_m)$;
where each x_i for $i = 1, \cdots, m$ stands for a tuple variable ranging over an LCS relation X_i at site s_i.

The define is used to define a GCS relation:

define⟨grelname⟩(⟨gatt-list⟩)⟨sub-def⟩{:⟨sub-def⟩};
⟨grelname⟩ and ⟨gatt-list⟩ define a scheme of a GCS relation to be defined. ⟨sub-def⟩ is called a subdefinition of the GCS relation:

⟨sub-def⟩: :=(⟨target-list⟩) where ⟨qual⟩
The target list and qualification are the same as QUEL. The ⟨sub-def⟩ defines a join of relations like QUEL. The list of subdefinitions delineated by colons means that the GCS relation is the union of results each of which is derived with respect to each subdefinition.

The drop is used to remove the defined GCS relation from the GCS.

In order to integrate LCS relations into a GCS relation, the LCS relations have to share the common union-compatible parts. The relationships are expressed by set-theoretical expressions on semantic links as shown in Fig. 2.1. For example, the expression PRJ2 [pno, pname] $\sqsubset$ PROJ [no, name] shows that both projections are union-compatible and PROJ [no, name] contains the same value set as PRJ2 [pno, pname]. PRJ1 $\bar{\cup}$ PRJ2 indicates that both are union-compatible. Fig. 2.2 shows the definition of a GCS relation PROJECT by means of the GSDL based on the semantic links as shown in Fig. 2.1. Such definitions of GCS relations are stored as the distribution information. This information is stored

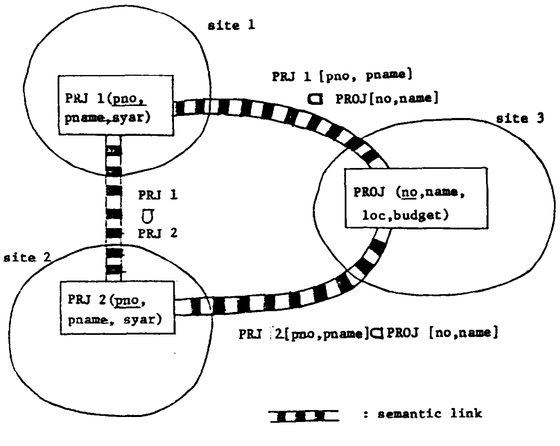

Fig. 2.1 The semantic links between three LCS relations.

```
drange (p1, p2, p) (PRJ1: 1, PRJ2: 2, PROJ: 3);
define ESR PROJECT (no, name, budget, location)
  (p. no, p. name, p. budget, location=p. loc)        }
  where p. no=p1. pno and p. name=p1. pname:          } (1)
  (p. no, p. name, p. budget, location=p. loc)        }
  where p. no=p2. pno and p. name=p2. pname;          } (2)
```

Fig. 2.2 The definition of the GCS relation PROJECT.

redundantly at every site and used to process the query decomposition.

3. Query Decomposition

The query decomposition process is composed of two subparts: the decomposition of GCS relations referenced by a GCS query into corresponding LCS relations, and the processing of inter-site joins. Query modification technique [STONM76] can be adopted for doing the first. Since the network causes a bottleneck of the distributed database system, the question is how to process the second efficiently is very significant in the query decomposition. This problem is closely related to a way of designing. In the case of top-down designing, data can be allocated to sites so that query processings are localized. Furthermore, sizes of intermediate results can be estimated more easily. However, in the case of a bottom-up design, since data have existed already independently of applications' requirements, more inter-site processings may be required and it is difficult to estimate the sizes of intermediates. [MAHMS79, ADIBM80] also proposes query decomposition algorithms based on dynamic decisions. [MAHMS79] uses the graph representation of the query. In his graph, the conjunctive and disjunctive relationships between links among the nodes are confused. That is, the relationships between the graph and query is not clear. By defining the query graph in which the conjunction of links represents the query qualification, we can make clear the meaning of the query graph. [ADIBM80] uses the pipeling method of tuple transmissions based on the binary tree representation of the query. But, since his method is essentially based on the user defined sequence of algebraic operations, it is still a problem to determine how much the transmission cost is optimized by his algorithm. Our algorithm has the a possibility of being optimized because it is based on the non-procedural representation of the query, i.e. query graph. Furthermore, we make clear the feasible management mechanism of our algoritm under actual situation, and also show the system architecture (in Ch. 7).

3.1 Basic Assumptions

We make the following assumptions [HEVNA78] on the network:

a) It is a site-to-site type.

b) It is always lightly loaded. Therefore, there is no need for considering queueing delay.

c) Local processing costs can be ignored compared with communication costs. An additional assumption is made.

d) Communication cost depends on distance, and time. A logical cost, LC_{ij}, is defined as a delay time for transmitting a packet from site i to j. It is also proportional to the number of hops between them.

