Administration and Autonomy In A Replication-Transparent Distributed DBMS

Kenneth R. Abbott, Dennis R. McCarthy

Computer Corporation of America
Four Cambridge Center, Cambridge, MA 02142

Abstract

Administrative issues are of vital importance to organizations adopting distributed database technology. Most research systems and emerging commercial DDBMSs have assumed site autonomy as a guiding principle. This paper presents some general problems associated with autonomy and administration in a DDBMS, and discusses the incompatibility between replication transparency and site autonomy. In a DDBMS which supports replication transparency, the degree of site autonomy in a system should properly be a policy decision made by system administrators. Parameterization of the degree of site autonomy in a DDBMS involves careful design of a mechanism involving storage structures, operations, and authorization. A mechanism which extends the ANSI SQL authorization model is described, and examples of how users can use the mechanism to implement both centralized and decentralized administration policies are presented.

1. Introduction

As distributed DBMS technology becomes more widely available in commercial systems, issues related to administration, configuration, and operation of distributed systems are becoming increasingly important. Pragmatically, administrative issues will probably be the hardest problems faced by organizations attempting to adopt distributed database technology.

Research in distributed DBMS has focused on the problem of access to distributed data (the problem of translating and decomposing a database update or query into local updates or retrievals at a set of communicating sites), but has not paid much attention to administration of distributed data. A few papers have discussed general organizational problems of administering a distributed system [1] [2], but little attention has been paid to the technical aspects of DBMS functionality that are useful for implementing distributed administration policies [3]. Lack of work in the area may reflect a bias that administrative and operational problems are not technically interesting, and may also be a consequence of the fact that the problems are not evident or important in an R&D environment where most work on distributed systems has occurred.

Single-site DBMS solutions to administrative problems have evolved but they do not readily scale up to the distributed case for several reasons:

- Administration is complicated by fragmentation, replication, and allocation of data to sites.
- Clear hierarchical lines of control at a single site do not exist in a distributed system.
- In the distributed case, operations such as resource allocation and backup/recovery decompose into distinct global and local components that are not distinguished in the single-site case.

As was recognized early on in the R* project, the issue of site autonomy [4] is a seminal issue for the designer of a distributed DBMS. The degree of site autonomy has a profound effect on both the data access and administrative aspects of the DBMS. R* and emerging commercial systems (Ingres/Star and Oracle*) adopt site autonomy as a basic design principle. While the assumption of site autonomy makes sense as a basis for migrating existing centralized databases into a confederated distributed system, it affects the ability of the system to hide distribution and replication. Furthermore, the strong identification of autonomy with sites obscures a useful distinction between the logical concept of autonomy and the physical concept of site. In fact, considerations of replication transparency force a distinction between logical and physical concepts of autonomy and administration. Logical level administrative operations such as naming, backup, and reorganization of tables necessarily involve cooperation (and hence loss of autonomy) of sites which store replicated data. Local, physical-level administrative operations (such as resource allocation at a site) may be performed autonomously even in the presence of replicated data.

During the design of Adaplex [5], a homogeneous, replication transparent DBMS, replication transparency [1] and site autonomy were found to be antagonistic goals. The goal of fully transparent distribution and replication of data strongly affects administrative DBMS functions as well as data access functions[2]. Our work on Adaplex differs from other systems because we began with the assumption of replication transparency and investigated the implications of this assumption on administration and autonomy. Our approach was to parameterize the "degree of autonomy" in the distributed system and allow database administrators to trade-off between autonomy and replication transparency on a per-installation basis. The key components of our parameterized approach are:

- introduction of the *database* storage structure (the logical and administrative unit of autonomy) which may be mapped to physical sites in various ways, depending on the degree of site autonomy or replication transparency desired;
- definition of intermediate storage structures and operations which support a clear delineation between logical level (site-independent) administration and physical level (site-specific) administration;
- extension of the ANSI SQL authorization mechanism to encompass these storage structures and operations

Even with this flexible approach, it is necessary to make a base assumption about the trustworthiness of the DBMS: with full, "paranoid" site autonomy, the DBMS at one site never trusts the DBMS at another site. However, many algorithms and protocols for implementing transparent replication must implicitly "trust" DBMS software at multiple sites. Therefore, the parameterized mechanisms described here cannot achieve "paranoid" site autonomy (because users must trust DBMS software at remote sites) but the mechanisms can achieve "functional" site autonomy (users can issue commands and perform operations just as they would in a fully autonomous system).

