BASIS OF A PARTIALLY INFORMED DISTRIBUTED DATABASE

Mark Blakey

Research Laboratories
Telecom Australia
and
Department of Computer Science
Monash University

ABSTRACT

This paper proposes a new class of replicated distributed databases offering high levels of distribution transparency for very large networks of processing sites. The class is distinguished by the restriction that processing sites possess limited rather than complete knowledge of the data objects and sites in the system. The traditional data directory is replaced by a more sophisticated knowledge model. An axiomatic framework identifying the fundamental properties of the class is presented. A topological network model is developed from this framework as the basis of the knowledge model. The practical importance of supporting autonomous sub-domains is recognised and accommodated in the topological model. A propositional calculus is presented to simplify reasoning about the physical location of data. A series of heuristics that minimize the search effort required to discover an object's location are presented. The merits of the proposed model and search heuristics are demonstrated by developing an outline of the main operational procedure peculiar to the proposed class: the data location algorithm.

1. INTRODUCTION

Most Distributed Databases (DDBs) to date have employed small numbers of processing sites. It is, however, likely that, with the increasing maturity of the supportive communications and computing technologies, new DDB applications will emerge requiring larger numbers (e.g. hundreds) of cooperating sites. Examples might include new telecommunications services such as on-line electronic directory services or public access databases. New value-added telephony services such as automatic call redirection could also be established by incorporating a DDB site within each telephone exchange. Other industries also likely to develop large DDB applications include financial institutions (e.g. account records for electronic funds transfer), travel (e.g. airline reservations) and real estate (e.g. property listings).

The data directory[1] is critical to the success of these larger systems. There are however a number of problems inherent in determining which sites should possess versions of the directory, and how complete these versions should be. Suppose, for example, that the entire directory resided at a special site known by all other sites. This arrangement suffers from 2 major disadvantages: (1) all interactions with the database incur the cost and delay of a remote access to find the required data, and (2) the reliability of the system may be compromised as no interactions could be processed if the storage site or its communication links failed. Another possible arrangement would be to replicate the directory at each site. This may however impose an unreasonable storage burden on smaller capacity sites, and result in significant network congestion as all directory copies would require updating whenever data was relocated or a site joined or left the system. There is a risk of network congestion in this arrangement and it is likely that the response delays would often prove unacceptable. Access to the complete directory could also result in the contradictory situation that a user knows the location of an object whose existence is supposed to be unknown to that user (i.e. knowledge of an objects existence may be subject to an access control).

One solution to these operational problems is proposed and explored in this paper: the directory knowledge available to any site is restricted to include only its own objects, and those possessed by a well-chosen subset of other sites. No other information about the rest of the network, or even of its existence, is directly available to any site. Such a system, introduced here, is called a *Partially Informed Distributed Database (PIDDB)*. Partitioning the directory in this way enables it to be distributed while avoiding the problems discussed above. Sites may possess differing subsets of the global directory and some components of the directory may be replicated at several sites. The PIDDB class provides the conceptual framework for a wide range of information retrieval and transaction processing applications. The principal purpose of this paper is to establish this framework and to demonstrate its potential for developing efficient implementations.

Distributed query processing typically involves transferring data objects between sites for partial execution at different sites. Once the

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].

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

1. DDBs typically maintain directories associating data objects with their storage sites.

location of all of the required objects is known, an acceptably efficient processing schedule can be generated. A preliminary phase of PIDDB query processing therefore involves discovering the locations of any required data objects that are not described by the locally available directory knowledge. Numerous techniques have been developed for efficient query processing both within a single site and between sites when data locations are known [3-9]. Distributed schedule generation algorithms typically employ heuristic hill climbing techniques based on semi-join programs [9]. Examples include the query processor used in SDD-1 [10] and the AHY algorithms [11, 12].

The emphasis of this paper is placed on data location techniques since it is this problem that characterizes the PIDDB class. The main objectives are:

1. to formalize a number of assertions and observations as a conceptual framework of general applicability, and
2. to develop the basis of some specific data location techniques within this framework.

Further justification of the PIDDB notion is given in section 2. The conceptual framework of the PIDDB class is presented in Section 3. The notion of *refining* a query to determine the set of relevant object locations is introduced. A model of the required meta database is developed in Section 4. This includes a logical network topology and a descriptive knowledge framework. An outline of a distributed data location algorithm is given in Section 5. This algorithm employs an heuristic *search hierarchy* and demonstrates the features of the proposed knowledge model. A *knowledge calculus* is proposed in an Appendix to simplify the expression of and reasoning about the location of objects within the database.

