Preliminary Design of ADMS±: A Workstation-Mainframe Integrated Architecture for Database Management Systems

Nick Roussopoulos
and
Hyunchul Kang

Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

This paper presents a new architecture that fully integrates local and global database management in a transparent for the user fashion. The architecture utilizes the workstation's local processing and uses the global mainframe for sharing and maintenance of consistency. Two access path distribution protocols distribute data and processing by localizing uncommon paths to their requesting workstations while avoiding repetition of globally shared paths in workstations. A new concurrency control protocol is used which has its foundation on the deferred update strategy, the concept of differential files, and a new lock for derived objects.

1. Introduction

Recent advances in hardware technology and the drop of the cost of powerful workstations dictates new architectures in which terminals are replaced by workstations which can be fully integrated with the mainframes. Such a workstation-mainframe integrated environment provide several advantages the most important of which is that (a) local processing on the workstation is independent of the mainframe's load and (b) local storage can be accessed even when the mainframe is unavailable. Although a good deal of work has been done in the area of workstation computing environments, [Yalamanchili et al 1984], [Schroeder et al 1985], [Adobe 1985],etc., no research has been reported on tightly coupled cooperating environments with data bindings and dynamic data download facilities between mainframes and workstations.

In this paper we define a new database system architecture environment based on a hybrid system ADMS± built around an Advanced Database Management System ADMS [Roussopoulos et al 1984], [Roussopoulos 1985].

It consists of two similar and cooperating DBMSs. The mainframe DBMS, called ADMS+, is a full DBMS which in addition to the ordinary management tasks, keeps track of data and access paths downloaded to workstations. The workstation DBMS, called ADMS–, maintains data downloaded from ADMS+ to answer queries on the workstation and needs no concurrency and security control subsystem because it operates in a single user mode. ADMS– can be thought of as an intelligent cache database access subsystem which capitalizes on the locality of data usage and its bindings. The communication in ADMS± is between the workstation and the mainframe. No communication exists between workstations which are typically turned off when not in use.

The ADMS± architecture is not a distributed one but rather an extended centralized architecture. The user accesses the database as if he were on a centralized system using his workstation as a terminal. The workstation-mainframe interactions are hidden from him. As the user interacts with ADMS±, database access paths along with data associated with them are dynamically downloaded using an access path distribution protocol. The downloaded access paths are then **incrementally** maintained locally at the workstation using differential files between the ADMS+ and ADMS–. The distribution protocols localize the user's subset of the database on his own workstation providing speedier access to it. To offset a total localization, local access paths are uploaded and become global in the mainframe when they are found to be shared by a number of workstations.

The ADMS± architecture does not prevent additional database distribution among several ADMS+s. The rationale behind such an architecture is that, in the foreseeable future, there will always be a need to handle differently local from global data. Furthermore, at this time, and it may be the case for a while, the difference in speed and capacity between a mainframe and a workstation is significant. When tomorrow's workstations achieve the capabilities of today's mainframes, tomorrow's mainframes would be supercomputers that explore parallelism, and other special hardware storage machines.

Section 2 of this paper outlines the basic ADMS system features which provide the foundations for ADMS$\pm$. Section 3 describes the hybrid architecture of ADMS$\pm$, the access path distribution protocols, and the concurrency control protocols. Conclusions are in section 4.

2. Basic ADMS Features

ADMS has two distinctive features not found in other DBMS's: a cache technique for rapid access to views and a deferred update strategy. These are instrumental in providing very efficient "incremental computation on demand".

2.1. Access Path Model

ADMS supports the access path model defined in [Roussopoulos 1982a,b]. According to that model, an access path is defined by a query graph whose nodes are base relations and/or views, and whose edges are relational operators, such as select, join, union, intersection, etc. Each node in an access path is either a preexisting base relation or a view that is referenced by the query or an intermediate result needed to generate the target relation. The lowest node on the query graph is the target relation. The collection of all access paths executed against the database are integrated to form a Logical Access Path Schema.

2.2. View Cache

The view cache idea is to maintain for each view a pointer array consisting of pointers to those tuples of the base relations and/or other views needed to construct the view. We refer to these pointer arrays as view indexes. These indexes can then be cached to materialize the view at a much lower cost than either reexecution of the view definition or query modification [Stonebraker 1975]. View indexes are hierarchically structured, therefore, materialization of a view is a matter of following one or more levels of indirection to fetch the necessary tuples from the underlying base relations and mapping the view attributes into their corresponding base relation attributes. Since view indexes are maintained sorted, minimal buffering is required to avoid reading more disk blocks than it is absolutely necessary (optimal caching).