In section 2, we will motivate and enumerate some of the major technical issues in distributed database administration. In section 3, the role of operations, storage structures, and authorization in database administration are discussed in general terms. Section 4 describes the design of the Adaplex storage structures and authorization mechanism, and section 5 gives examples of how the Adaplex mechanism can be used to implement a variety of administrative policies.

2. Issues in Distributed Administration

Techniques for administering a single-site DBMS have become well established as DBMS has become an essential element of data processing. A key element in administration is identification of various administrative roles and the association of various capabilities with them: DBMS operators can perform DBMS operations that regular users can't; system administrators can do things that operators can't; etc. A common feature of single-site administration schemes is that there is an implicit or explicit hierarchy of roles - there is usually one role — that subsumes all others.

However, in a distributed system, roles that are bundled in the single-site case become unraveled. Responsibility for the logical structure and integrity of the distributed database is a system-wide role, while responsibility for detailed administration tends to be distributed to localized roles at individual sites [2]. What was a hierarchy in the single-site case is better understood as a balance of powers between "spheres of control" [7] scattered throughout the distributed system. Notice that a distributed system can be administered in either a centralized or decentralized fashion. However, single-site administration techniques are not directly applicable even in a centrally administered distributed system.

The following paragraphs summarize some of the major administrative issues which come into play in a distributed system.

---

[1] For the purposes of this paper, we adopt the definitions of distribution and replication transparency applied in [5]:

- Users see a logically centralized view of the catalog (data dictionary) and the database. Names of objects share a global namespace, and site names *do not* appear in object names.
- Users are not associated with sites. Users may log in to the DBMS and access data without regard for their location (login site) or the location of data.
- When a user accesses a fragmented table, the DBMS automatically decomposes the query into accesses to fragments of the table. For retrievals, the DBMS merges the results so that the user sees a single, logically integrated table.
- When a user accesses replicated data, the DBMS automatically transforms the access into an access to a particular copy of the data item at some site. On updates, the DBMS automatically propagates and merges updates to all copies so as to maintain one-copy serializability.

[2] It is interesting to note in this context that R* researchers originally intended to support replicated data, but eventually abandoned the effort [6].

Replication Transparency vs Site Autonomy

In a distributed system, it is desirable to consider distribution and replication as physical details of an object's representation. In the query language, logical objects (e.g. tables) are manipulated without regard for their physical representation. Replication transparency means that the user is unaware that a logical object may be represented by multiple physical copies. Some administrative implications of replication transparency conflict with site autonomy:

- *Access control* - with site autonomy, sites can control a user's ability to access individual copies of data. This violates transparency because a user may be able to access some, but not all, copies of a data item, and the behavior of a query may vary depending on the copy selected. Furthermore, the DBMS cannot guarantee mutual consistency of replicated data if it cannot update all copies in an identical fashion.

- *Operation* - autonomous operation of sites can cause inconsistencies between copies of a data item. For example, individual copies of a data item cannot be restored from tape, because the restored copy will be out synch with other copies. Possible solutions (prohibiting uncoordinated restores, or forcing all sites to restore simultaneously) violate site autonomy.

- *Resource allocation* - physical resources such as disks are controlled by individual sites. In general, the resources at a site will be used in a variety of applications; the distributed DBMS will be merely one activity among many being performed at a site. The need to control allocation of site resources to the DBMS must be balanced against the DBMS need to maintain identical copies of data at different sites.

Local View vs The Big Picture

In the distributed case, there is no "all-seeing, all-powerful" person with total authority over all details of distributed system operation. In general, designing, implementing, and using a distributed database involves the interaction of several people in different roles. For example, the designer of a distributed database may have the "big picture" in terms of the overall logical design (e.g. global access paths for distributed queries, system-wide requirements for availability and recovery, etc) but cannot be expected to have the detailed knowledge of particular sites (e.g. names of disk units, filenames, etc) necessary to implement the design.

