MODELS OF A VERY LARGE DISTRIBUTED DATABASE

Mark Blakey

Research Laboratories
Telecom Australia
and
Department of Computer Science
Monash University

ABSTRACT

The problems inherent in managing a database distributed over a very large number of sites are considered. The applications of such databases to the provision of telecommunications and other public services are discussed. It is shown that the distribution and maintenance of the directory information describing object locations poses some fundamental problems. A new partially informed class of distributed databases is described which distributes the directory information on a "needs-to-know" basis. The class is described by models of the network topology, and by the knowledge available to each site. These proposals are sufficiently general to support the partitioning of data relations into distribution fragments, and for those fragments to be replicated at multiple sites.

1. INTRODUCTION

Recent developments in computers and communication technologies are enabling a rich variety of advanced information-based and transaction processing services to be offered over public networks. Typical applications include electronic funds transfer, airline reservation systems and real estate listings. Sophisticated telecommunication services, such as on-line directory services, are also emerging. All of these applications rely on an underlying Distributed Database (DDB) technology. Most present day services make relatively modest demands of the DDB given that information is distributed across a small number of sites (typically 20 or less) so that the information processing demands are readily supported by existing commercial DDB systems (see [3,4] for example). It is, however, likely that applications will evolve requiring much higher degrees of distribution. New telephony services such as automatic call redirection could, for example, be established by incorporating a DDB site into each telephone exchange. Manufacturing and operational efficiencies may also be significantly improved by enabling direct communications between an enterprise's databases (e.g. warehouse contents which may themselves be distributed) and those of its clients and suppliers. Applications such as these require DDBs distributed over a very large number of sites (possibly hundreds) so that new techniques must be found to assist in their management.

Some of these applications will naturally require that data objects be partitioned into fragments [5], suitable for storage allocation at specific sites, and that some fragments be replicated at multiple sites. These requirements, together with the size and dynamic nature of the network, pose some fundamental problems in the design of the system's internal data directory that associates fragments with storage sites. These problems are further compounded by the common requirement that the partitioning and distribution of objects be transparent to system users and application programs. Centralizing the directory to a special nominated site is undesirable because (1) failure of that site effectively prohibits any distributed transaction processing, and (2) an additional overhead of a directory interrogation is imposed on each transaction. The alternative arrangement of fully replicating the directory at every site is also undesirable as (1) an unreasonable storage burden is imposed on every site, and (2) an exorbitant amount of network traffic will result whenever data is relocated (i.e. every directory copy would have to be updated).

These problems may be solved by partitioning and distributing the directory information on a "needs-to-know" basis. Each site has a restricted view of the rest

This work is a synopsis of a doctoral project being undertaken by the author at Monash University, Australia. A more detailed account of this material is given in [1,2].

of the network and of the data objects stored by the visible sites. These proposals define a new class of *Partially Informed Distributed Databases (PIDDB)*. In effect, this proposal partitions the directory information so that sites may need to communicate with other sites in the network to discover the identities and location of those fragments that are not described by the local knowledge. Despite the partial knowledge held at each site, these systems will allow any distributed data object to be located from any site.

The following sections present an overview of the PIDDB class. A more complete description, involving non-trivial formalisms and algorithms, may be found in [1,2]. Models are presented in Section 2 that describe the logical network topology, and the knowledge available to any site. An extension of these models that supports *distributed objects* is outlined in Section 3. The objectives and overview of a distributed algorithm for dynamically determining data locations is presented in Section 4. The relationship of this algorithm to PIDDB query processing is also discussed.

2. NETWORK TOPOLOGY

A logical model of the network topology is proposed below that specifies what kind of information sites possess about the rest of the network and how this information is organized. This model anticipates the development of an efficient and distributed algorithm for dynamically determining data locations. The topology defines the *meta-database* known to each site. It is implicit in the definition of the PIDDB class that no site would know the entire meta-database. The portion known to a site constitutes its *local knowledge view (LKV)*. The proposed topology does not require that sites store any data fragments in order to be able to initiate or execute queries (although their meta knowledge is subject to a series of constraints on the LKV).

The topology partitions the network into sets of neighbours called *N-Sets*. All N-Sets contain at least one site and all sites are assigned to at least one N-Set. Sites in a common N-Set know of each others existence and may possess extensive descriptions of the data fragments possessed by each neighbour, although these descriptions may be incomplete or even null (subject to the constraints on the LKV). At least one site in each N-Set would be nominated as an *entry point*. These sites are distinguished by being known (i.e. name and network address) by sites in other N-Sets. All incoming data location or update requests from outside an N-set would be directed into an entry point site. These sites locally distribute the requests within the N-Set as appropriate. Sites assigned to multiple N-Sets define the overlapping *articulation points* between those sets. Articulation points would typically arise where a site makes frequent access to differing N-Sets so that membership of those sets becomes attractive. Groups of N-sets overlapping in this way define a *neighbourhood*. Articulation points would commonly be entry points to improve the efficiency of the data location procedure.

A network would typically be partitioned into a number of disjoint neighbourhoods. These neighbourhoods must however be connected in some fashion so that the location or existence of remote data can be determined. *Hyper-Sets (H-Sets)* of N-Sets are introduced to provide this connection. H-Sets define the *mappings* of fragment names onto the N-sets possessing replications of those fragments. H-Sets may contain any arbitrary collection of N-Sets and neighbourhoods, however neighbourhoods may not span H-Set boundaries. H-Sets may be nested, but other forms of overlap are not permitted. Nesting would typically be useful in large networks where groups of sites were administered by differing agencies; each H-Set would correspond to an autonomous management domain. Such complex networks require a global H-Set (H_g) that encloses the entire network to ensure that disjoint partitions of knowledge do not occur. Since H_g provides connectivity between disjoint sub-networks, its definition must be known within every H-Set.