We also make the following assumptions on processings at sites:

a) Each site has a working space managed in a relational form. Relations transmitted from the other sites and results of joins are stored in it.

b) Each site has two kinds of logical processors: a global database processor (GDP) and local database processors (LDPs) [TAKIM79]. The GDP plays a role of the query decomposition, integration, and management of the distribution information and working space at each site. Especially, the GDP to which a GCS query is issued is called a coordinate GDP (CGDP) that is a centralized controller for processing the query. The LDP is responsible for the query translation, homogenization, and management of the heterogeneity information, and exists against one database. LDPs which support data required by the GCS query are cooperated under the CGDP's control.

Data replication is an important concept for availability and reliability purpose [ROTHJ77]. However, in a bottom-up system, no replication can be assumed. As stated in Ch. 2, the existence of replication rather means that the semantic link between LCS relations containing such replication exists.

3.2 Objectives

The query decomposition purposes largely to generate an optimal sequence of transmissions of relations. The objectives which have been taken up [HEVNA78] are to minimize the communication cost and the response time. The network causes a bottleneck of the distributed database system due to its restricted capability. On the other hand, each site has some processing capacities. Therefore, in order to achieve these objectives, it is necessary to reduce the network traffic for query processing and process queries in parallel at multiple sites. Works which have been done so far [CHUW79, HEVNA78, EPSTR78, WONGE77] aim at achieving the objectives. Their characteristic is that strategies for transmitting relations are determined by estimating sizes of intermediates in an off-line manner. This estimation is based on statistics on relations such as selectivities and cardinalities. The selectivity [HEVNA 78, SELIP 79] of an attribute is defined as the ratio of the attribute cardinality to the cardinality of the relation to which it belongs. This definition can be true under an assumption that values of the attribute are uniform-distributed. But we think that there are still problems whether actual distribution of values follows well this assumption. In order to make the selectivity more precise, [CHUW79] proposes a method such that selectivities of values which are used frequently are accumulated in the directory each time the values are accessed. However, it is noted that the more precise statistics on selectivities we have, the more storages are required.

We argue the following points. First, the performance information like selectivities and cardinalities have dynamic properties compared with the schema information. Secondly, the query decomposition and its required information, i.e. the distribution information, have to exist at the same site for efficiency. It implies that every site has to provide a full copy of the distribution information. If each site provides such dynamic information redundantly, the overhead for not only storing but also controlling consistency of and concurrent access to them becomes serious and enormous [BERNP80]. Besides the objectives as stated above, therefore, we would like to add on more objective, i.e. to keep the information required by the query decomposition as small and static as possible.

We can summarize these discussions as follows. First, if every site has the facility of the query decomposition, the distribution information should not include the performance information like selectivities. Secondly, we cannot obtain necessary and sufficient performance information to decide strategies in an off-line manner. Therefore, we would like to propose an algorithm such that strategies are decided operationally and information required are kept static and small.

3.3 Strategies

Our strategies for processing inter-site queries are as follows. First, the CGDP decides consequent stages dynamically based on the statistics of result relations of the preceding stages. This results in small and static distribution information. Here, the stage is defined as the unit of inter-site processings, i.e. the pair of a transmission of a relation and its join.

Secondly, the query parts referencing only one site are processed locally at the site before the inter-site parts are processed, because the local processings are neglectable in cost and result in the reduction of sizes of relations to to transmitted.

Thirdly, if two relations at different sites are to be joined, the smaller one is transmitted to the larger through a path with minimum transmission cost. We do not consider strategies such that two relations at different sites are transmitted to an other site and joined there.

Lastly, if all the strages issued by the CGDP are not complete, if there exists a relation not being processed and a path with transmission cost less than some threshold value, then it can be transmitted through the path. Hence, more than one stage can be processed in parallel.

3.4 Query Normalization

A qualification part of a GCS query is generally written in an arbitrary boolean combination form of comparison predicates. First, it is normalized in a disjunctive normal form. Next, the normalized query is

range $(1_1, \cdots, 1_m)(L_1: s_1, \cdots, L_m: s_m)$;
retrieve into R $(r_1 =$ a $\text{exp}_1, \cdots, r_k =$ a $\text{exp}_k)$ where qual;
qual: $:= c_1$ and $\cdots$ and c_m
c_i: $:=$ cpred_{i1} or $\cdots$ or cpred_{i1_i}

Here, cpredij is either a join or a restriction predicate. All predicates cpred_{ij} in c_i have to reference the same variable (s). The aexp_i is an arithmetic expression over local attributes.