Top Down vs Bottom Up Database Design

Top-down design and implementation requires centralization of authority in the database designer. The designer is responsible for defining the logical structure of the database and allocating and replicating data at sites. Individual sites are treated as anonymous resources to be used as the global design dictates.

Bottom-up design and implementation is done in a decentralized fashion: individual pieces of the database are independently created at various sites, and the system administrator is responsible for passively reacting to decisions made locally at sites. Bottom-up design mode is compatible with full site autonomy.

3. Overview of Distributed Database Administration

Although the query language is often considered the only "important" interface to a DBMS, most vital administrative DBMS functions are not accessible from the query language. Consider the case of the SQL [8] language, which is purely a query language in the sense that it deals only with the "logical" objects presented by a DBMS: tables, indexes, authorization ids, etc. It does not address the way in which a particular DBMS maps these abstract relational concepts to structures or facilities provided "outside" the DBMS, nor does it address issues pertaining to operating the DBMS in a failure-prone environment (viz, the real world). Commercial DBMS's based on SQL (e.g., DB2, Oracle) have extended SQL to address these administrative aspects of database management. In fact, for most commercial DBMSs, more linear feet of documentation are dedicated to administrative functions than to data access functions. From the standpoint of an organization using a DBMS, easy and reliable administration of the DBMS may be as important as accessibility of data in the DBMS.

In this paper, we will focus on on the following aspects of DBMS administration:

- *Resource Allocation and Utilization* - The issue of resource allocation reflects the need to share physical resources such as disks, tapes, processors, etc between various functions, one of which is data management. There is a need to be able to conveniently allocate needed resources to a DBMS and to be assured that the DBMS will not exceed its allocation. The issue of resource utilization reflects a related need to exercise fine-grained control over how the DBMS uses particular resources. The placement of data on disk, in particular, can be an extremely important factor in DBMS performance.

- *DBMS Operation* - Administering a database involves performing many operations. Some of these

operations involve interaction with both the DBMS and the host system. Many of these operations do not directly involve logical data objects such as tables. Some major categories of DBMS operations are: installation, system and site control, routine maintenance, and recovery.

3.1 The Role Of Storage Structures in Administration

To support administrative DBMS functions, most commercial DBMSs introduce "intermediate" storage structures into their storage architecture (e.g. DB2 has databases, table spaces, and storage groups [9]; Oracle has partitions, spaces, and clusters,[10] etc.). Administrative operations work on intermediate storage structures rather than logical objects. In effect, intermediate structures are the objects of administrative "design" of the database, just as physical structures are objects of physical design.

In a distributed DBMS, the issues related to intermediate storage structures are quite complex. The mapping from logical objects to physical storage has an additional layer of complexity introduced by the need to locate data at sites. Also, the physical representation of an object may span site boundaries. Support of fully transparent distribution and replication of data further complicates the role of intermediate storage structures. There are needs both for distributed intermediate structures (to facilitate administration of logical objects - e.g. groups of related tables) and for localized structures (to facilitate placement of data on disk at a particular site.)

3.2 Authorization And Administration

To tailor the configuration of a DBMS to conform to a particular organization's needs, an authorization mechanism is provided.This mechanism allows administrators to selectively control a user's ability to access data and/or to perform administrative operations. SQL defines an authorization mechanism based on the granting and revoking of privileges on objects which is derived from the System R authorization mechanism [11]. The SQL authorization mechanism supports access control to data in tables and indexes. When administrative functions are factored into a DBMS, there is a parallel need to control access to and operations on objects which are not visible in the query language. To serve these needs, the basic SQL authorization mechanism must be extended [12].

In extending the SQL mechanism, the DBMS designer must make design choices along several dimensions :

- How are users and data identified and named for purposes of authorization [3] [13]?
- What intermediate storage structures does the DBMS provide and what primitive operations are allowed on them?
- How are are various administrative roles represented for purposes of authorization?

There are innumerable subtleties in design choices along any dimension, and there are often unexpected connections between superficially unrelated design choices.[3]

To achieve the goal of designing a flexible mechanism that works well in many different situations, it is necessary to make compromises along all dimensions. The measure of success is not in making the mechanism optimal for any particular application, but in making it close to optimal for a spectrum of anticipated applications.