2. JUSTIFICATION OF THE PIDDB NOTION

The notion of a partially informed system is justified by demonstrating that a completely uninformed system is infeasible (the problems associated with a large, fully informed system are discussed above). Such a system may locate data by broadcasting requests for directory knowledge; the complexity of this search procedure is O(N), for a network of N sites. While theoretically acceptable, it is likely to result in a great deal of unproductive processing and network congestion may result when N is large. The problem is compounded as the number of concurrent transactions T and objects O referenced by each transaction increase. As T is likely to increase with N it is likely that the response time will become unacceptably long for certain combinations of N, T and O. The introduction of a knowledge model, in which sites *know* certain remote objects and sites, has the potential to significantly reduce the cost and delay of the data location process; the resultant system is *partially informed* as asserted. Use of a knowledge model implies that $n < N$ sites will be visited (hopefully $n << N$). The cost and delay of the data location problem will be minimized when an optimal search path for visiting the n sites is found. Unfortunately, the number of possible search paths grows exponentially[2] with N. Thus to achieve a linear reduction (scaling) in the delay and cost of the data location process, an intractable optimization problem must be solved. This paper proposes the basis of a suitable knowledge model to realize the benefits of constraining the data location procedure. Firmer notions of which sites and objects should be known, how this information should be expressed and the special properties of the PIDDB class are developed below.

2. This optimization problem is shown to be NP-complete in [1].

3. CONCEPTUAL FRAMEWORK

The PIDDB conceptual framework is developed below by adopting a reference architecture, identifying an axiomatic framework, and by proposing the notion of refining a query.

The relational data model [13], [3] is assumed throughout this paper: relation x (or any lower case letter) is represented R_x. The assumption that all user level data objects are relations does not imply any loss of generality.

3.1 Reference Architecture

A distributed database reference architecture[3] is developed by augmenting the ANSI/SPARC model [14, 15] of a centralized (single site) database system with additional schemas that describe the distribution of information. Four new schemas are introduced above the centralized conceptual schema as shown in Figure 1. The global, fragmentation and allocation schemas are site independent in the sense that they are not concerned with the details of the local data models, storage structures or access strategies. The local mapping schema is however site dependent. This schema defines the translation between the global and local data models or Data Base Management Systems (DBMS). Data objects may, for example, be organized locally into CODASYL [16] sets and globally into relations. Such a system is *heterogeneous*, unlike *homogeneous* systems in which identical DBMSs (and hence data models) are used at all sites.

The *global schema (GS)* provides a time-invariant[4] abstract description of the global relations R_x. Its role is similar to that of the ANSI/SPARC conceptual schema in that it is concerned purely with the content and semantics of the conceptual records. The logical or physical distribution of these records is transparent in the global schema.

The *fragmentation schema (FS)* is concerned with partitioning the global relations into logical fragments. This partitioning is typically useful where subsets of a relation possess common properties so that it is convenient to allocate them as atomic entities. The FS of a specific relation is denoted $F(R_x)$. Fragment α (an integer) of relation R_x is represented R_x^α. (e.g. R_j^3).

The *relation allocation schema (RAS)* is concerned with the physical allocation of the logical fragments to the processing sites. It is often convenient to consider the allocation schemas of particular fragments rather than of complete relations. It is generally true that assertions and restrictions effecting the *fragment allocation schema (FAS)* also hold for the relation allocation schema. The unqualified term "allocation schema (AS)" shall therefore be taken to refer to the FAS below. The RAS and FAS, denoted $A(R_x)$ and $A(R_x^\alpha)$ respectively, are related by the constraint:

$$A(R_x) = \bigcup_\alpha A(R_x^\alpha)$$

This implies that all components of R_x are mapped onto some fragment R_x^α. It is desirable that this mapping minimize the overlap between fragments. Otherwise it would be difficult to model the allocation of data objects if they were partially replicated between logical fragments.

3. The architecture used in this paper is substantially that described by Ceri and Pelagatti [9] with minor extensions to the allocation schema.

4. Practical and commercial considerations may dictate that changes to these schemas occur from time to time. It is implicit that such changes should occur infrequently and *offline*.