Fig. 3.1 Normal Form of the Query.

further decomposed into a set of queries each of which has each disjunct as its qualification. It is noted that the qualification is in a conjunctive normal form [Fig. 3.1]. Such a query is called a decomposed GCS query. This process is called a horizontal query decomposition. Each decomposed query can be executed independently. The result of the original query can be obtained by taking the union of results of the decomposed ones.

Then, by looking up the distribution information, the semantic correctness of the target-list and qualification is checked for each decomposed query. Finally, its tree representation is constructed. This process brings in the easiness of logical handling of relational queries. But, this does not mean that it results in the optimal processing of the queries. We think that, in order to adopt the query modification method, it is necessary to normalize queries in the disjunctive normal form. There are still problems such as how to enhance the performance to process "or" like "$x.a = y.a$ or $x.a = z.a$" ($y \neq z$).

3.5 Query Modification

The horizontally decomposed GCS query is translated into queries referencing corresponding LCS relations by means of query modification [STONM76]. That is, first, GCS attributes in the GCS query are replaced by expressions defined over LCS attributes, which are stored in the distribution information. Then, qualifications in definitions of the referenced GCS relations are conjuncted with the query's qualification. The resultant query is called a global LCS query. A GCS definition includes generally more than one subdefinition [see 2]. Hence, a global LCS query is created with respect to each combination of subdefinitions, each of which belongs to the definition of each GCS relation referenced by the GCS query. The modified query is normalized in a conjunctive normal form.

Let us consider the GCS relation PROJ in Fig. 2.2 and a following query: "find names of the projects which exist at JIPDEC." It is also written in QUEL as follows:

```
     range pr PROJ;
DGQ:  retrieve into R (pr. name)
      where pr. location = "JIPDEC";
```

By looking up the distribution information for the GCS relation PROJ, we can find its definition as shown in Fig. 2.2. From the 1st and 2nd sub-definitions marked(1) and (2), respectively, the following global LCS queries are produced:

```
      range (p1, p2, p) (PRJ1: 1, PRJ2: 2, PROJ: 1);
GLQ1:  retrieve into R1 (p. name)
       where p. loc = "JIPDEC" and
```

(1) p. no $= p1$. pno and p. name $= p1$. pname;
GLQ2: retrieve into $R2$ (p. name)
where p. loc = "JIPDEC" and
(2) p. no $= p2$. pno and p. name $= p2$. pname;
The result of the GCS query, $\overline{R}$, is the union of $R1$ and $R2$.

4. Initial Local Query Processing

The global LCS query references the LCS relations at different sites. The next problem is how to process such an inter-site query. The query can be divided into two parts. One references only one site, and the other multiple sites. The former parts have to be processed, first, closely at one site, because it results in a reduction of relations to be transmitted and its cost is assumed to be neglectable. We call such a local processing an initial local query processing.

It is composed of the following functions:

1) to make a query graph[TAKIM80] of the global LCS query which is called a GLQ graph [see Fig. 4.1],

2) to classify nodes in the GLQ graph into groups each of which consists of the nodes at the same site and connected by join-links in the site,

3) to generate LCS queries each of which corresponds to each group, and send them to the corresponding sites.

4) Each site processes these LCS queries which reference relations only in the site, and generates the intermediate relations which are to be processed in cooperation with the other sites [see Ch. 5].

For example, let us consider the GLQ graph in Fig. 4.1. The boxes represent tuple variables. The links between nodes are join-links. The arrowed links are result-links that represent result attributes. The remaining links are restriction links. The dotted circles indicate the groups in which all nodes are locally connected. The conjunction of formulas expressed by these links represents the qualification of the query.

The LCS queries generated in such a manner is written in QUEL. Its target list has to contain attributes for

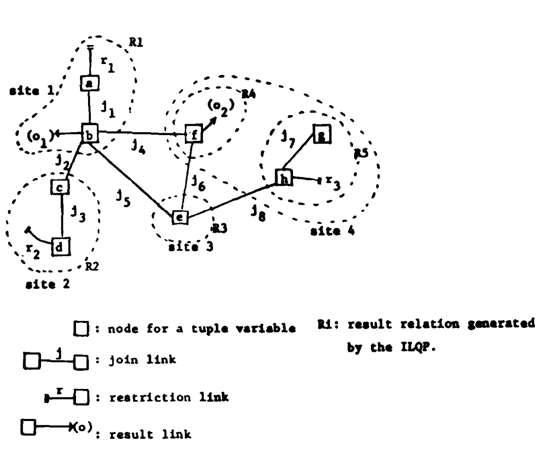

Fig. 4.1 An Example of the GLQG.