4. The Adaplex Administrative Concepts and Mechanisms

The Adaplex authorization model and mechanisms were designed to allow implementation of policies with the following requirements:

1. Allow a database designer the ability to specify a logical and physical design without requiring detailed knowledge of sites that may eventually store data.
2. Allow a naive user to perform simple operations (e.g. CREATE TABLE without fancy options) without intervention of site or database administrators.
3. Allow an "indifferent" site administrator to permit the system to place data at the site by issuing simple commands, without requiring knowledge of the overall database design.
4. Allow a "concerned" site administrator to have fine-grained flexibility and control over physical placement of data at the site, but deny him the ability to override logical or physical design decisions made by a database designer.

Notice that it is a policy decision (at each installation) to allow or deny the capabilities implied by these requirements.

---

[3] An example of an unexpected connection between design decisions was the discovery that implementation of "protection views" in R* was significantly complicated by considerations of site autonomy [14] [15]. "Protection views" are objects of authorization which can be used to provide users with a restricted view of tables which they would not otherwise be authorized to access.

The remainder of this section is organized as follows. First we introduce a generalization of the notion of site autonomy. Then we discuss administrative roles. Finally, we describe the authorization mechanism used to implement these roles.

4.1 Storage Structures

In Adaplex, a *database* the unit of administrative autonomy. It is a self-contained subset of the distributed DBMS with a unique system-wide name. A database encapsulates a namespace which contains tables, users, indexes, etc. A database contains both user data and the catalogs which describe the data. Catalogs for a database are physically replicated at every site which stores any data stored in the database. There is no "hard-wired" association between sites and databases: a site may store data from zero, one, or many databases. However, it is useful the think of a database as the logical analog of a site, in the following senses:

- Each database contains a disjoint namespace. References from one database to another must be qualified by database name.
- Users are validated relative to a database (i.e. a user is named object within a database's name space).
- Databases are decoupled from each other — activity in one database cannot affect data or users in another database.

Thus the database abstracts the various logical notions of autonomy (e.g., independent operation, separate namespace) without implications about physical representation.

Using databases, users have considerable flexibility in configuring a distributed DBMS. If all data is placed in a single database, then the system acts like a logically centralized DBMS with a single namespace and globally replicated metadata. If each site is associated with exactly one database, then the system acts like a loose federation of autonomous sites, where each site maintains its own namespace, users, and data. It is also possible to define intermediate configurations where multiple databases are stored at multiple cooperating sites.

A table exists within a database and can be fragmented and replicated (within the containing database) as described in [5]. A copy of a fragment of a table is stored at a site in a *pagespace*, which is the Adaplex intermediate storage structure that represents actual physical storage at a site.

The following table summarizes the intermediate structures used to represent and store table and index data:

The configuration of a distributed DBMS is determined by control over pagespaces as well as by the use of databases. Sites can maintain a degree of autonomy by retaining control over their pagespaces. To centralize database administration, sites must give up control of pagespaces. Adaplex provides flexible control over pagespaces through its authorization mechanism.

Storage Structure	Description
Database	A self-contained environment (virtual DBMS) in which users, tables, etc, are defined. A database can exist at one or more sites, and a site may store one or more databases.
Fragment	A horizontal partition or "slice" of a table. A fragment is defined by a predicate defined on columns within a single table.
Fragment Group	A set of logically related fragments which are distributed, replicated, and recovered as a unit.
Pagespace	A set of disk pages at a site used to store fragments with similar characteristics. A pagespace is defined in terms of host system files, which may have been created outside of Adaplex.

Figure 1. Adaplex Storage Structures

4.2 Administrative Roles

In a centralized DBMS, the roles played by various classes of DBMS users and operators are fairly well understood and established. In DB2, there is a hierarchy of roles, with the "system administrator" being all-seeing and all-powerful. In a distributed DBMS, the roles are more differentiated and are no longer hierarchical. In particular, the centralized system administrator role bifurcates into two roles: a global administration role and a site administration role [2]. The following list summarizes administrative roles identified for Adaplex:

- **Global Administrator** - This role is responsible for managing a distributed Adaplex system. The role has complete control over all logical objects in the system, but does not directly control allocation of resources at a particular site. Responsibilities include system-wide installation and configuration.