The N-Set/H-Set model is suitable for modeling the network topologies commonly encountered in practice (e.g. star, ring, hierarchical). In many cases either N- or H-Sets can be used to model a part of the network. The specific design depends on factors such as the anticipated frequency of access between sets of sites, and on the desired degree of autonomy for such sets. As an example, consider the network of nine sites organized into the logical hierarchy of Figure 1. Many PIDDB representations of this network are possible. One topology, in which all sites are grouped into a single global H-Set H_g is shown in Figure 2a. Sites and N-Sets are represented as solid circles and enclosing ellipses respectively. The articulation points at sites 2, 3 and 5 link the entire network into a single neighbourhood. The N-Set boundaries group together those sites expected to frequently access one another.

The constraints on the LKV require that at least one site in each N-Set possesses a description of the immediately enclosing H-Set. This ensures that all sites can locate each of the other sites in that H-Set. Extending the sample topology to very large networks is, however, problematic. Single H-Set topologies reduce N-Set autonomy as data or topological updates must be acknowledged by large numbers of sites. It is also likely that communications between sites in certain pairs of N-Sets would occur infrequently. These considerations provide the motivation for partitioning the network into *multiple* H-Sets. Such an arrangement minimizes the propagation domain of an update and reduces the amount of information that sites must possess about other portions (i.e. H-Sets) of the

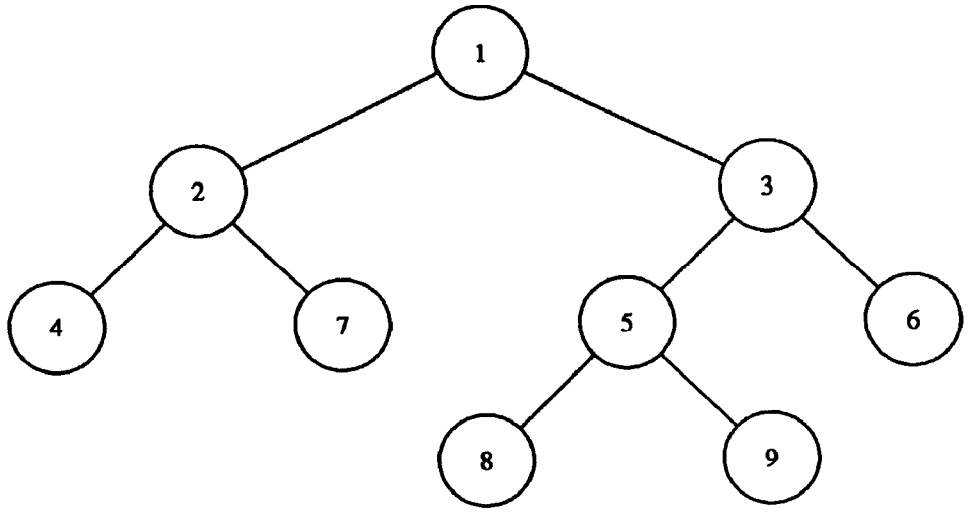
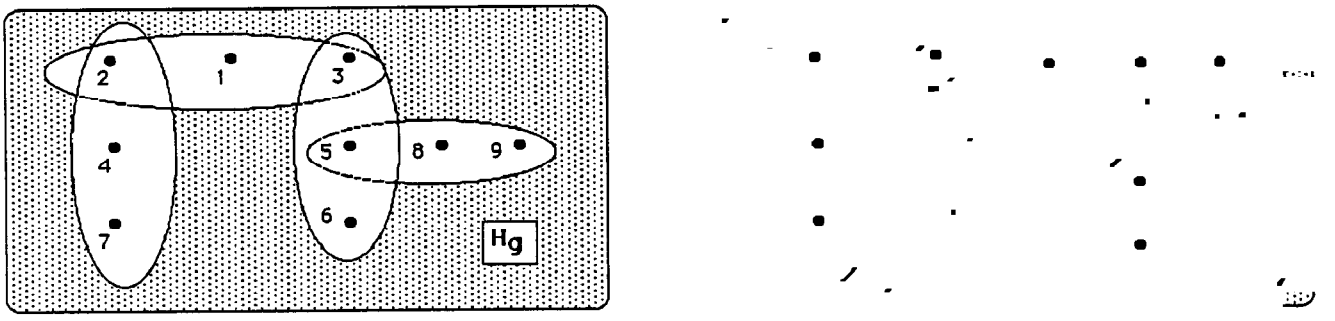


Figure 1: Example of an Hierarchical Network



a) Single H-Set Representation **b) Multiple H-Set Representation**

Figure 2: Representations of the Hierarchical Network of Figure 1

network. As an example, a multiple H-Set representation of the network of Figure 1 is given in Figure 2b. This topology partitions the network into 5 H-Sets, shown here as shaded regions. H-Sets at the same level of nesting (with respect to H_g) are shaded similarly.

2.1 Gateway N-Sets and the Knowledge Kernel

Each H-Set includes a distinguished gateway N-Set N_g through which all communications with other management domains (i.e. remote H-Sets) are directed. Gateways local to the immediately enclosing H-set (H_l) are denoted N_{gl}; gateways to a remote H-set (H_r) are denoted N_{gr}:

$N_{gl} \in H_l$
$N_{gr} \in H_r$

The three principal reasons for introducing gateways are: (1) the volume of traffic between H-Sets may be significantly reduced, (2) secure communications may be established between domains (i.e. authentication protocols can be employed between gateway N-Sets), and (3) the characteristics (e.g. speed, storage, communication channels etc.) of the gateway sites can be chosen to avoid communications bottlenecks.