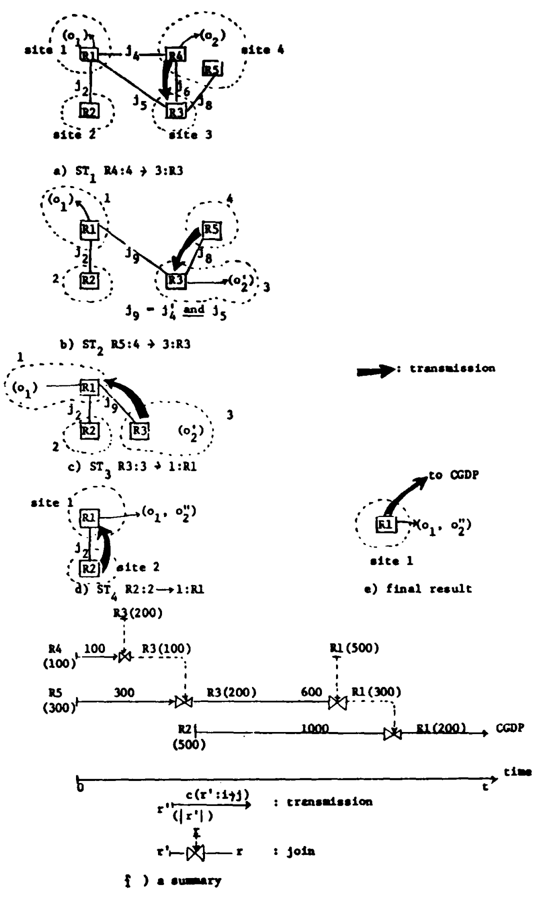

Fig. 4.2 An Example of the TS.

inter-site joining, along with the result attributes specified in the original global LCS query. The result relations of the LCS queries are stored in the working space of the site.

Fig. 4.2a shows the resultant query graph. As seen in this figure, it contains only inter-site joins. Hence, it is called a join query graph.

5. Transmission Scheduling

We consider the generation of a transmission schedule, i.e. an efficient sequence of stages, from the join query graph. Let r' and r be relations at sites i and j, respectively. The stage consists of two subparts: a transmission of r' from i to j, and a join of r' and r and storing of the result as r at j. Hence, let $r': i \rightarrow j$, $c(r': i \rightarrow j)$, and $r': i \rightarrow j: r$ be such a transmission, its cost, and its stage, respectively. Let LDP_k be the local database processor at site k.

Our algorithm for generating the transmission schedule is operational but not static. This means that the CGDP decides consequent stages based on monitored information on results of the preceding stages. To monitor such results, each GDP manages two kinds of directories along with the distribution information, i.e. logical transmission cost table (LCT) and query processing information (QPI). In the QPI, the performance information of intermediate results are stored in two relations: QPI/REL (site-no, rel-no, cardinality, width) and QPI/ATT (site-no, rel-no, att-no, width). Both maintain the performance information on relations and their attributes produced by stages, respectively. Such information are carried back to the CGDP by acknowledgement messages (ACKs) of stages from the destination sites.

Each LCT entry, LC_{ij}, shows the communication cost between sites i and j. At present, LC_{ij} is the number of hops in the shortest path from i to j. Here, $c(r\colon i \rightarrow j)$ is $|r| * LC_{ij}$, where $|r|$ stands for the size of a relation r.

A primitive unit of our algorithm is composed of the following parts: decision of next stage, reduction of the join query graph, and update of the QPI. Initially, all the nodes in the graph are marked FREE.

Suppose that a stage $r'\colon i \rightarrow j\colon r$ is selected as the next one. The CGDP modifies the join query graph. First, it marks r' SOUCE and r DEST in the graph. Then, join-links one, each of which corresponds to a join-link incident on r', are attached to r. If r' has a result-link, it is also attached to r. In relation to such modification of the graph, the QPI/ATT is updated so as to meet the new scheme of r. Let us consider the join query graph in Fig. 4.2a. Suppose that a stage, $R4\colon 4 \rightarrow 3\colon R3$, is selected. Then, the graph is reduced to one if Fig. 4.2b. A join-link, j'_4, corresponding to j_4 is attached to $R3$. Thus, j_9 is a conjunction of j'_4 and j_5. A result-link, o'_2, corresponding to o_2 is also attached to $R3$.

On receipt of an ACK for the stage, $r'\colon i \rightarrow j\colon r$, from j, the CGDP trys to reduce the join query graph. First, it removes r' and its related join-links from the graph. The ACK carries the cardinality of the result relation of the stage. Then, the QPI relations are updated using information in the ACK. That is, all the tuples concerning r' are deleted from these relations and the cardinality of r in the QPI/REL is updated by the new value carried by the ACK.

Next, we decide the next stage. A stage, $r'\colon i \rightarrow j\colon r$, that satisfies the following conditions is selected as the next stage:

1) r' is marked FREE,
2) r is adjacent to r' in the join query graph,
3) r is marked either FREE or DEST, and
4) $c(r\colon i \rightarrow j)$ is not only the minimum in the graph but also less than some threshold value (THV).