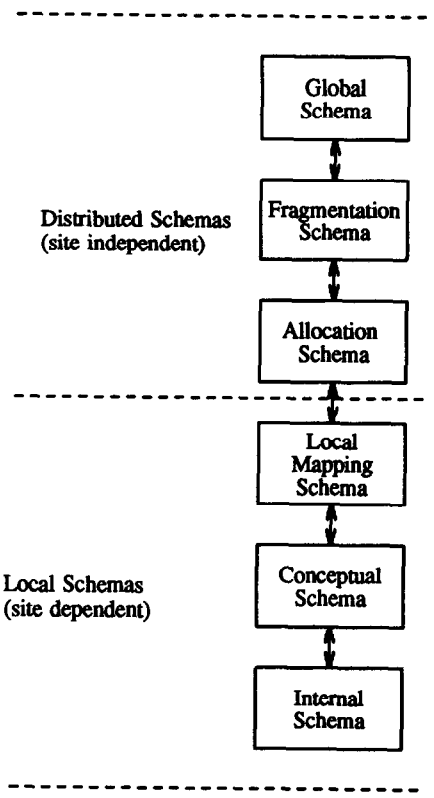


Figure 1: DDB Reference Architecture

Definitions of the information contained in the global, fragmentation and relation allocation schemas of data objects are proposed in [1]. These definitions provide a precise environment for the development of data location and other operational procedures.

3.2 Distribution Transparency

It is desirable that all of the schemas under the global level be transparent to users and application programs. Such systems then appear centralized to their users and offer a high degree of *distribution transparency*. The two aspects of distribution transparency of particular relevance to this paper are: (1) fragmentation design, and (2) location and replication of fragments.

It is a fundamental property of the PIDDB class that the system be capable of automatically selecting and "navigating" to the storage sites without user intervention.[5] It would otherwise be difficult and unreasonable to expect users to associate data objects with their storage locations; it may be impossible in larger systems where allocations are more dynamic. It would also be difficult to preserve global consistency between replicated objects during update operations in PIDDB systems that could not automatically determine the current locations of all replications of an object. Large systems capable of automatically maintaining global consistency must also be capable of offering full data location and replication transparency to the user.

3.3 PIDDB Axiomatic Framework

An axiomatic framework of the PIDDB class of DDBs is presented in Table 1. This framework is partitioned into three classes. The *schema* class develops the application of the reference architecture of Figure 1 to PIDDB systems. The *knowledge assertions* extend this and defines terminology to assist reasoning about object locations. (These notions are developed further in the Appendix as a *knowledge calculus.)* The *atomicity assertions* identify some inherent restrictions of the PIDDB class.

While most of these axioms are self explanatory, some elaboration concerning axiom A10 is required. The restriction this introduces is justified by considering that: (1) the additional data volume required to store the FS of those fragments not held locally is typically insignificant compared with the size of the local fragments, and (2) the data location process would be significantly more complex if A10 did not apply. (see [2].)

3.4 Query Refinement

Various levels of knowledge may be held about objects. A query is said to be *completely unrefined (unrefined), partially refined,* or *completely refined (refined)* according to how well the fragments referenced by the query are known:

- *unrefined:* All the relations in the query are completely unknown (by axiom A8).
- *partially refined:* At least one sought relation is in the LGS and its FS is known (by axiom A7).
- *refined:* All the relations in the query are completely known (by axiom A8).

Three refinement classes are defined by identifying these definitions with knowledge of the corresponding schemas. Queries are therefore classified by their refinement class as *global, fragment,* or *allocation queries.*

The objective of the data location process is to refine the user's global query into a corresponding allocation query. The initiating site would typically refine the query using its locally available information. Any unrefined residues would be relayed to other sites to continue the process. The query is therefore *incrementally refined* as progressively simplified versions are relayed between sites. The data location process terminates when the query is fully refined with respect to each of the objects required to process the query. Detailed definitions of the information associated with each refinement class are necessary for the development of the data location procedure; these are given in [1].

4. TOPOLOGICAL MODEL

The logical network topology defines the information framework within which any data location procedure must operate. This framework specifies what kind of information sites possess about the rest of the network and how this information is organized (i.e. the operational *meta database*). Without such a framework the only possible search algorithm would be one that randomly explored the network until the query was completely refined. The framework proposed below is suitable for modeling the network topologies commonly encountered in practice (e.g. star, ring, hierarchical). 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 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 assigned to multiple N-Sets define the overlapping *articulation points* between those sets. Groups of N-sets overlapping in this way define a *neighbourhood.* Sites are not required to store any data objects in order to be able to initiate or

5. The user may, however, explicitly limit the scope and domain of the search as a safe guard against "expensive" queries.