All sites within N_g possess a *knowledge kernel.* This contains information about:

- the local (immediately enclosing) H-Set H_l,
- the set of remote H-Sets knowing H_l,
- the global H-Set H_g,

2.2 Enclosure Hierarchy Trees

The relationships and relative nestings between H-Sets may be described by an *Enclosure Hierarchy Tree (EHT).* Each node of an EHT represents a specific H-Set and indicates that it contains or *owns* each of the descendant subtrees. N-Sets are <u>not</u> explicitly represented in EHTs. The EHT including H_g spans the

entire network and is called the *Global Enclosure Hierarchy Tree (GEHT)*. The GEHT for the sample topology of Figure 2b is shown in Figure 3.

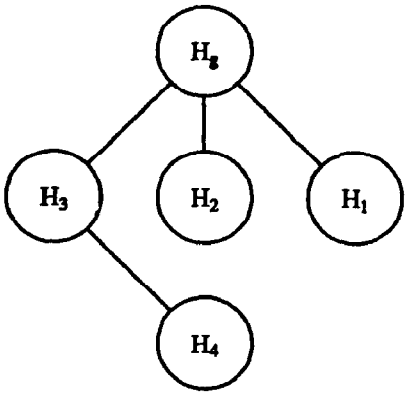


Figure 3: Sample GEHT (Network of Figure 2b)

It is unlikely that any site would know the GEHT. Instead each site possesses a *Local Enclosure Hierarchy Tree (LEHT)* describing only that portion of the topology known locally (i.e. the LEHT defines the LKV). The nodes of an LEHT represent knowledge *about* H-Sets and define *where* this knowledge resides. The relative positions of nodes within an LEHT are consistent with the relative H-Set nestings defined by the GEHT. Individual sites could not otherwise deduce the relationships between the known H-Sets. This information is typically used when determining which remote H-Sets may be able to assist in the refinement of a query. It is also required when changes to the network topology alter the GEHT. Sites informed of the update could not otherwise infer the scope of the change. Eight types of LEHT nodes have been identified representing four different classes of knowledge.

2.3 Network Topology Schemas

The PIDDB topology proposed above is not yet complete enough for the development of operational procedures such as the data location algorithm. An overview of the *conceptual schemas* defining what information is associated with N-Sets and H-Sets, is given below. The schemas are expressed as sets of n-ary tuples of data elements; the ordering of tuples within sets and of elements within tuples is not significant. Underscored elements in the schema definitions indicate that the *name* rather than the *value* of the object is referenced. Some of the parameters included in the schema definitions will not always be available (e.g. status of another site). These parameters are enclosed in square braces ([]) to indicate that their inclusion is *dependent* on their availability.

2.3.1 Notation

Before presenting the detailed schemas, some notation and knowledge operators are required. In addition to the usual structural and value instance information, an object in a PIDDB is not fully described without knowledge of its location. The boolean operator ϕ is useful for expressing or testing this knowledge:

ϕ^n OBJ is *true* if object OBJ is possessed by site S_n.

Another operator, Φ, is a macro that defines the set of sites possessing OBJ.

$$\Phi(\text{OBJ}) = \{S_n: \phi^n \text{OBJ}\}$$

The relational data model [6,7] is assumed throughout this paper: fragment α of relation x is denoted R_x^α. Two further definitions are required:

- $F(R_x)$ The *fragmentation schema* defining the partitioning of relation R_x into the set of fragments $\{R_x^\alpha\}$,
- $A(R_x^\alpha)$ The *fragment allocation schema* defining the assignment of fragment R_x^α to its set of storage sites $\Phi(R_x^\alpha)$. This schema also includes the cardinality $|R_x^\alpha|$.

Queries may or may not be defined with respect to these schemas. Queries that are not refined with respect to either schema are *completely unrefined.* Those defined with respect to the fragmentation schema are *partially refined.* Queries that are also defined with respect to the allocation schema are *fully refined.*

The allocation schema does not constitute a fragment directory in the sense described in §1. Rather, this schema *together* with the topological (i.e. N-Set and H-Set schemas) and knowledge models (i.e. LEHT) constitute the distributed directory.

2.3.2 N-Set Schema

Sites possess extensive knowledge about their *neighbours,* limited knowledge about certain other N-Sets, and no knowledge about any other site. Each site S_n in N-Set N_i would typically be described by its communication parameters C_n, usage costs U_n, current status D_n, the set of fragmentation and allocation schemas held by S_n (F_n and A_n respectively), and boolean indicators P_n and E_n that are *true* if S_n is an articulation point and/or an N-Set entry point respectively. Another boolean N_g associated with each N-Set is *true* if N_i is a gateway N-Set. These parameters constitute the schema:

$$N_i = <\underline{N_i}, N_g, N_s>$$

where $\underline{N_i}$ identifies the N-Set and N_s describes the set of sites participating in N_i:

$$N_s = \{<\underline{S_n}, P_n, E_n, [F_n], [A_n], [C_n], [U_n], [D_n]>\}$$

where $\underline{S_n}$ identifies[1] a specific site. F_n and A_n simplify the data location process within an N-Set as all sites know what data their neighbours possess. It is not therefore necessary to explore the local N-Sets during query processing when these elements are available. The policy adopted for inclusion and maintenance of these elements is local to each N-Set and may vary between N-Sets.

2.3.3 H-Set Schema

The fundamental purpose of the H-Set schema is to express the *mappings* of data fragments onto their containing N-Sets. The H-Set schema therefore contains mapping tuples expressing facts such as fragment R_x^α resides in N-Set N_k:

$$<\underline{R_x^\alpha}, \underline{N_k}>$$

The H-Set schema also includes parameters to ensure that the network is both navigatable and maintainable. For example, sites require knowledge of the identities of at least one ancestor[2] H-Set, and each of the descendant H-Sets. The names of these H-Sets are represented by the sets $\underline{H_a}$ and $\underline{H_d}$ respectively. Two further parameters complete the H-Set schema. C_i is a boolean that is *true* if H_i is the local H-Set H_l. N_h identifies the set of N-Sets participating in H_i and is required to distribute updates of the H-Set schema or extension. The complete H-set schema is expressed by the tuple:

$$H_i = <\underline{H_i}, F_i, C_i, \underline{H_a}, \underline{H_d}, N_h>$$

where $\underline{H_i}$ identifies the H-Set, and F_i is the set of tuples associating fragments with their storage N-Sets in H_i.