The 1st condition ensures that r' is not being executed. The 2nd condition guarantees that there exists a join referencing r' and r. The 3rd one ensures that, even if r is a destination of the other stages that have not completed yet, r' can be sent to r. It means that transmissions of more than one stage can be overlapped in a palallel manner. Since transmission costs are overwhelming, we think these overlappings are effective. The last condition plays a role of protecting relations of larger size from being transmitted when relations of smaller size are currently being executed. If nodes with transmission cost $\leq$THV are not found, the CGDP waits for completion of stages being executed. If all nodes are marked

```
CGDP algorithm
0) [assumptions]
     —let $r'\colon i \rightarrow j\colon r$ be a stage where $r'$ and $r$ are a source relation at
       site $i$ and a destination relation at site $j$.
1) [initial local query processing]
     —an ILQP is considered as a stage $\colon \rightarrow j\colon r$ where $r$ is a result
       relation of it.
     —form the LQs each of which corresponds to each subgroup
       which consists of the nodes not only at the same site but
       also connected by join links.
     —send the LQs to the corresponding sites.
     —create the JQG from the GLQG, whose nodes are the results
       of the ILQP and links are the inter-site joins.
     —mark all the nodes in the JQG "FREE".
     —initiate the QPI which contains all the relation schemes
       corresponding to the nodes in the JQG.
     —initiate the THV (threshold value) using the DI.
2) [wait for ACKs]
     —if the ACK from $r$ is received, then if $r'$ corresponding to $r$
       is NIL, i.e. ACK for the ILQP, then go to 4).
3) [reduction of the JQG]
     —delete $r'$ from the JQG.
     —delete the join-links incident on $r'$ from the JQG.
     —delete tuples concerning $r'$ from the QPI.
4) [update of the QPI]
     —if the ACK is not for the most recent stage to $r$, then go to 2).
     —update the information of $r$ in the QPI using the information
       carried by the ACK.
     —mark $r$ "FREE".
5) [final result]
     —if the reduced JQG contains only one node, then send it to
       the CGDP, i.e. the node is a final result. terminate.
6) [decide a next stage]
     —select the nodes $r'$ as a source node and $r$ as a destination
       note such that they satisfy the following conditions:
         i) $r'$ is marked "FREE",
        ii) $r$ is adjacent to $r'$ in the JQG,
       iii) $r$ is marked either "FREE" or "DEST", and
        iv) $c(r'\colon i \rightarrow j)$ is the minimum and less than the THV value.
            Here, $c(r'\colon i \rightarrow j) = |r'| * LC_{ij}$.
7) [form a stage and send it to the destination site]
     —if such $r$ and $r'$ are found, then
     —form a stage, $r'\colon i \rightarrow j\colon r$.
     —send a transmission command ($T$) to site $i$ and a join
       commands ($J$) to site $j$.
     —mark $r'$ "SOURCE" and $r$ "DEST".
     —set up the join-links between $r$ and the nodes adjacent to $r'$
       except $r$, each of which corresponds to an link between $r'$
       and each node adjacent to $r'$.
     —if $r'$ has the result-link, move it to $r$.
     —update the QPI/ATT so as to meet a new scheme of $r$.
8) [satisfiable $r$ and $r'$ are not found]
     —if all the nodes are marked "FREE",
      then reset the THV using the QPI. go to 6)
      else                              go to 2).
   where
   GLQG: global LCS query
   ILQP: initial local query processing
    JQG: join query graph
     LQ: LCS query
    THV: threshold value
```

Fig. 4.3 TS Algorithm for the CGDP.

FREE and no nodes satisfying this condition can be found, the THV value is reset using the QPI. At present, the THV value is determined to be the average size of nodes.

The detailed description of our algorithm for the CGDP is shown in Fig. 4.3 and for the LDP in Fig. 4.4.

6. An Example of Transmission Scheduling

Let us consider Fig. 4.2. Suppose that all the initial local query processings have finished, i.e. all nodes are marked FREE, and the logical transmission table (LCT) and sizes of relations are given in Figs. 4.5 and 4.6, respectively. Off course, the initial local query processing

```
LDP algorithm
TRANS
1) if the transmission command (T) is received from the CGDP,
   wait for the WSA (WS allocated) from the destination site.
2) if the WSA is received, transmit the source relation along with
   its scheme to the destination site.
3) if the ACK for the transmission is received, then release the
   source relation, r'.
JOIN
1) if the join command (J) is received from the CGDP, then if
   the WS is available, then send WSA to the source site sort r'
   on the join attribute, wait for transmission.
2) if all the source relation is received, send ACK to the source
   site.
   join the source relation to the destination relation with respect
   to thet arget-list and qualification in the join command using a
   merge-join [SELIP79]. form the information on the statistics
   of the result relation of this join.
   send ACK along with this information to the CGDP.
QT
1) if the LQ is received from the CGDP, translate the LQ into a
   DML program by the QT.
   execute the DML program.
   store the result to the WS as a relation.
   send ACK to the CGDP.
REC
1) sort r' on the join attribute while receiving it.
2) if r' is received, send ACK to the source site.
```

Fig. 4.4 TS Algorithm for the LDP.

i	j	LC_{ij}
1	2	2
1	3	2
1	4	5
2	3	2
2	4	2
3	4	1

i, j=site numbers

Fig. 4.5 Logical Transmission Cost Table (LCT).