TABLE 1: PIDDB Axiomatic Framework

Class	Name	Assertion
Schema	*A1*	The global (GS) and fragmentation schemas (FS) of a relation are *stable* (in the sense that the conceptual schema in a centralized database is stable).
	A2	Sites may maintain subsets of the GS called the *local global schema (LGS)*. Queries may only reference relations defined in the LGS. All relations defined in the LGS must be accessible from the site holding that variant of the GS.
	A3	A site may not know the FS of a relation unless that relation is in the LGS.
	A4	The allocation schema (AS) of a relation may vary over time as replications are created or destroyed.
	A5	All sites must share mutually consistent versions of the global schema.
Knowledge	*A6*	An item (schema, relation or fragment) is *known* by a site if it is available locally *(possessed* by the site), or its storage location is known. Sites may similarly know of the existence of other sites.
	A7	A relation is *partially known* (or *partially unknown)* by sites knowing only the FS of that relation.
	A8	A relation is *completely unknown (unknown)* by sites that do not know either the FS or AS of that relation. Conversely sites knowing both the FS and the AS *completely know (know)* the relation.
	A9	A site may not know the AS of a relation unless it also possesses the FS.
Atomicity	*A10*	Knowledge of a FS is atomic. If a site knows the definition of any fragment R_x^α of a relation R_x then it must know the definition of each fragment of that relation.
	A11	Knowledge of an AS is not atomic. Sites are permitted to hold some fragments without knowing the allocations of any other fragments of that relation.
	A12	A site may not possess a fragment of a relation unless it also possesses the corresponding portion of the AS.

execute queries.

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 object names onto the N-sets possessing replications of those objects. These mappings identify objects by their global rather than allocation schema names as H-Sets would typically be used to begin processing unrefined queries. H-Sets may contain any arbitrary collection of N-Sets and neighbourhoods[6]. 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.

Each N-Set is immediately contained within or *possessed* by exactly one H-Set. N-Sets may be either *free* or *bound* in H-Sets. N-Set N_k is free in H-Set H_i if H_i encloses N_k, and no other H-Set enclosed within H_i encloses N_k. N_k is otherwise bound in H_i if H_i encloses N_k.

Sites possess extensive knowledge about their *neighbours*, limited knowledge about certain other N-Sets, and no knowledge about any other site. Each neighbour is described by its communication parameters, usage costs and status. Communication parameters would typically include the network address and the expected connection costs and delays. Usage costs[7] would typically include charges per query or may be on the basis of connect time or the volume of data accessed. Status information[8] would typically include whether the neighbour is up or down and its dynamic utilization (e.g. load average).

Figure 2 demonstrates a sample topology comprising 19 sites, 11 N-Sets and 6 H-Sets. Sites are represented as solid circles, N-Sets as enclosing ellipses and H-Sets as shaded regions. H-sets at the same level of nesting (with respect to H_g) are shaded similarly. An example of an extended neighbourhood connected via articulation points is given in H-Set H_4.

6. Neighbourhoods may not however span H-Set boundaries.

7. While communication and usage cost parameters are likely to prove useful for planning execution strategies, this aspect of PIDDB query processing has not yet been studied closely.

8. Status information would typically be used by the data location algorithm to balance the processing loads between neighbours.

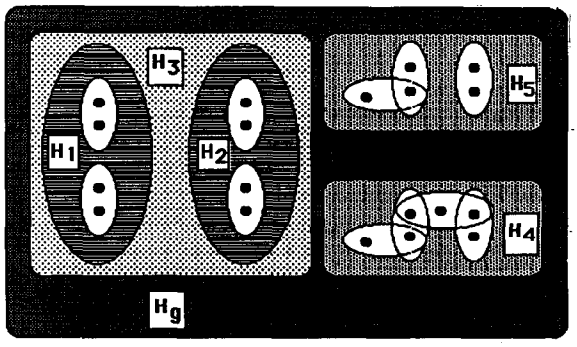


Figure 2: A Sample Network

Guidelines for the establishment of set boundaries are discussed in [1]. Formal schemas defining the information associated with N-Sets and H-Sets have been proposed and applied to the development of the data location and update algorithms [1, 2].

4.1 Gateway N-Sets

Each H-Set containing free N-Sets includes a distinguished gateway N-Set N_g through which all communications with other management domains (i.e. remote H-Sets) are directed. Local gateways are denoted N_{gl}; remote gateways are denoted N_{gr}:

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

There are several reasons for introducing gateways:

1. the volume of traffic between H-Sets may be significantly reduced. The gateway may distribute remotely sourced updates within its local H-Set,
2. secure communications may be established between domains (i.e. authentication protocols can be employed between gateway N-Sets),
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,
- the set of remote H-Sets possessing shadows (§4.2) of locally owned objects, and
- enclosing H-Sets that do not contain free N-Sets[9].