3. DISTRIBUTED OBJECTS

Large database systems often serve as repositories of information relevant to a number of applications. It is unlikely that all users of such systems would be interested in all of the available data. The *view* concept [7, 8] simplifies the apparent structure and contents of the database to include only those portions relevant to an application or a users interest. A view is a description of a *virtual object* that identifies the components of the fundamental *base objects* (e.g. relations) included in the view, and defines how these components are combined to instantiate a virtual object.

Distributed systems introduce additional complexity into the view management problem as view components may be both logically and physically distributed across a computer network. An application of the view concept arises in the PIDDB context as a mechanism for modelling *distributed virtual objects*. A distributed object is an instantiation of a virtual object whose components reside at differing locations. This concept permits agencies to cooperate to "provide" *services* to the rest of the network. A telecommunications administration may, for example, wish to define virtual objects combining parts of, say, its telephony and electronic mail directories. Users may then directly interrogate the integrated directory without knowledge of its structure or distribution. The base objects of such integrated services may be owned by *different* administrations. This class of view is commercially significant as it permits information providers to "repackage" their individual information pools into a number of distinct service offerings that would typically be used and tariffed quite differently.

The network is partitioned into two types of domains with respect to each distributed object:

1. *Service Provider Domains (PDs)* which own some component(s) of the object, and
2. *Subscriber Domains (SDs)* which know of or possess an instantiation of the distributed object.

Distributed objects are indistinguishable from other base objects within the SDs and may be represented via LEHTs in the usual way.

The view defining a distributed object may be arbitrarily complex and may contain other views as subcomponents. The structure of these views is defined by a *dependency* or *view graph*. View graphs are both directed and acyclic. The direction of the edges indicates the nature of the dependencies (i.e. arrow heads point towards the more basic objects). Cycles in this graph would represent *recursive* view definitions that could not be instantiated. An example of a view graph for a distributed object is given in Figure 4. R_a, R_b and R_c are base relations which may reside at differing sites. R_{abc} is a view on these relations (e.g. a natural join following projection or selection over each component). R_{abcd} is another view defined in terms of R_{abc} and R_c. Two nodes in this graph are dependent on R_c illustrating why a tree description would be inadequate.

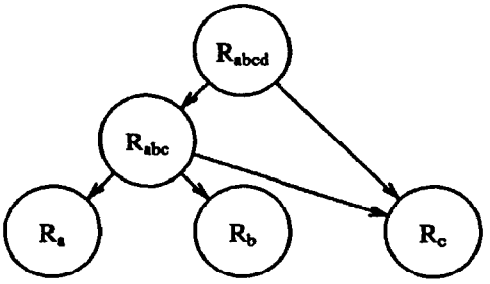


Figure 4: Sample View Graph

1. It is implicit that the site name $\underline{S_n}$ is sufficient to uniquely identify a specific network address.
2. An *ancestor* H-Set encloses H_i. Similarly, a *descendant* H-Set is nested within H_i.

The PDs are responsible for ensuring that each of the SDs receive updates whenever (1) a change in a base object causes a resultant change in a distributed object, or (2) the view definition is altered resulting in changes to the distributed object.

4. PIDDB QUERY PROCESSING

The special nature and requirements of PIDDB query processing are identified below by developing the ramifications of the conceptual framework proposed above. The distribution of responsibility for evaluating and combining portions of the input query, and the requirements of the data location algorithm to support it are presented.

4.1 H-Set Decomposition

The PIDDB framework imposes the constraint that all communications between H-Sets must be via their gateway N-Sets, and that these N-Sets are the only ones known to other H-Sets. It is also implicit that sites can access fragments in remote H-Sets *without* knowing their specific storage locations. It is therefore a natural extension of these principles to *partition* queries involving multiple H-Sets into sub-queries that can be independently evaluated within each H-Set. The gateway sites are then responsible for locating and accessing fragments within their local H-Set on behalf of sites in other H-Sets. Query evaluation therefore requires a *master schedule* concerned with finding an optimal strategy for transporting and combining the partial results produced by each participating H-Set. The initial partitioning of the query is called *H-Set decomposition.* One ramification of this decomposition is that the data location algorithm need not refine the locations of remotely owned relations beyond identifying the relevant gateway N-Sets N_{gr}. This also suggests that data location is an independent phase of query processing that terminates within each H-Set before evaluation of the local sub-query commences. While this Section is predominantly concerned with the data location algorithm, it is necessary to consider its relationship to the larger query processing problem. An operational basis for managing the generation, evaluation and combination of the partial queries is proposed below. The initiating site S_{init} coordinates the overall activity in this scenario and is responsible for:

1. performing the initial H-Set query decomposition,
2. distributing the partial queries to the gateway N-Sets of the identified H-Sets,
3. determining the locations of each of the data objects required for the local sub-query executable within H_l,
4. evaluating the local sub-query,
5. planning and executing a master schedule for distributing and combining the partial results produced within each H-Set. (This step could be delegated to N_{gl} to minimize interactions within H_l.)