LCS relations	sizes
$R1$	500
$R2$	500
$R3$	200
$R4$	100
$R5$	300

(in bytes)

Fig. 4.6 The Sizes of Relations.

and transmission scheduling can be overlapped. Each LCT entry, LC_{ij}, represents a minimum hop number between i and j. Let the threshold (THV) value be 500. Let ST_k and ACK_k be the k-th stage and its ACK, respectively. The communication costs with respect to join-links are calculated as follows:

$j2$: $c(R1: 1\rightarrow 2) = c(R2: 2\rightarrow 1) = 500*2 = 1000$

$j4$: $c(R4: 4\rightarrow 1) = 100*5 = 500$ $\because |R4| < |R1|$

$j5$: $c(R3: 3\rightarrow 1) = 200*2 = 400$ $\because |R3| < |R1|$

$*j6$: $c(R4: 4\rightarrow 3) = 100*1 = 100 < 500$ $\because |R4| < |R3|$

$j8$: $c(R3: 3\rightarrow 4) = 200*1 = 200$ $\because |R3| < |R5|$

Since $R4: 4\rightarrow 3: R3$ has the minimal cost, it is selected as ST_1 and the transmission and join commands are sent to 4 and 3, respectively. $R4$ is marked SOURCE and $R3$ DEST. The graph is modified as shown in Fig. 4.2a. As ST_2, $R5: 4\rightarrow 3: R3$ is selected [see Fig. 4.2b], because $R3$ and $R5$ are marked DEST and FREE, respectively, and $c(R5: 4\rightarrow 3) = 300 < 500$ that is also the minimum. Here, $R4$ and $R5$ are transmitted in parallel.

Then, let us try to decide ST_3. Here, only $R1$ and $R2$ are marked FREE. Costs for possible transmissions are as follows:

$j2$: $c(R2: 2\rightarrow 1) = c(R1: 1\rightarrow 2) = 500*2 = 1000 > 500$

$j9$: $c(R1: 1\rightarrow 3) = 500*3 = 1500 > 500$.

Hence, no satisfactory stage can be found. Since $R3$ is not marked FREE, we wait for ACK_1 and ACK_2. On receipt of ACK_1, $R4$ is deleted from the graph and QPI, and ACK_2 is waited for. On receipt of ACK_2, $R5$ is deleted and $R3$ becomes FREE. Suppose the size of $R3$ is 200. Since $c(R3: 3\rightarrow 1) = 600 > 500$, the THV value is reset, i.e. $THV \leftarrow (1500 + 1000 + 500)/3 = 1000$. So, ST_3 is $R3: 3\rightarrow 1: R1$ and executed [see Fig. 4.2c]. $R3$ is marked SOURCE and $R1$ DEST.

Since $R2$ is FREE and $c(R2: 2\rightarrow 1) = 1000$, $R2: 2\rightarrow 1: R1$ is selected as ST_4 [see Fig. 4.2d], and $R2$ is transmitted to $R1$. When both stages complete, the join query graph is reduced to one node graph [see Fig. 4.2e]. Since it is a final result, it is transmitted to the CGDP.

Fig. 4.2e summarizes this example. The horizontal axis shows time.

7. The Architecture of the GDP and LDPs

Fig. 7.1 shows the architecture of the global database processor (GDP) and local database processors (LDPs) for query processing. The User's query is stated to the GDP at his site, i.e. CGDP. The CGDP takes it and translates it into global LCS queries using the distribution information. The ILQP creates LCS queries from the global LCS query, issues them to corresponding LDPs, and creates the join query graph. The TS issues T and J commands for executions of stages generated from the graph and controls their executions, monitoring their intermediate results. The T command, T (k, i, r', j), means a transmission of the k-th stage, $r': i\rightarrow j$. The J command, $J(k, i, r', j, r,$ target-list, qualification, $n)$, means that the source relation r' to be received from i is joined to the destination r at j with respect to the