2.3. Incremental View Update

Updates to base relations, (i.e. insertions, deletions and modifications), outdate view indexes. Therefore, a subsequent request against an indexed view may have to be preceded by an **incremental update** of the view index to reflect the changes made to the base relations before caching. We refer to the **Incremental Update Cost** by IUC, and to the **Caching the Index Cost** by CIC.

ADMS supports VIEWCACHE, [Roussopoulos 85], a system of **very carefully** designed algorithms for incremental updating and optimal caching of view indexes. The principle idea behind VIEWCACHE is that, between any two consecutive accesses of the same view, only a small part of the index gets outdated and, therefore, the access path search is only necessary on the **increments** rather than the whole thing. The unaffected part of the index needs no access path search but, instead, it is directly cached. VIEWCACHE achieves extremely fast access to views. We have shown that IUC plus CIC is much less than the cost of reexecuting the views.

An additional revolutionary feature of VIEWCACHE is the interleaving of update and cache. This feature saves a lot of the CIC because whatever needs to be updated is read in, updated, and displayed immediately. Therefore, its corresponding CIC is saved, subsumed by the IUC. This interleaving and the special data structures used, (variations of B-trees and R-trees which maintain the view indexes sorted), make VIEWCACHE optimal with minimal buffering. Analytical and experimental results of VIEWCACHE are reported in [Roussopoulos & Kang 1985].

2.4. Deferred Updates

The second novel feature of ADMS is the deferred (or lazy) update strategy. Maintenance of views and other derived objects (such as secondary indexes) is deferred until a direct or indirect request to them is made. This avoids global overhead associated with updates to the database. When a query needs to access a possibly outdated view, say V, the VIEWCACHE incremental update algorithms propagate the changes of the base relations down to all intermediate views used in the derivation of V. These update algorithms use a set of differential files stored in the form of backlogs, and apply the performed updates against the view indexes. The update cost of V is part of the cost of accessing V. And since, the incremental update plus the cache costs are less than the cost of reexecuting the definition of the view, the cost of deferred update cost is zero; maintenance of views is totally free of overhead.

The same technique is used with secondary indexes. When a query needs to use an outdated index, it updates it first in a much more efficient way by batching, sorting and merging all the updates together. The cost of maintaining a given index becomes now the cost of using it in answering the query instead of being uniformly distributed as overhead.

The deferred update strategy allows the definition of a new class of concurrency control protocols for redundantly derived or replicated data, see next section. Views, secondary indexes, multiple copies, and other access aids that are derivable from base relations are referred to as **derived objects.**

2.5. Slow down for speeding up- A fully concurrent protocol

ADMS uses a concurrency control protocol based on a new type of lock, called **derived object lock**, or **d-lock** for short for concurrent retrieval and update of access paths and other derived objects. Because of the deferred update strategy, a query involving retrieval of derived objects on a given access path, is not necessarily a pure retrieval process with regards to this access path, but it may first invoke the access path update algorithm to bring any outdated subpaths up-to-date and then retrieve from them. d-locks are used for both of these processes.

To retrieve/update a derived object on an access path, a d-lock for it is acquired. A d-lock is preceded by a shared lock to all base relations deriving the object. This implies that while a derived object of an access path is brought up-to-date, concurrent access to the deriving base relations is allowed but not updates. During the retrieve/update of an access path, only dependent paths are not allowed to retrieve/update concurrently. However, as the retrieve/update of an access path proceeds, previously dependent subpaths are released from their d-locks as they become independent and concurrent retrieval/updating through them is permitted.

For example, while access path B in Figure 2.1 updates object V2, path A can simultaneously update object V1 but path C, that is dependent on V2 of B, has to wait. As soon as V2 gets updated, its d-lock is released and path C can proceed immediately and in parallel with path B which continues retrieving/updating.

What is even more important to point out is that this protocol is a fully concurrent non-positive delay protocol. This is true because even when a process requesting to update a path, say path C in the previous example, waits for another process to finish updating a dependent path, say path B, the waiting process C will not have to update the common subpath, V2 and above. The cost of updating shared paths is paid only once. The earlier process B had d-locked the common subpath, V2 and above in the example, the better for process C because it only has to wait for the remaining time of B which is a fraction of the time it would take C to do it itself. Thus, the more a process is slowed down by finding more and more d-locked objects, the speedier its execution! Note that, since almost zero time is needed to check whether an object is up-to-date, the slow down for speeding up concept holds for any number of waiting processes.