An example of H-Set decomposition in PIDDB query processing is demonstrated in Figure 5. Messages associated with data location and master schedule

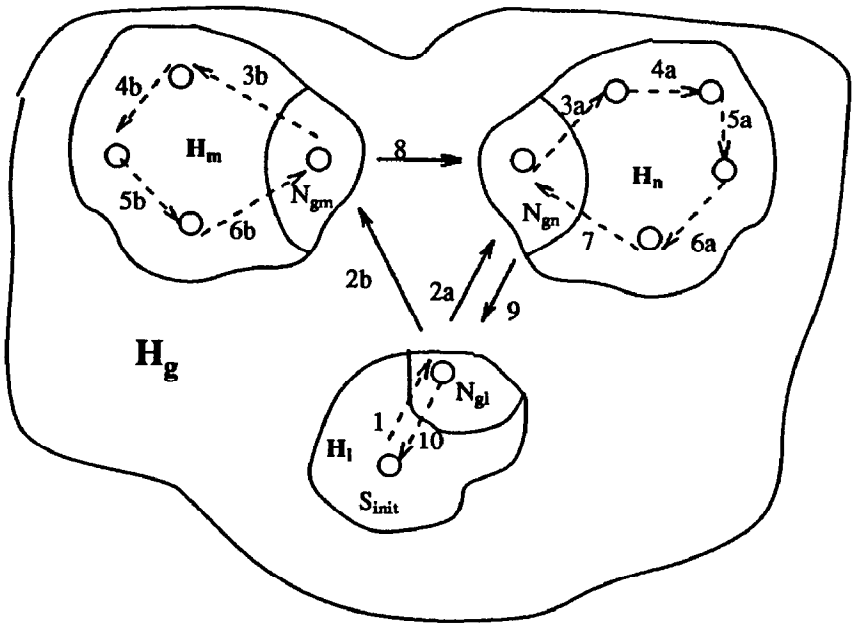


Figure 5: H-Set Query Decomposition

coordination are not shown. Only gateway N-Sets are identified; other N-Set boundaries within each H-Set are ignored for simplicity. Dashed arcs represent the optimal query evaluation sequences within each H-Set. Solid arcs represent the master schedule and illustrate the distribution and combination of the partial queries and results respectively. The numbers on the arcs indicate the chronological order of the messages. Messages distinguished by lower case letters are issued simultaneously.

The partial queries would typically be distributed simultaneously (messages 2 in the Figure). Each H-Set assisting H_l invokes the data location algorithm within its local gateway N-Set to plan and then execute an optimal schedule for evaluating the local sub-query. This is represented by messages 3a to 7 and 3b to 6b in the example. The partial results are then combined according to the master schedule planned within H_l. This would typically require exchanging additional messages with H_l to report the cardinalities of the partial results. (The master sequence cannot be completely planned until this information becomes available. The coordination messages reporting cardinalities and returning instructions are omitted from the Figure for clarity.) In the example scenario, it is assumed that the partial result produced in H_m is smaller than that produced within H_n, and that the final result is significantly smaller than either partial result. The optimal strategy is therefore to send the result produced in H_m to H_n (message 8), and then to return the final result to H_l (message 9). Message 10 relays the final result to S_{init}.

4.2 Objectives of Data Location

The PIDDB query processing paradigm proposed above provides a basis for defining the objectives of the data location algorithm:

1. to identify the set of relevant fragments possessed within H_l (i.e. fragments that are implicitly referenced by the user's global query),
2. to determine the specific storage sites and cardinalities of each relevant fragment in H_l,
3. to determine which remote H-Sets contain sites owning relevant relations[3] that are not available within H_l.

Many heuristic techniques for finding an acceptably efficient[4] query evaluation strategy are described in the literature and are suitable for evaluating the partial queries within each H-Set. Early examples, based on semi-join techniques [5,9], include the SDD-1 query processor [10] and the techniques developed by Hevner et. al. [11,12]. More recent developments were described by Chiu and Ho [13], Chu and Hurley [14] and by Lafortune and Wong [15]. The fragment cardinalities obtained during data location are required by these techniques to plan an efficient schedule for transferring partial results between sites within an H-Set. This schedule cannot be planned until all of the required locations and cardinalities have been determined. Data location is therefore an independent and preliminary phase of query processing that completes within each H-Set before evaluation of the local sub-query commences.

4.3 Phases of Data Location

The global query, denoted gq, explicitly references a set of relations. Two refined representations of gq, namely fq and aq, are proposed. The canonical fragmentation query fq is derived by analysing the relevant fragmentation schemas to replace each relation in gq by an appropriate set of data fragments. The canonical allocation query aq is derived by augmenting fq with additional information that defines the cardinality and location of each relevant fragment. The data location problem can be phrased in this context as the task of refining gq to fq and then refining fq to aq.

The fragmentation schemas required to produce fq may not all be locally available at S_{init}. It is therefore necessary to locate the required fragmentation schemas before the locations of specific fragments can be determined. There are therefore two logically distinct and serial phases of data location:

- a preliminary *knowledge acquisition phase* (*K*-Phase) during which the locations of the required fragmentation schemas are determined, and
- a *sites identification phase* (*S*-Phase) during which the identities, locations and cardinalities of the relevant fragments are determined.

The *K*-Phase determines where the required fragmentation schemas are possessed. It is implicitly assumed by invoking the *S*-Phase at these sites that they can identify and locate the required fragments. The proposed data location algorithm relies on the observation that sites capable of identifying any fragments implied by gq must also be capable of determining the locations and cardinalities of those fragments. Theorem 1 states this observation more formally. The knowledge and topological models proposed in [1] are used to prove this theorem in [2].

3. The set of relevant *fragments* implied by these *relations* are determined by the instances of the data location algorithm in the remote H-Sets.
4. It is typically necessary to employ *acceptably efficient* schedules as discovery of an *optimal* solution is often intractable.

Theorem 1: Refinement Capabilities

Suppose that, as a result of the *K*-Phase, S_{init} determines that S_n possesses each of the fragmentation schemas necessary to refine a subset gq' of gq to fq'. Then if S_n is capable of refining gq' to fq' it must also be capable of refining fq' to aq'.

4.4 Query Taxonomy

A number of distinct query versions are appropriate as data location proceeds. A total of 16 query versions in four classes have been identified. This taxonomy provides a precise basis for the development of the *K*-Phase and *S*-Phase data location algorithms. The *input* class contains a single member that is the user's input query. The next two classes cater for the requirements of the data location algorithm: the *knowledge* and *sites* queries cater for the *K*-Phase and *S*-Phase respectively. The final *evaluation* class is concerned with H-Set decomposition for query evaluation. An overview of the various query versions is given in Table 1. All of these queries are generated at S_{init}, except fq_j^s and aq_j^s which are generated at the sites refining gq_j^s during the *S*-Phase of data location.