- **Database Administrator** - This role is responsible for a single database. The role has control over all objects in the database. Responsibilities include database design, maintenance of tables and indexes, and fragmentation and replication of data.

- **Site Administrator** - This role is responsible for managing a single site. The role has complete control over physical resources at the site, but may have limited control over the mapping of logical structures to physical structures. Responsibilities include site-specific installation and configuration, creation and maintenance of local storage structures, and local operations.

The implementor of an authorization policy has the option of assigning multiple administrative roles to a single user or may choose to assign different administrative roles to different users. In the latter case, administration of the DBMS requires active cooperation among the various administrators. The authorization mechanism makes it possible for non-cooperating administrators to effectively "veto" operations that might impinge on his or her sphere of control. For example, data may not be stored at a site until the proper physical structures have been initialized at the site. If the authorization policy separates the site and database administration roles, then a site administrator may block the storage of data at his site by refusing to create the requisite structures. However, if the site administrator does create the structures, the site administrator still does not have any control over the stored data since control over the "logical" data remains vested in the database administrator.

4.3 Authorization Mechanism

The Adaplex authorization mechanism is based on the ANSI GRANT/REVOKE model [8], and is strongly influenced by (but not identical to) the DB2 mechanism [12]. In the simplest form of this mechanism, *users* perform *operations* on *objects*. A *privilege* is a ternary relation between a *user*, an *operation*, and an *object* which represents that the user is *authorized* to perform the operation on the object. There are numerous extensions to this basic model, some of which have been implemented in DB2 (e.g. aggregate privileges), and some of which have been proposed (e.g. classes or groups of users [16]). Adaplex provides aggregate privileges to support the administrative roles described above.

Wilms and Lindsay [16] briefly discusses the issue of authorization in a distributed, site autonomous system. The approach taken by Adaplex differs from that approach because authorization is also site transparent.

The basic user/privilege/object mechanism has the drawback that all privileges must be attached to DBMS objects. This leads to bootstrapping problems: installing or configuring the DBMS requires authorization, but the DBMS may not be functional or may contain no objects. DB2 solves this problem by imprinting itself with the host operating system id of the "superuser" who initially configured the DBMS, and recognizing that id as a special one. This technique is very awkward in the distributed case, because the same user can access the DBMS from many different sites with different host system ids.

To generalize this notion, the concept of an *authority* was introduced. The appropriate privilege or authority is required to perform an operation. Authorities are authenticated using passwords, without reference to a database. They are "hardwired" into the DBMS, and cannot be granted or revoked. Adaplex recognizes two authorities: system and site. *System Authority* corresponds to "global administrator" role. It controls addition and deletion of sites, creation and replication of databases. *Site Authority* corresponds to the "site administrator" role. It controls local storage structures and operations at a site.

A user may gain site authority over his login site if he knows the password for the site. (The password for the site is stored at the site during the installation procedure.) Notice that Adaplex will only grant site authority to users actually logged in at the site. Using the site authority, privileges to perform operations on pagespaces can be granted to users in databases. In this manner, control over local resources can be transferred from the site administrator to a database administrator.

5. Implementing Administrative Policies

In designing the authorization mechanism, it became clear that there is an unavoidable conflict between the administrative roles described above. Site administrators want

total control over resources at their site; global and database administrators want to be able to commandeer resources from sites as dictated by the global needs of the system. We attempt to resolve this conflict by division of labor and authority between administrative roles. In general, configuring a site as part of a distributed Adaplex system will require the execution of some operations requiring authority at a site, some commands requiring authority on specific objects, and some commands requiring authority on both simultaneously.

5.1 Site Administration

When a copy of a fragment group is stored at a site, each fragment in the fragment group must be assigned to a pagespace at the site. Before the fragment group is stored, the pagespaces must be created, and the user placing the fragments in the pagespaces must be granted privileges to use the pagespaces. A site administrator has control over the creation and use of pagespaces at a site. Initially, site authority is required to create a pagespace or to store a fragment in a pagespace.