4.2 Types of Replications

Data and meta-objects may be replicated at multiple sites within a PIDDB. Replications are classified as either *active* or *passive* and are referred to as *replications* and *shadows*[10] respectively. All active replications are owned and maintained within the owning H-Set H_o. Active replications are updatable and converge towards consistent values within a settling period after an update. Shadows are possessed by remote H-Sets and are static "snapshots" of active replications. H_o knows where shadows reside and automatically issues refreshed snapshots whenever the active replications are updated. Sites in H-Sets remote to H_o can therefore minimize retrieval costs and delays by accessing a local shadow.

Sites may possess shadow descriptions of remote H-Sets H_r. The version of H_r (remote to and possessed by H_l) is denoted H'_r and differs from H_r in 2 ways: (1) information irrelevant outside H_r is not shadowed (e.g. the specific mappings of data objects onto their containing N-Sets in H_o is not required since all remote communications are directed through gateway N-Sets), and (2) additional information of value within the shadowing H-Set augments H'_r (e.g. identifying shadows with their local N-Sets). H'_r would be generated in N_{gl} from the version received from N_{gr}.

All sites knowing an object (replication or shadow) are granted either *explicit* or *implicit* update rights. Sites knowing a replication are necessarily within H_o and have explicit update rights. These sites may invoke update procedures to revise all replications and issue refreshed shadows. Sites knowing a shadow have implicit update rights meaning that they may issue an update request to H_o. H_o is not however bound to implement the request[11]. It is therefore only necessary to maintain strict consistency between the active replications in H_o.

4.3 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 not 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 network of Figure 2 is shown in Figure 3.

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. 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 (see §5). 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. Each class comprises a complementary set consisting of a knowledge and an "inverse" knowledge LEHT node. Knowledge nodes of type $X(H_r)$ (where X is one of P, R, L or S as described below) define different kinds of information about H_r and where this information resides. The inverse knowledge nodes $X^{-1}(H_r)$ identify those sites possessing class X of knowledge. These nodes are possessed by the sites identified by the $X(H_r)$ nodes. $X^{-1}(H_r)$ information would typically be used to distribute refreshed snapshots of both the user-perceived

9. This is due to the LKV constraints discussed below.

10. This notion of shadows is a generalization of that proposed by CCITT for their Electronic Directory System (CCITT Study Group VII, Question 35, 1984).

11. Some remote requests may have to be refused by H_o in order to preserve consistency, explaining why shadows may not be directly updated.

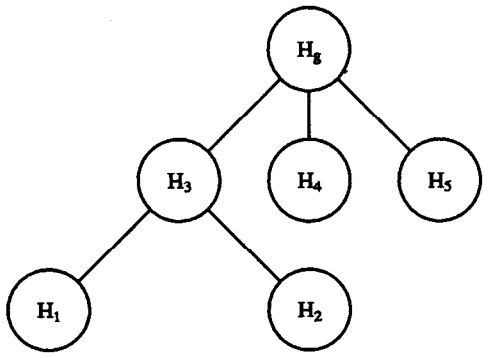


Figure 3: Sample GEHT (Network of Figure 2)

and the meta data objects to the shadowing sites. An analysis of the information necessary for data location and update distribution [1,2] suggests the following eight types of LEHT nodes:

$P(H_i)$ A description of H-Set H_i is possessed locally. Knowledge of H_i is atomic; all fragments $\in$ LGS of S_i that are owned within H_i are described. (i = l or r).

$P^{-1}(H_r)$ Remote H-Set H_r is known to possess a description of the local H-Set H_l. Relevant updates to H_l of global significance can therefore be directed to N_{gr} (e.g. changes effecting the set of available fragments in H_l).

$R(H_r)$ The existence of remote H-Set H_r is known locally. H_r is implicitly accessible via N_{gl} and N_{gr}.

$R^{-1}(H_r)$ Remote H-Set H_r is known to know the existence of local H-Set H_l. Relevant updates to H_l of global significance can therefore be directed to N_{gr}.