The relationships between these queries are illustrated as a directed graph in Figure 6 where each node represents a specific query. The edges identify the specific dependencies: the query at the head of an edge is derived from and generated after the query at the tail. gq_{init}^s is, for example, a subset of gq^{sl} which in turn is generated using the information contained in gq and gq^k.

The fundamental objective of data location can be restated in terms of this taxonomy as the task of transforming the input query gq to the local allocation schema sub-query aq^l and the remote global schema sub-queries gq_y^r that are sufficiently detailed to enable planning an optimal execution strategy.

4.5 Overview of Data Location

Theorem 1 provides an operational basis for the data location algorithm. A series of 14 conceptual

TABLE 1: PIDDB Query Taxonomy

Class	Query	Content
Input	gq	The user's input query defined on the global schema.
Knowledge	gq^k	*K*-Phase input version of gq identifying each relevant relation
	gq^{sl}	*K*-Phase output/*S*-Phase input identifying relations with N-Sets in the local H-Set H_l.
	gq^{sr}	*K*-Phase output identifying relations with remote H-Sets H_r.
Sites	gq_{init}^s	The portion of gq^{sl} refinable by S_{init} (i.e. each of the relevant fragmentation schemas implied by gq_{init}^s has been determined by the *K*-Phase to be possessed by S_{init}).
	gq_j^s	The portion of gq^{sl} refinable by N-Set $N_j \in H_l$ (i.e. each of the relevant fragmentation schemas implied by gq_j^s has been determined by the *K*-Phase to be possessed within N_j).
	fq_{init}^s	The partially refined version of gq_{init}^s.
	fq_j^s	The partially refined version of gq_j^s.
	aq_{init}^s	The fully refined version of fq_{init}^s.
	aq_j^s	The fully refined version of fq_j^s.
	aq^s	The aggregate of the aq_j^s and aq_{init}^s generated within H_l.
Evaluation	gq^l	The portion of gq such that each of the fragments implied by the relations referenced in gq^l is possessed by some site in H_l.
	gq^r	The portion of gq such that each of the fragments implied by the relations referenced in gq^r is possessed by a site within some remote H-Set H_r.
	gq_y^r	The portion of gq^r refinable within remote H-Set H_y.
	fq^l	Partially refined version of gq^l.
	aq^l	Fully refined version of fq^l.

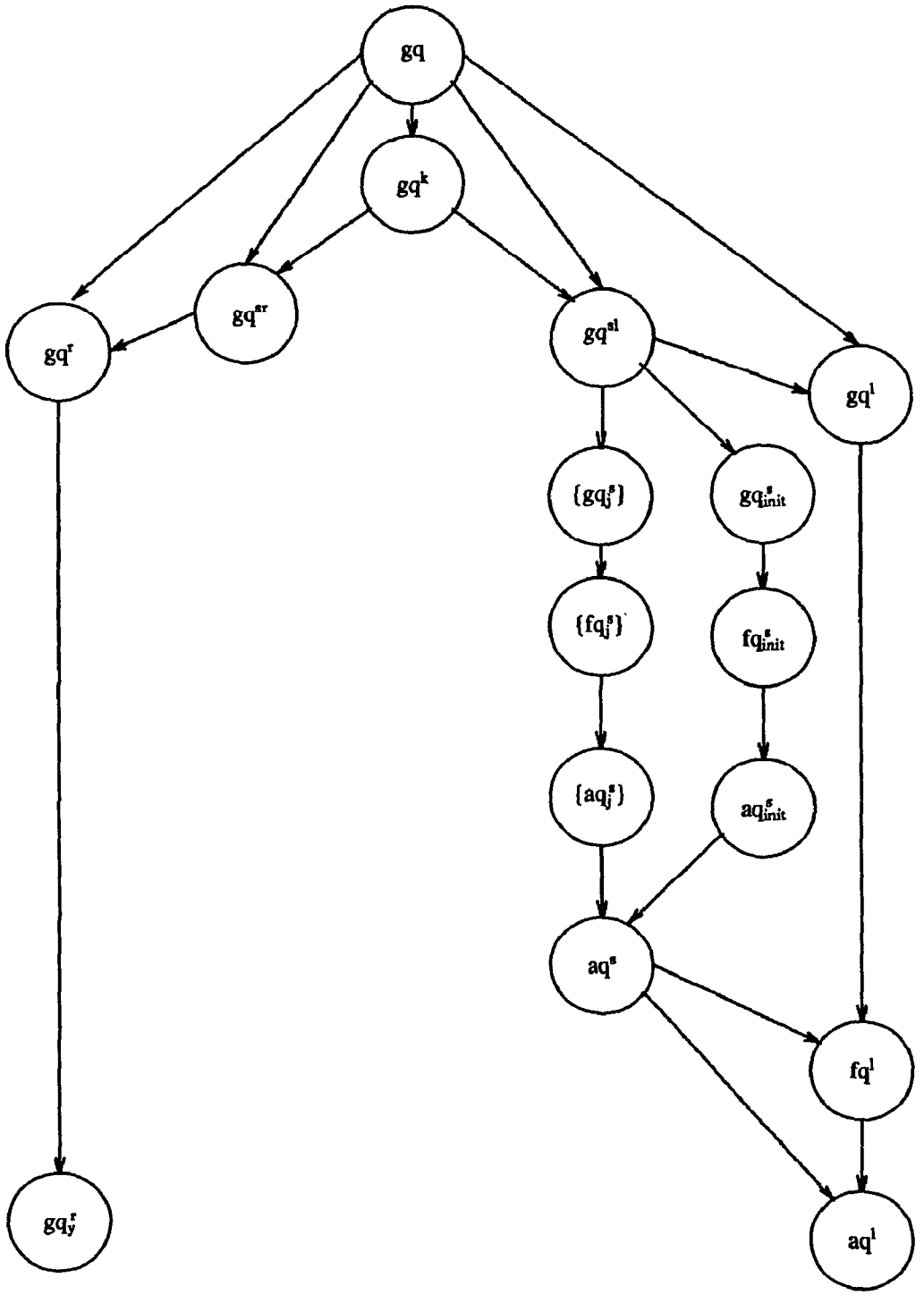


Figure 6: Dependencies Between PIDDB Queries

algorithms that collectively constitute the data location algorithm are presented in [2]. A simplified overview (incorporating both the *K*-Phase and *S*-Phase) is given below.