To retain local control, the site administrator creates pagespace (using site authority) and grants database administrators the privilege to store fragments in the pagespace. In this case, storing a copy of a fragment group at a site is a two stage process, with responsibility divided between the site administrator and the database administrator. First the site administrator must create the pagespaces and grant privileges to use them. Then the database administrator stores a copy of the fragment group at the site in the pagespaces. The site administrator can veto the placement of a fragment group at a site by refusing to set up the pagespaces. Local control of a site is desirable from the standpoint of site autonomy, but is undesirable because it requires frequent close collaboration between database administrators and site administrators.

Alternatively, the site administrator can grant privileges on the site to users in a database (that is replicated at the site). For example, granting the privilege to create pagespaces on the site makes it possible for users to create pagespaces (in a single database) at the site without explicit intervention by the site administrator.

A logically centralized DBMS can be configured by granting privileges for site operations on all replication sites of a database to the database administrator. The database administrator then has control over the "logical" data as well as its physical placement.

5.2 Distributed Database Design and Implementation

A distributed database can be designed, implemented, and used in either a centralized fashion or a decentralized fashion. The activities of design, implementation, and use are independent in the sense that it is possible (for example) to design a database centrally, implement it in a decentralized manner, and use it in a centralized fashion. Since a DBMS does not usually directly support database design (independent of implementation), in this section we focus on database implementation and usage, and describe schematically how the authorization mechanism can accommodate both centralized and decentralized styles of administration.

To illustrate how the authorization mechanism is used, we distinguish two *usage modes* and two resource *control modes* for a distributed database. The usage modes are:

1. *Interactive mode:* In this mode, users can create and drop tables and views. There is no single database designer who has the universal view of all objects in the database. This mode of usage might occur within a department where the DBMS is used as a decision support tool and ad hoc queries are common.

2. *Production mode:* In this mode, the schema for a database is designed and implemented specifically to support an application. There is a database administrator for each database who has a universal view of database contents and who is responsible for monitoring and implementing all changes to the database. This mode of usage might occur in a large company developing MIS applications for in-house clients.

Orthogonal to the usage modes described above are "resource control modes" for processors and disks in the distributed system:

1. *Cooperative mode*: Resources are used cooperatively and are allocated on an as-needed basis. All the resources in the distributed system are controlled by the same organization, and can be used interchangeably. This mode might correspond to a DBMS running on a small LAN where all processors are controlled by the same department in a company.

2. *Autonomous mode*: Resources are strictly allocated along site boundaries and each site has strong control over its own resources. This mode might correspond to a DBMS running a geographically distributed network, where each site is controlled by a different corporate organization.

Role	Action
Database Administrator	Create user profiles in the database for all interactive users. Grant privileges to all users which allow them to create tables, indexes, and fragment groups.
Site Administrator	At each site where data will be stored, create default pagespaces and grant interactive users the privileges to place fragment groups in the pagespaces.
SQL User	As desired, define new tables and indexes. Use default fragmentation and replication parameters so that data is automatically placed in default pagespaces created by site administrators.

Figure 2. Interactive Usage/Cooperative Resource Control Scenario

The following scenarios are constructed by considering combinations of usage modes and resource control modes. In all of them, it is assumed that the global administrator has already created a database and replicated it at all of the sites where data is to be stored.

Figure 2 describes the combination of interactive and cooperative modes. In this scenario, "naive" interactive use is accommodated by heavy use of default values. Database and site administrators collaborate in setting up author-ization and storage structures so that default mechanisms work transparently.

Figure 3 describes the combination of interactive and autonomous modes. In this scenario, interactive usage is accommodated by liberally granting privileges on logical objects (tables) but limiting privileges to create pagespaces to site administrators. By NOT creating default pagespaces, site administrators force users to be explicit in specifying physical storage. If a user wants to store data at a site, the user must get the site administrator at the site to create pagespaces and/or grant privileges on pagespaces at that site.

Role	Action
Database Administrator	Create user profiles in the database for all interactive users. Grant privileges to all users which allow them to create tables, indexes, and fragment groups.
Site Administrator	When a user wants to store data at a site, the user negotiates with the site administrator, who creates a pagespace at the site and grants the privilege to place data in the pagespace to the user (and no other user).
SQL User	Create tables and explicitly specifies fragmentation, replication, and creation of fragment groups. For each fragment group, replicate it at sites by explicitly placing it in the pagespace(s) created by the site administrator.