$L(H_i)$ An N-Set in H_l is known to possess a description of H-Set H_i (i = l or r). These *local* shadows permit the data location algorithm to readily determine whether H_i can refine any portion of the query. Where H_i is a remote H-Set H_r, L nodes permit the benefits of relaying to H_r to be assessed within H_l. This helps avoid unprofitable remote transactions and reduces the overall cost and delay of data location. H_r is only aware that some remote H-Set possess a shadow description of its local H-Set. The special role of this shadow is entirely local to its containing H-Set.

$L^{-1}(H_r)$ The sites possessing a description of remote H-Set H_r know any local sites in H_l possessing L nodes referencing this description.

$S(H_r)$ Sites possessing a shadow copy of fragment R_x^{α} must also possess an S node to be able to request and receive updates concerning that shadow. This node type identifies the local N-Sets possessing a version H_s of the H-Set H_r owning R_x^{α}. H_s need not be directly known by the shadowing sites and is the minimal subset of H'_r that is sufficient to define $H_o(R_x^{\alpha})$.

$S^{-1}(H_r)$ H-Sets whose fragments are shadowed know which remote H-Sets shadow which fragments. N_{gl} can therefore ensure that refreshed shadow copies are distributed where required following a local update. Relevant updates to H_l can also be directed to N_{gr}. The H-Set subset H_s would be generated and distributed by the gateway sites possessing S^{-1} nodes.

Definitions of the information associated with each type of LEHT node, where they are possessed and the resultant knowledge gained is given in Table 2. Underscored quantities in the schema definitions indicate that the *name* rather than the *value* of the object is referenced. The knowledge operators (ϕ, Φ, κ and K) used throughout this section are defined in the Appendix. This table indicates that $L(H_r)$ nodes, for example, are maintained within H-Sets that may possess shadow copies of a remote H-Set. The schema definition sayes that the set of sites possessing some version of remote H-Set H_r is known at some site(s) within a local N-Set N_k. The possession constraint states that the LEHT node must be possessed by some site S_n within the local N-Set N_l. The knowledge constraint defines the information gained by virtue of possessing the node; in this case the fact that remote H-Set H_r is known by the site in H_l possessing the $L(H_r)$ LEHT node.

While the schema definitions and knowledge constraints of the P^{-1} and R^{-1} nodes are identical, their semantics and applications are quite distinct. The sets K in their schema definitions are not Φ because of the constraint that only the gateway sites of another H-Set may be known.

Some sample LEHTs for the network of Figure 2 and GEHT of Figure 3 are given in Figure 4.

4.4 Local Knowledge View Constraints

There are a series of *local knowledge view constraints* defining the minimal knowledge that each site must possess. These constraints ensure that objects in other H-Sets can be located, that consistency is maintained between replications and that shadowed objects may be refreshed. These requirements are met by imposing two *completeness constraints* on the set of LEHTs: all GEHT nodes (H-Sets) must be described by at least one LEHT P node, and the relative structure of the GEHT must be deducible at all sites. The following knowledge constraints are proposed to satisfy these requirements:

1. all sites within N_{gl} possess the knowledge kernel (see §4.1),
2. gateway N-Set sites possess P, L or R LEHT nodes for each descendant of H_g (i.e. this constraint defines what the knowledge kernel contains about H_g).
3. all sites not in N_{gl} know N_{gl} (i.e. know the network addresses of each of the sites in N_{gl}.)
4. at least one site in each neighbourhood possesses a description of the immediately enclosing H-Set H_l,
5. at least one site in each neighbourhood knows:
   a. at least the immediate ancestor H-Set, and/or
   b. all immediate descendent H-Sets.

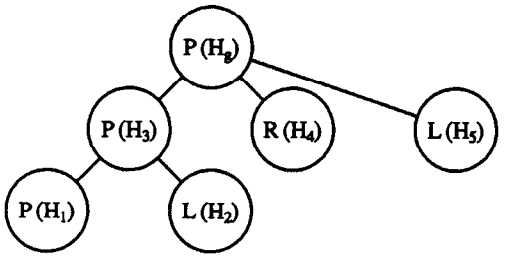
These constraints are weak in the sense that, from an information theoretic point of view, a less stringent set of requirements could be deduced. The proposed set replicates some knowledge for operational simplicity and robustness.