4.5.1 Simplified Algorithm

The algorithm demonstrates the application of part of the proposed query taxonomy and is concerned with identifying the local sub-query gq^{sl} and with locating the

fragments implied by it:

1. for each R_x named in gq^k such that $\neg \phi^{init} F(R_x)$, S_{init} uses its local knowledge view to determine a set of candidate N-Sets possessing the relevant fragmentation schemas:

   $$\{N_j \in H_l: \exists S_n \in N_j: \phi^n F(R_x)\} \quad (4.5a)$$

   Hence refine gq^k to gq^{sl}.

2. decompose gq^{sl} into a set of partial queries such that each partial query can be fully refined within one of the identified N_j:

   $$\{gq_j^s \subseteq gq^{sl}: \cup_j gq_j^s = gq^{sl}\} \quad (4.5b)$$

   on the basis that for every R_x named in gq_j^s

   $$\exists S_n \in N_j: \phi^n F(R_x) \quad (4.5c)$$

3. distribute the partial queries gq_j^s to their respective N-Sets.

4. await the partial responses aq_j^s and assemble the final result aq^s.

Step 1 identifies that portion of the global query refinable within H_l. Step 2 defines an *N-Set query decomposition* that identifies the portion of gq^{sl} refinable by each N_j. The criteria for N-Set selection in given in expression 4.5a. The constraint defined in expression 4.5b ensures that each relation in gq^{sl} is represented in some partial query; the constraint of expression 4.5c reflects the N-Set selection criteria of expression 4.5a.

This algorithm assumes that S_{init} has a sufficient local knowledge view to complete the N-Set decomposition of step 2. This is not necessarily true in general: it is the task of the preliminary *K*-Phase identified above to collect sufficient knowledge before attempting N-Set decomposition.

Sites in the selected N-Sets $\{N_j\}$ fully refine the decomposed query gq_j^s by applying Theorem 1. An overview of their algorithm is (note that "local" and H_l are with respect to the site in N_j, not the originating site):

1. use the locally held fragmentation schemas to refine gq_j^s to fq_j^s.

2. refine fq_j^s to aq_j^s by determining the locations and cardinalities of each R_x^α named in fq_j^s such that,

   a. if $\phi^l A(R_x^\alpha)$ then, by definition, $\Phi(R_x^\alpha)$ and $|R_x^\alpha|$ are available; fq_j^s may be refined to aq_j^s with respect to R_x^α.

   b. if $\neg \phi^l A(R_x^\alpha)$ then request $A(R_x^\alpha)$ from another site in H_l (using a description of H_l to identify the assisting site); fq_j^s may now be refined to aq_j^s with respect to R_x^α.

3. return the partial result aq_j^s to S_{init}.

An example of the manner in which the complete data location algorithm would explore the PIDDB topology is given in Figure 7. N_i and N_j represent two typical N-Sets local to S_{init}. If data location cannot be completed within these N-Sets than the neighbourhoods within H_l are searched. NH_l and NH_a represent the local and some other neighbourhood respectively within H_l. If these are also found to be inadequate then a wider domain search involving other H-Sets is invoked. H_m

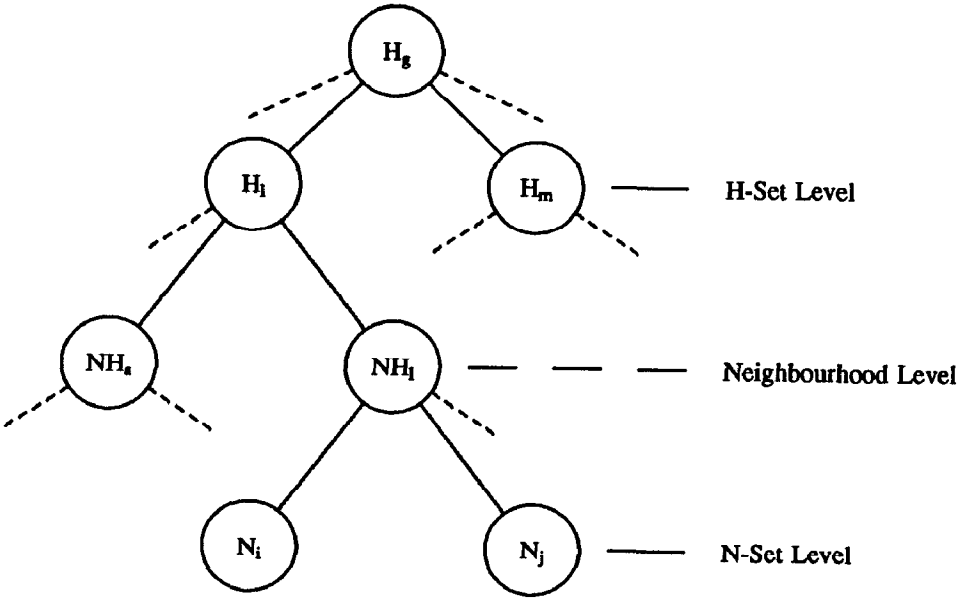


Figure 7: Overview - Levels of Data Location Search

represents another typical H-Set that may be investigated. Finally, if the available or acquired knowledge proves inadequate then the global H-Set H_g enables a systematic and exhaustive search of the entire topology to be undertaken.