A simpler but less concurrent protocol was implemented in ADMS. Instead of d-locking derived objects, only base relations deriving a derived object are d-locked and remain d-locked until the update of the path finishes. This protocol in the example of Figure 2.1 would make path C wait until path B finishes updating (this is not the case in retrievals). At that point, C starts updating but, again, it needs not repeat updating the shared subpath V2 and below. This simplification in the implementation was adopted because it drastically reduces the number of object locks and overhead.

3. ADMS± Architecture

The principle design goal of this architecture is to distribute (download) local database access and centralize (upload) global access. As the user on a workstation accesses the database through ad hoc or precompiled queries, a local portion of the database is dynamically built and maintained on the workstation that a) provides very quick access to it because most of the local access needs not be scheduled with other access requests, and b) alleviates the mainframe load as it is performed on the workstation.

A basic assumption of this architecture is that, unlike network communication lines, the lines between the mainframe and each of the workstations have high bandwidth with no delays, similar to those connecting terminals to a mainframe. The cost of downloading data to the workstation is only slightly higher than dumping the data on the terminal. The only additional cost is that of locally storing the downloaded result but this is only a one time cost. Subsequent requests are capitalizing the cache benefits of the downloaded portion of the database.

In order to provide a simple and efficient technique that guarantees database consistency, updates on replicated (downloaded) base relations[1] are done on the mainframe first, using a standard update protocol, [Eswaran et al 1976], based on exclusive and shared locks. All committed updates are then recorded on the objects' backlogs.

Workstations that issue an access request to an updated base relation or to a derived data object dependent on updated base relations, including the workstation that issued some of these updates, pass along with the request a pointer to the backlog entry beyond which the updates have not been reflected on the downloaded object. If this pointer points to the end of the backlog, the object is up-to-date and can be immediately accessed locally. If more updates have been made to the object since the workstation's prior request, the difference of the backlog between the previous request and the current is transmitted (differential file and dirty pages only see section 3.4). The workstation then resets the pointer to the end of the backlog. No broadcasting of any sort to any workstations that have data objects affected by the updates is done. Updating of derived objects on the workstation is deferred until these objects are accessed

[1] We consider direct updates on base relations only. Updates to views are first translated into valid base relations updates using techniques presented in [Furtado & Sevcik 1977], [Dayal & Bernstein 1978], [Bancilhon & Spyratos 1981], and then are reflected to the views by the VIEWCACHE update algorithms.

again.

The main advantages of the ADMS± architecture are briefly outlined below:

a) The response time of queries that are locally processed is dramatically decreased because the workstation runs in a single-user mode and no dynamic security checking is necessary for the downloaded portion of the database. Security is checked only on the mainframe once before downloading. Only authentication of the user at login time is done on the workstation. Thus, ADMS– does not need either a concurrency or a security control subsystem.

b) Although ADMS± provides an extended centralized database environment, it distributes data and processing to workstations and achieves this in a simple and powerful manner that avoids the difficult problems of concurrency and data consistency control of fully distributed environments [Traiger et al 1979], [Stonebraker 1979]. d-locks and pointers to the backlogs easily handle concurrent access of derived objects on the mainframe, the only one that needs concurrency control. This distribution not only alleviates the mainframe's load but it increases the overall concurrency because locks in the mainframe are released much earlier than in a fully distributed environment.

c) Data distribution is dynamically done based on the workstations' requests. The user interacts with ADMS± as if he were using a centralized system. Initially there is no data stored in the workstations but downloaded on demand and maintained on the workstation as the system processes workstations' requests. This dynamic data allocation permits each workstation to build and maintain the portion of the database that is pertinent to its applications. Removal of once downloaded but not frequently accessed objects can be based on (a function of) the ratio of the sizes of the differential file needed to be propagated in order to update the downloaded objects over the sizes of the objects themselves. Clearly, for ratios close to one (or higher) the outdated downloaded portion is of little value. Other criteria, such as storage constraints, can also be used for removal of workstation objects.