5. DATA LOCATION ALGORITHM

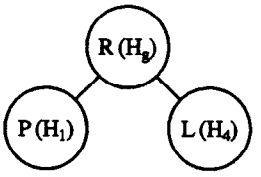
The basis of a data location algorithm is presented to illustrate the typical application of the LKV. The algorithm relies on an heuristic *search hierarchy* in which the scope of the search is progressively increased until the sought fragments are either located or found not to exist. This hierarchy has the following levels:

TABLE 2: LEHT Node Definitions

Type	Schema	Constraints	
		Possession	Knowledge
$P(H_l)$	$<\underline{H_l}, H_l>$	$\exists S_n \in N_l$: $\phi^n P(H_l)$	$\exists S_n \in N_l$: $\phi^n H_l$
$P(H_r)$	$<\underline{H_r}, H'_r>$	$\exists S_n \in N_l$: $\phi^n P(H_r)$ $\forall S_n \in N_{gl}$: $\kappa^n \Phi(P(H_r)) \wedge \kappa^n N_{gr}$	$\exists S_n \in N_l$: $\phi^n H'_r$
$P^{-1}(H_r)$	$<\underline{H_r}, K(H_l)>: K \in N_{gr}$	$\forall S_n \in N_{gl}$: $\phi^n P^{-1}(H_r)$	$\forall S_n \in N_{gl}$: $\kappa^n K(H_l)$
$R(H_r)$	$<\underline{H_r}>$	$\exists S_n \in N_l$: $\phi^n R(H_r)$ $\forall S_n \in N_{gl}$: $\kappa^n \Phi(R(H_r))$ $\wedge \kappa^n N_{gr}$	$\exists S_n \in N_l$: $\kappa^n H_r$
$R^{-1}(H_r)$	$<\underline{H_r}, K(H_l)>: K \in N_{gr}$	$\forall S_n \in N_{gl}$: $\phi^n R^{-1}(H_r)$	$\forall S_n \in N_{gl}$: $\kappa^n K(H_l)$
$L(H_l)$	$<\underline{H_l}, \Phi(H_l)>: \Phi \in N_k \in H_l$	$\exists S_n \in N_l$: $\phi^n L(H_l)$	$\exists S_n \in N_l$: $\kappa^n H_l$
$L(H_r)$	$<\underline{H_r}, \Phi(H'_r)>: \Phi \in N_k \in H_l$	$\exists S_n \in N_l$: $\phi^n L(H_r)$	$\exists S_n \in N_l$: $\kappa^n H_r$
$L^{-1}(H_r)$	$<\underline{H_r}, \Phi(L(H_r))>: \Phi \in H_l$	$\exists S_n \in N_k \in H_l$: $\phi^n L^{-1}(H_r)$ $\wedge \phi^n P(H_r)$	$\exists S_n \in N_k \in H_l$: $\kappa^n K(H_r)$
$S(H_r)$	$<\underline{H_r}, \Phi(H_s)>:$ $\Phi \in N_k \in H_l \wedge$ $R_x^\alpha \in H_s \wedge H_s \subseteq H'_r$	$\exists S_n \in N_l$: $\phi^n S(H_r)$ $\forall S_n \in N_{gl}$: $\phi^n S(H_r) \wedge \kappa^n N_{gr}$ $\forall S_n \in \Phi(H_s)$: $\phi^n P(H_s)$	$\exists S_n \in N_l$: $\kappa^n H_s$
$S^{-1}(H_r)$	$<\underline{H_r}, K(H_l), \{R_x^\alpha: \exists S_n \in H_r:$ $\phi^n R_x^\alpha \wedge \underline{H_l} = H_o(R_x)\}>:$ $K \in N_{gr}$	$\forall S_n \in N_{gl}$: $\phi^n S^{-1}(H_r)$	$\forall S_n \in N_{gl}$: $\kappa^n K(H_s)$



(a) LEHT for $S_n \in H_1$



(b) LEHT for $S_n \in H_5$

Figure 4: Sample LEHTs (Network of Figure 2)

- The LKV is used to attempt local query refinement at the initiating site S_{init}.
- Any residues remaining after local refinement are distributed to sites in a local N-Set to solicit sufficient knowledge.
- If the query cannot be completely refined within any local N-Set, then the residue is relayed through an articulation point to other N-Sets in the local neighbourhood NH_l. This process continues until either the query is fully refined or NH_l may not be explored further.
- If the query cannot be fully refined within NH_l then an exhaustive search from the root (H_g) of the GEHT is begun. (Interrogating the LKV and exploring NH_l is analogous to exploring the GEHT from the leaves upwards towards the root. If this search fails, due to the incompleteness of the available LEHTs, then it is necessary to begin a top down search from the root of the GEHT.)