Figure 3. Interactive Usage/Autonomous Resource Control Scenario

Role	Action
Site Administrator	At each site which will store data, grant the privilege on the site to allow the database administrator to act as site administrator.
Database Administrator	Design the logical schema, and determine fragmentation, fragment grouping, and placement of fragment groups at sites. Create pagespaces at sites as needed. Create tables, fragments, and fragment groups, and explicitly place fragment groups in previously defined pagespaces. Grant ordinary users privileges to access the logical objects (tables) but not the underlying pagespaces.

Figure 4. Production Usage/Cooperative Resource Scenario

Figure 4 describes the combination of production and cooperative modes. In this scenario, all control over resources is centralized by granting the site administrator privilege on each site to the database administrator. By NOT granting privileges on pagespaces to users, users are prevented from creating new tables without collaboration with the database administrator.

Figure 5 describes the combination of production and autonomous modes. In this scenario, the database administrator and site administrators share responsibility for designing and implementing the database. Collaboration is forced by NOT granting privileges on sites to the database administrator.

5.3 Operational Example: Backup and Recovery

Even routine DBMS operations (e.g. backup, software installation, log maintenance, etc) can be complicated by distribution, autonomy, and replication. To illustrate this point, this section uses backup and recovery operations as examples of a basic DBMS operations that are affected by these considerations.

Backup and recovery must work on fragmented, distributed, replicated, and geographically dispersed data. The obvious implementation (mount a tape and put all the data you want on it) is unwieldy for two reasons:

Role	Action
Database Administrator	Design the logical schema, and determine fragmentation, fragment grouping, and placement of fragment groups at sites. In cooperation with site adminstrators, determine pagespaces required to hold fragment groups at each site.
Site Administrator	At each site, create the pagespaces agreed on with the database administrator. Grant the privilege to place fragment groups in each pagespace to the database administrator.
Database Administrator	Create tables, fragments, and fragment groups, and explicitly place fragment groups in previously defined pagespaces. Grant ordinary users privileges to access the logical objects (tables) but not the underlying pagespaces.

Figure 5. Production Usage/Autonomous Resource Control Scenario

- To obtain a consistent snapshot of a table (for example) may require retrieving data from fragments at remote sites. Transmission of the large volumes of data required for backups over the network would swamp the network and would cause backups to take a long time (and hence be less likely to complete successfully).

- If recovery is required, then the backup tape may be at a different site from the recovery site. This not only implies the data transmission problems mentioned above, but has the operational problem of getting the correct tape mounted at the correct time at a remote site (which may be anywhere else in the world!).

Individual sites cannot be allowed to autonomously backup local data, because there is no way to ensure that backups taken independently at different sites contain a consistent snapshot of the system. This is a problem when data at one site may refer to or depend on data at another site (even in the absence of replication). Therefore, even if backup operations only put local data on tape, there needs to be a global synchronization of backup operations. Similarly, recover operations must be synchronized when there are inter-site data dependencies. Furthermore, recovery of replicated data must synchronize all copies of the data, not just the copy being recovered.

Adaplex implements distributed backups using the read-only transaction mechanism [17]. The backup operation is performed in two phases: in the global phase, a database administrator "declares" a backup and gives it a name; in the local phase, individual site administrators "attach" to the backup and make tapes containing local data which is consistent with data put on tape at other sites. Internally, Adaplex starts a distributed read-only transaction when the backup is declared. This transaction reserves a consistent snapshot of the database at all sites which will participate in the backup. When individual sites attach to the backup, data is retrieved using the read-only transaction and put on tape. Since the database snapshot is automatically reserved until transaction termination, there is no need for real-time synchronization of local backup operations.

The distributed recovery operation has a similar two phased structure. The recover operation is first declared, and then individual sites perform local recover operations. To guarantee mutual consistency, Adaplex will not make the logical object being recovered accessible until all local operations have successfully completed.