d) The deferred update strategy with no broadcasting drastically reduces the overhead from message traffic to synchronize the updates. In an ordinary distributed DBMS architectures, the message traffic to obtain the appropriate locks would be four times the number of sites, [Date 1983]. Therefore, even for a moderate number of workstation, the message traffic would be very high.

e) ADMS± architecture is very modular and extendible. Very little preparation is necessary for adding new workstations.

f) The user can speed-up the query response by accessing "almost up-to-date" data instead of "up-to-moment" data. In this case, his queries are processed mostly locally on the workstation without having to wait for the current updates. This is not uncommon in many applications where a checked out portion of the database can go a long way before it needs be refreshed. For example, queries for browsing in a database for statistical gathering, or for searching archived files not affected by the current activities, or for developing and testing new queries, etc. Current DBMSs do not support almost up-to-date retrieval, but, instead uniformly distribute concurrency control overhead to all queries.

In the next subsections we describe the separation of global and local access paths, the global and local data model, the access path distribution protocols, and the concurrency protocol of ADMS±.

3.1. Global and Local Access Path Separation

The extension to the access path model required by ADMS± deals with data downloads, uploads, and bindings between mainframe and workstation objects. For example, an operator may be performed on the mainframe but the target relation TR is downloaded to the workstation. Figure 3.1 shows the extended set of unary and binary relational operators (unary in single lines and binary in double lines) and the implicit download ADMS± operators that build up the access paths in ADMS±. The horizontal line is used to separate the subpaths that correspond to mainframe and workstation respectively. M(R) and M(V) correspond to base relations and views of the mainframe materialized on a workstation (see section 3.3).

The access path defined by a query on a workstation may refer to global and/or local base relations and/or views. This access path can be separated into a global subpath that accesses data objects in the mainframe and the local subpath that accesses local objects. An example of a global/local access path is shown in Figure 3.2. Execution of the two subpaths is processed in the following order: the global first on the mainframe, followed by a release of all locks acquired on the mainframe to process the global subpath, followed by an uninterrupted and independent execution of the local subpath on the workstation.

ADMS± requires different catalog management in the mainframe and the workstation. The ADMS+ catalog keeps track of not only the global objects but also all local objects derived from global ones. The ADMS– catalog keeps track of all global and local to the workstation data objects, but has no knowledge of objects local to the other workstations.

3.2. Global-Local Data Model and their Bindings

Base relations and views of the mainframe are downloaded to a workstation when they are accessed for local processing. Similarly, local to a workstation views are uploaded when they are found to be shared by several workstations. Downloading and uploading is done differently for each object but in all cases the result is a binding between a mainframe object and a local one. The binding is similar to that of a derived object in ADMS.

We extend the class of derived objects to include downloaded base relations, downloaded materialized views, downloaded backlogs, downloaded secondary indexes, etc. When an upload of an local object O takes place, the mainframe object becomes the deriving and O becomes the derived one. This means that every data object on a workstation, with the exception of local only base relations, is either derivable from mainframe objects or it is derivable from objects that are themselves derivable from mainframe objects.

The storage structures used for the downloaded base relations and its associated secondary indexes are identical to those used in the mainframe. In figure 3.2, a downloaded relation R1 is represented by M(R1). Mainframe views, on the other hand, which consists of pointers, are materialized first and their materialization is downloaded. This is necessary to make access to the view local to the workstation. After the download of a mainframe view, the view becomes like a base relation but the tuple correspondence between the view in the mainframe and its downloaded materialization of it is maintained by ADMS– for efficient update processing. Figure 3.2 shows two materialized views, M(V1) and M(V2). M(V1) is bound to the mainframe view V1; M(V2) does not correspond to a global view, but it is the materialization of the result of a join between R2 and R3.

When a local to a workstation view is uploaded, the view that corresponds to the local data object is created on the mainframe. This view is then linked to the objects used to derive the local object in the first place. For example, if MV2 in figure 3.2 is uploaded, a new view V2 is created on the mainframe and MV2 is then bound to it. This is shown in figure 3.3. It is possible that by uploading a local view, more local views must also be uploaded to make the derivation of the first object possible on the mainframe. Consider the case of uploading V4 in figure 3.2; it results in uploading V3 as well and the final path is shown in figure 3.3.

A download or upload operation defines a binding between one or more global mainframe objects to one or more local objects on workstations. The bindings are points of transfer of control and processing from global to local. Even when retrieval of objects on a given access path is totally local, update control always has a global and a local component. It starts off by requesting all updates that affected their deriving objects on the mainframe and translates them into corresponding local updates. After this step, update control is passed to the workstation which processes the rest of the path independently.