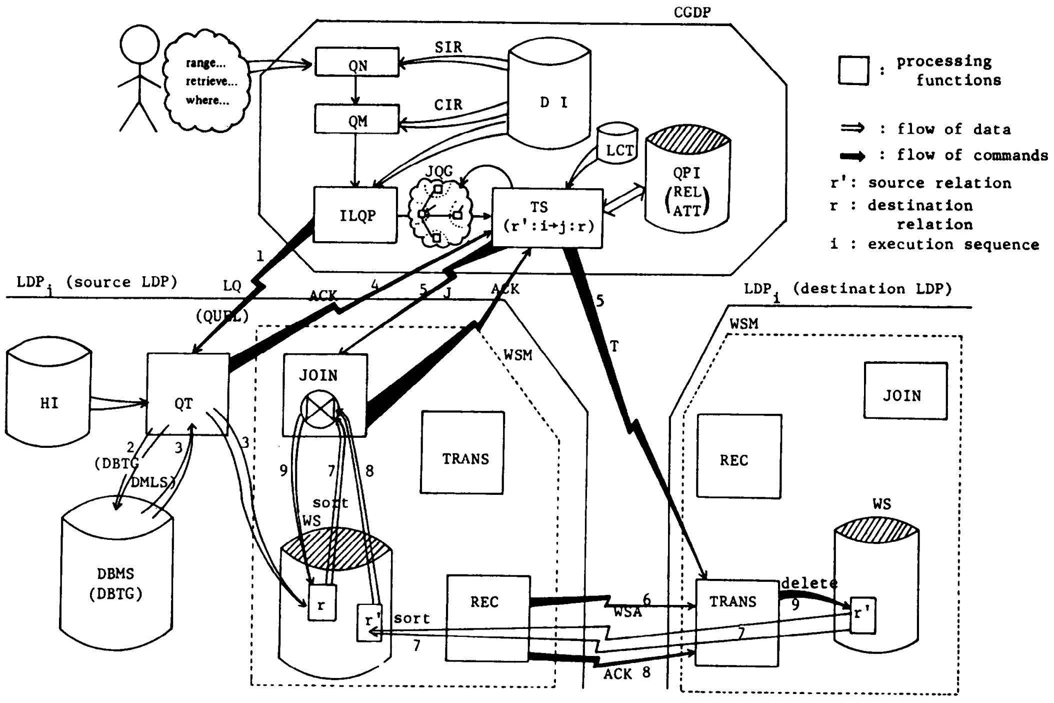

Fig. 7.1 The Architectures of the CGDP and LDPs.

target-list and qualification. The n is a size of r', which is maintained in the QPI. On receiving it, the LDP_j can allocate the working space for receiving r'. The target-list includes join-attributes of r' with respect to its adjacent nodes except r and join-attributes of r with respect to its adjacent nodes except r' along with the union of result-attributes of r and r'.

An LDP exists for one database. The LDP is composed of two main modules. The one is called the query translation [TAKIM80]. It translates the LCS query written in QUEL into an executable sequence, e.g. DBTG DMLs, executes it, and stores the result as a relation in the working space (WS). The other is a WS manager (WSM). It is composed of four submodules, WS, JOIN, TRANS, and REC. The WS is a storage for storing intermediates. It will be implemented as a SAM file. The TRANS takes a T command from the CGDP and transmits the source relation to the destination. The REC of the destination also sorts it on a join-attribute while receiving it. The JOIN takes a J command and sorts the destination relation on a join-attribute. If the source relation is all received, the JOIN joins them, stores the result as the destination relation, and sends ACK with the information on the result to the CGDP. Since both relations are sorted already, they can be easily joined by means of merge-join technique[SELIP79]. Thus, we think it is easy to implement the WSM.

8. Concluding Remarks

In this paper, we have presented mainly the distribution problems in a distributed database system. The distribution problems consist of three subproblems: 1) integration, 2) query decomposition, and 3) distribution information. The integration is a process which derives the GCS relations from the LCS relations. It is also similar to the view definition [STONM76] in the relational model. The main difference between them is that only join operations are used as multi-relation operations in the view definition, but union operations are required in addition to joins in the integration. We have proposed a GCS definition language (GSDL) which is an extension of a relational calculus language QUEL so as to take the union of relations.

The set of GSDL statements is called a distribution definition for the GCS. The correspondence between the GCS and the LCSs is expressed in a relational calculus form. It is also sotred in the distribution information in a relational form. It is desirable for the directory information to exist at the same site as the query decomposition process which requires it. This means that the distribution information is stored fully redundantly at each site. In order that the information is fully redundantly stored and the overhead for controlling the consistency and concurrency of redundant copies is reduced, it has to be as small and static as possible.

Our query decomposition algorithm which is called a

TSA aims at minimizing the distribution information and keeping it static. The decomposition algorithms developed so far have been based on the estimation of the size of the intermediate results. The more information and statistics of relations we have, the more complete our decisions of strategies can be. However, it requires a large amount of information which are also dynamic. That is, there exists a trade-off between complete decision of the strategies and the management of required information. From such an observation, then trying to decide the complete strategy in an off-line manner implies a large amount of dynamic information. We think that it is better to decide the strategy operationally. In this paper, we made clear the meaning of the query graph and the feasible management mechanism of query decomposition based on the dynamic decision.