An outline of the data location procedure is given below. A more complete algorithm that integrates refinement of the global query with LEHT-based navigation is described in [2]. This version assumes the user's query is already refined to the fragmentation level, and that only a single fragment R_x^α is sought:

1. if $\kappa^{init} R_x^\alpha$ then end.
2. $\forall \phi^l P(H_i)$: if $R_x^\alpha \in H_i$ (or H'_r if i=r) then end.
3. $\forall \phi^l L(H_r)$: relay to $S_n \in H_l$: $\phi^n P(H_r)$ to interrogate H'_r. Await response and end if R_x^α now located.
4. $\forall \phi^l R(H_r)$: relay to $N_{gr} \in H_r$ to interrogate H_r. Await response and end if R_x^α now located.
5. explore local neighbourhood until $\kappa^{init} R_x^\alpha$ or NH_l exhausted. End if R_x^α now located.
6. relay to N_{gl} to begin an exhaustive search from the root of the GEHT (i.e. from H_g). The user would typically interact or otherwise limit the extent (and hence cost and delay) of the

search.

This algorithm would be invoked as a preliminary phase of processing any query. It is therefore important to minimize the associated processing delay to avoid creating a data location "bottleneck". Some aspects of this problem are reported in [2].

6. CONCLUSION

The PIDDB class of replicated distributed data bases has been identified and developed. It is characterized by limiting the knowledge available to each site to *subsets* of the sites and objects in the network. The PIDDB 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. An heuristic network topology that groups processing sites into Neighbouring and Hyper-Sets (N-Sets and H-Sets) has been proposed to ensure that the data location procedure is tractable. A tree description of this topology has been presented. It has been shown how this representation can be transformed into the local knowledge trees possessed by the sites. The application of these trees to an algorithm for determining fragment locations during query processing has been demonstrated.

A mechanism has also been proposed [1] 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 like any other shadowed object. The proposed mechanism defines distributed objects as views [3] on the global schema.

7. ACKNOWLEDGEMENT

The support of Telecom Australia to undertake this work is appreciated. The permission of the Chief General Manager, Telecom Australia to publish this paper is also acknowledged.

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

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

5. M. Jarke and J. Koch, Query Optimization in Database Systems, *ACM Comp. Surveys 16* , 2, (Jun. 1984), 111-152.

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

7. C. T. Yu and C. C. Chang, Distributed Query Processing, *ACM Comp. Surveys 16* , 4, (Dec. 1984), 399-433.

8. C. T. Yu, C. C. Chang, M. Templeton, D. Brill and E. Lund, Query Processing in a Fragmented Relational Distributed System: Mermaid, *IEEE Trans. on Software Eng. SE-11* , 8, (Aug. 1985), 795-809, IEEE.

9. S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, 1985.

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. E. F. Codd, A Relational Model of Data for Large Shared Data Banks, *Comm. ACM 13* , 6, (1970), 377-387.

14. C. Mohan and R. Popescu-Zeletin, Impact of Distributed Data Base Management on the ISO-OSI And ANSI/SPARC Frameworks, *Proc. 15th Hawaii International Conf. on Systems Sciences*, Honolulu, Jan., 1982, 532-542.

15. ANSI, Reference Model for DBMS Standardization, *ACM SIGMOD Record 15* , 1, (Mar. 1986), 19-58.

16. CODASYL, *Data Base Task Group Report*, ACM, New York, Apr., 1971.

APPENDIX: KNOWLEDGE CALCULUS

The development and analysis of query execution strategies in a PIDDB often requires reasoning about the knowledge possessed by the processing sites. A propositional calculus is presented below that simplifies the expression and manipulation of that knowledge. Four new *knowledge operators* in two classes are defined.

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 operators, κ and ϕ, are useful for expressing or testing this knowledge:

- κ — κ^nOBJ is TRUE if object OBJ is known at site S_n (meaning that OBJ is either *possessed* by S_n or that the identity of another site possessing OBJ is *known* by S_n).
- ϕ — ϕ^nOBJ is TRUE if object OBJ is possessed by site S_n.

These operators are related by the inference rule:

$$\phi^n \text{OBJ} \rightarrow \kappa^n \text{OBJ}$$

The truth of the assertions expressed with these operators is *independent* of *where* the assertion is stated. For example, $\phi^3 R_x^5$ may be true but unknown at S_4. OBJ may be either a data or meta object, or a site. If OBJ is a site then the ϕ operator is not applicable as the PIDDB framework does not support the notion of sites "possessing" each other.

The second class of operators, K and Φ, are macros that define the set of sites knowing or possessing OBJ. This notation simplifies the expression of knowledge schemas and constraints:

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