6. Conclusions and Areas For Future Work

Before distributed DBMS technology can be adopted, adequate solutions must be provided for the problems of administering a distributed DBMS. Existing solutions for single-site DBMS's do not address the complications introduced by distribution. Database administrators must deal with fragmentation and replication; there are operations that span sites and must produce consistent results; the hierarchy of administrative roles in a single-site DBMS breaks down in the distributed case. Site autonomy has been proposed as a guiding principal in distributed DBMS administration, but it is incompatible with replication transparency. We propose a generalization of site autonomy in which the database is the unit of autonomous administration. Autonomy is raised to the logical level; sites are treated as physical resources. This allows a range of configurations for a distributed DBMS. At one extreme, functional site autonomy can be achieved by imposing a one-to-one correspondence between sites and databases. At the other extreme, a system can be configured to behave like a logically centralized DBMS by storing all data in a single database which is present at all sites. Intermediate configurations are possible with varying usage modes and resource control modes. We describe intermediate storage structures and an authorization mechanism that provide flexibility in configuring a distributed DBMS.

Areas for future work include:

- *Distributed Resource Control* - this paper has "routine" DBMS administration. It is not clear how this techniques will work in "exceptional" cases. For example, suppose a disk at one site overflows when applying updates to a copy of a replicated data item. There are many possible actions the system could take: abort all update transactions until the condition is fixed, take the copy offline, make the logical data item unavailable, etc. It is not clear what tools are needed to diagnose such problems in a distributed system, nor is it clear who is responsible for recognizing and fixing such problems.

- *Distributed Security* - the problems of providing security in a distributed system are well known. In the absence of reliably secure networks and trusted security kernels, it is not clear to what extent sites in a distributed system can trust each other.

References

[1] J. M. Gross, P. E. Jackson, J. Joyce, and F. A. McGuire, "Distributed Database Design and Administration", *Distributed Databases: An Advanced Course*, Draffan and Poole, eds., Cambridge University Press, 1981.

[2] H. Walker, "Administering A Distributed Database Management System", *ACM-SIGMOD Record*, 12:3, 1982.

[3] S. Ceri and G. Pelagatti, "Distributed Databases: Principles and Systems", McGraw-Hill, 1984.

[4] B. Lindsay and P. Selinger, "Site Autonomy Issues in R*: A Distributed Database Management System", IBM Research Report RJ2927, IBM, San Jose, CA.

[5] A. Chan, et al, "Overview of An ADA Compatible Distributed Database Manager", *ACM-SIGMOD Proceedings*, San Jose, CA, 1983.

[6] B. Lindsay, "A Retrospective of R*: A Distributed Database Management System", *Proceedings of the IEEE 75*, 5, 1987.

[7] O. Bray, "IRM In A Decentralized/Distributed Environment", *Database Engineering*, 9:2, 1986.

[8] ANSI, "ANSI X3.135-1986 Database Language SQL", American National Standards Institute, 1986.

[9] IBM Corp., "IBM Database 2 Data Base Planning and Administration Guide", SC26-4077-3, IBM Corp, 1987.

[10] Oracle Corp, "Oracle Database Administrator's Guide", 3601-V5.0, Oracle Corp., Belmont, CA, 1986.

[11] P. Selinger and B. W. Wade, "An Authorization Mechanism For a Relational Database System", *ACM-TODS*, 1:3, 1976.

[12] IBM Corp., "IBM Database 2 System Planning and Administration Guide", SC26-4085-3, IBM Corp, 1987.

[13] B. G. Lindsay, "Object Naming and Catalog Management For A Distributed Database Manager", *Proceedings 2nd International Conference On Distributed Computing Systems*, Paris, 1981.

[14] D. Daniels, et al, "An Introduction To Distributed Query Compilation in R*", *Proceedings of The 2nd International Symposium On Distributed Databases*, Berlin, 1982.

[15] E. Bertino, L. M. Haas, and B. G. Lindsay, "View Management in Distributed Data Base Systems", IBM Research Report RJ3851, IBM Corp., San Jose, 1983.

[16] P. F. Wilms and B. G. Lindsay, "A Database Authorization Mechanism Supporting Individual and Group Authorization", IBM Research Report RJ3137 (38514), IBM Research, San Jose, CA, 1981.

[17] A. Chan and R. Gray, "Implementing Distributed Read-Only Transactions", *IEEE Transactions on Software Engineering*, SE-11:2, February, 1985.