Every LDP has to have a WS manager which is also a relational database. One of its main tasks is to join two relations. It is very simple and easy to implement, because the source relation is sorted by the REC and the destination one is also sorted before joining.

We have already implemented the QM and ILQP in the GDP. We are now trying to implement the transmission scheduling using our in-house computer network JIPNET [YAMAK 75].

Acknowledgement

We would like to thank Ms. M. Yasuma for her helpful typing of this paper. We would also like to thank Mr. M. Suzuki, Systech Corp., and Ms. K. Yamamoto, Mr. S. Koseki, JIPDEC, for their useful and helpful discussions.

References

[ADIBM78] Adiba, M. and Euzet, C. A Distributed Data Base System Using Logical Relational Machines, Proc. 4th International Conf. on VLOB. Berlin, Sept. (1978) 450–461.

[ADIBM80] Adiba, M. et al. POLYPHEME: An Experience in Distributed Database System Design and Implementation, Proc. of the International Symp. on Distributed Data Bases, Paris, France (Mar. 1980), 67–84.

[BERNP79] Bernstein, P. A., and Goodman, N. EuH Reducers for Relational Queries, Using Multi-Attribute Semi-Joins, Proc. of the IEEE Computer Networking Symposium, Gaithersburg, Maryland (Dec. 1979), 206–215.

[CHU W79] Chu, W. W. and Hurley, P. A Model for Optimal Query Processing for Distributed Data Bases, Proc. of the IEEE Compcon 79 Spring (Feb. 1979), 116–122.

[CHENP76] Chen, P. P. Entity-Relationship Model—Toward a Unified View of Data, ACM TODS, Vol. 1, No. 1 (Mar. 1976), 9–36.

[CODDE70] Codd, E. F. A Relational Model of Data for Large Shared Data Bank, CACM, Vol. 13, No. 6 (June 1970), 337–387.

[EPSTR78] Epstein, R., Stonebraker, M. and Wong, E. Distributed Query Processing in a Relational Data Base System, UCB/ERL M79/18, Electronics Research Lab., UC. Berkeley (April 1978).

[HELDG75] Held, G. D., Stonebraker, M. and Wong, E. INGRES—A Relational Data Base System, AFIPS Conf. Proc. (May 1976), 409–416.

[HEVNA78] Hevner, A. R. and Yao, S. B. Query Processing on a Distributed Database, Proc. of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, San Francisco, CA. (Aug. 1978), 91–107.

[MAHMS79] Mahmond, S. A. et al. Distributed Database Partitioning and Query Processing, Proc. of the IFIP TC-2 Working Conf. on Data Base Architecture, Venice (May 1979), 35–54.

[POTHJ77] Rothine, J. B. and Goodman, N. An Overview of the Design of SDD-1: A System for Distributed Databases, Proc. of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, UC Berkeley California (May 1977), 39–57.

[SELIP79] Selinger, P. G. et al. Access Path Selection in a Relational Database Management System, Proc. of the ACM SIGMOD, Boston, MA. (May 1979), 23–24.

[STONM76] Stonebraker, M., Wong, E., Kreps, P. and Held G. The Design and Implementation of INGRES, ACM TODS, Vol. 1, No. 3 (Sept. 1976), 189–222.

[TAKIM78] Takizawa, M., Hamanaka, E. and Ito, T. Resource Integration and Data Sharing on Heterogeneous Resource Sharing System, Proc. of the ICCC'78, Kyoto, Japan (Sept. 1978), 253–258.

[TAKIM79] Takizawa, M. and Hamanaka, E. The Eour-Schema Structure Concept as the Gross Architecture of Distributed Databases and Heterogeneity Problems, Journal of Information Processing (JIP), Vol. 2, No. 3 (Nov. 1979), 134–142.

[TAKIM80] Takizawa, M. and Hamanaka, E. Query Translation in Distributed Databases, IFIP'80, Tokyo-Melbourne (Oct. 1980), 451–456.

[TSICD78] Tsichritzis, D. and Klug, A. (eds.), The ANSI/X3/SPARC DBMS Framework, Information Systems, Vol. 3, No. 3, 173–191.

[WONGE77] Wong, E. Retrieving Dispersed Data from SDD-1: A System for Distributed Databases, Proc. of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley (May 1977), 273–235.

[YAMAK75] Yamamoto, K. Computer Network System JIPNET, Proc. of the PACNET Symposium, Sendai, Japan (Aug. 1975), 199–206.

(Received August 27, 1980, Revised June 3, 1981)