4.6 Overview of *S*-Phase Algorithms

A simplified overview of the *S*-Phase algorithms is presented below. This overview develops steps 2 to 4 of the simplified data location algorithm for S_{init}:

1. initially decompose gq^{sl} into two components:
   a. $gq^s_{init} \subseteq gq^{sl}$ such that for every R_x named in gq^s_{init}, $\phi^{init} F(R_x)$
   b. gq' such that $gq' = gq^{sl} - gq^s_{init}$
2. locally refine gq^s_{init} to aq^s_{init}
3. perform N-Set decomposition on gq' to produce $\{gq^s_j\}$ and distribute sub-queries to the corresponding N_j
4. await aq^s_j responses, combine with aq^s_{init} and assemble aq^s. Delete the corresponding gq^s_j from $\{gq^s_j\}$ as each response is received
5. terminate when $\{gq^s_j\} = \varnothing$

Step 1a identifies those portions of gq^{sl} refinable by S_{init}. Step 1b finds the residue refinable by other N_j. This residue is (N-Set) decomposed in step 3 into the portions relevant to specific N-Sets. While many partitions may be possible, it is implicit that the *optimal* partitioning would be found. This minimizes the number of N-Sets involved by selecting those containing several relevant fragments in preference to those containing only a single fragment.

5. CONCLUSION

A model for managing very large distributed databases has been presented. This model involves restricting the meta knowledge available to any site and defines a new class of Partially Informed Distributed Databases (PIDDB). This class is fully transparent with respect to data fragmentation, location and replication. It is inherently suitable for very large systems and preserves the autonomy of sub-domains. No assumptions are made regarding the initial or current allocation of data fragments to sites so that replications may readily be created or deleted within their owning domains.

A mechanism has been described that permits information providers to cooperate to define *distributed virtual objects* using components of their individually owned objects. These objects are instantiated and distributed as snapshots.

The relationship of data location to PIDDB query processing has been developed and a mechanism for coordinating query evaluation has been proposed. It was shown that differing versions of the initial global query exist as data location proceeds. A descriptive query taxonomy was introduced to provide a concrete basis for the development of the data location algorithm. A total of 16 different query versions have been identified.

Data location has been found to consist of two distinct and serial phases: a knowledge acquisition phase (*K*-Phase) and a sites determination phase (*S*-Phase). The *K*-Phase is concerned with determining which domains (local N-Sets or remote H-Sets) contain sites possessing the fragmentation schemas relevant to the global query. The *S*-Phase is concerned with applying the knowledge acquired during the *K*-Phase to identifying, locating and determining the cardinalities of the relevant fragments. Conceptual algorithms have been presented that define the basis of the procedures required to develop a prototype system.

6. ACKNOWLEDGEMENT

The support of Telecom Australia to undertake this work is appreciated. The permission of the Director Research, Telecom Australia to publish this paper is acknowledged. Thanks are also due to Dr Ken J. McDonell, Department of Computer Science, Monash University, for his numerous constructive suggestions during the preparation of this paper.

7. REFERENCES

1. M. Blakey, Partially Informed Distributed Databases: Conceptual Framework And Knowledge Model, Tech. Rep. 80, Dept. of Comp. Science, Monash Univ., Melbourne, Australia, Dec., 1986.
2. M. Blakey, Partially Informed Distributed Databases: Data Location Algorithm, Tech. Rep. 87/85, Dept. of Comp. Science, Monash Univ., Melbourne, Australia, May, 1987.
3. C. Mohan, B. Lindsay and R. Obermarck, Transaction Management in the R* Distributed Data Base Management System, Report RJ 5037, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, Feb., 1986.
4. J. B. Rothnie, Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman and E. Wong, Introduction to a System for Distributed Databases (SDD-1), *ACM Trans. on Database Sys. 5* , 1, (1980), 1-17.
5. S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, 1985.
6. E. F. Codd, A Relational Model of Data for Large Shared Data Banks, *Comm. ACM 13* , 6, (1970), 377-387.

7. C. J. Date, *An Introduction to Database Systems, Volume 1, 4th Edition*, Addison-Wesley, Reading, Massachusetts, 1986.

8. J. D. Ullman, *Principles of Database Systems*, Comp. Science Press, Rockville, Maryland, 1982.

9. P. A. Bernstein and D. M. Chiu, Using Semi-joins to Solve Relational Queries, *J. ACM 28* , 1, (Jan. 1981), 25-40.

10. P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve and J. B. J. Rothnie, Query Processing in a System for Distributed Databases (SDD-1), *ACM Trans. on Database Sys. 6* , 4, (1981), 602-625.

11. A. Hevner and S. B. Yao, Query Processing in Distributed Database Systems, *IEEE Trans. on Software Eng. SE-5* , 3, (May 1979), 177-187.

12. P. M. G. Apers, A. R. Hevner and S. B. Yao, Optimization Algorithm for Distributed Queries, *IEEE Trans. on Software Eng. SE-9* , 6, (Jan. 1983), 57-68, IEEE.

13. D. M. Chiu and Y. C. Ho, A Methodology for Interpreting Tree Queries Into Optimal Semi-join Expressions, *Proc. ACM SIGMOD International Conf. on Management of Data*, Ann Arbor, Michigan, ACM, New York, Apr., 1980, 169-178.

14. W. Chu and P. Hurley, Optimal Query Processing for Distributed Database Systems, *IEEE Trans. on Computers C-31* , 9, (Sep. 1982), 835-850.

15. S. Lafortune and E. Wong, A State Transition Model for Distributed Query Processing, *ACM Trans. on Database Sys. 11* , 3, (Sep. 1986), 294-322.