3.3. Access Path Distribution Protocols

We describe two access path distribution protocols for downloading and uploading data paths and objects. Their purpose is to find an efficient "execution home" for a given query access path. The execution home may be either all global on the mainframe, or all local on the workstation, or both. The two protocols differ in the binding they allow between data objects of the mainframe and the workstation.

The Access Path Distribution (APD) protocols are designed with two basic principles in mind: first, to localize to each workstation the data and access paths (along with their associated computation for maintenance and access) that are very specialized to the applications of the workstation, but, are not general enough to be supported at the global level for the community of users. Specialized data and access paths, when downloaded to the appropriate workstations, produce no overhead to the mainframe and to other workstations. The second principle is to make global all those access paths that are common to good number of workstations. A common access path is uploaded to the mainframe if it is shared by k workstations, where k is set up by a policy maker. Making a path global increases the chance of finding it updated since it is accessed by several users. If updating is required, the cost paid in response to a requesting workstation can be "depreciated" against subsequent requests.

The access path distribution protocols are static and do not attempt dynamic query optimization. We consider this as a separate issue.

We assume that if a base relation is to be shared, then it is global because otherwise it cannot be derived by any workstation not having it. However, totally private relations can also be defined on the workstations. Private base relations cannot be seen by either the mainframe or any other workstation. In what follows, we are only discussing shared relations and their management rather than private ones. Private local relations can be easily handled by ADMS–.

To describe the protocols, we need the following definitions. Let

GS	be the set of relations and views residing on the mainframe,
LS	be the set of relations and views residing on a workstation,
DS(V)	be the set of deriving relation(s) and/or view(s) of view V in a given query. If V is derived from a unary operator, $\|DS(V)\| = 1$. If V is from a binary operator, $\|DS(V)\| = 2$. For a base relation $DS(R) = \emptyset$.
M(R)	be the materialized form of relation R on a workstation. If R is a mainframe base relation, M(R) is a copy of R. If R is a view on the mainframe, M(R) has the same structure as a base relation on the workstation.

The access path distribution protocols have three steps, the upload/download of the operand(s), the execution home, and the upload/retain of the result step. The upload/download of the operand(s) step determines which data objects need be uploaded/downloaded for the execution. The execution step decides whether the execution will be done at the mainframe or the workstation. Finally, the upload/retain step decides whether the result of the execution is uploaded to the mainframe and/or retained on the workstation.

The access path distribution protocols are best described in terms of requests provided in the query language. ADMS query requests can be classified into three types:

a. create a new base relation,

b. display an existing relation or view, and,

c. create an access path for a given query graph which consists of a set of existing or new intermediate relations/views and a target relation TR.

3.3.1. APD1- Non-Redundant Global & Local Access Subpaths

This protocol maintains the access paths in such a way that no overlap between global and local subpaths exists. In other words, in any access path there is unique binding between global and local objects and, thus, only one possible execution choice.

1. Creation of a new (shared) base relation R.
upload/download step:

Nothing is downloaded to the requesting workstation.

execution home step:

Execution is performed at the mainframe only.

upload/retain step:

GS := GS $\cup$ {R}, LS remains unchanged.

2. Display of a relation or view R.
upload/download step:

If R $\in$ GS, then set LS = LS - {R}. (This takes care of the case of R having been promoted to global since the last time it was accessed by the workstation.)

execution home step:

case (M(R) $\in$ LS):
Execution is performed locally on the workstation.

case (M(R) $\neg\in$ LS):
Execution is performed on the mainframe.

upload/retain step:

None. GS and LS remain unchanged.

3. Create an access path with TR as target relation.

The following is repeatedly applied to every subpath <D=DS(TR),TR> consisting of a single operator from the given access path. Since this is a repeated process, the resulting GS and LS sets are compared to the initial GS_0 and LS_0 before any actual download/upload/execution occurs.
upload/download step:

case (D $\subseteq$ GS):
Set LS = LS - D. This takes care of the local objects having been promoted since they were last accessed by the workstation and intermediate results in a long path. Note that some of the elements in LS are removed in the upload and reinserted in the download part of this step.

case (|D|=2 & **D** $\cap$ LS $\neq \emptyset$):
The non-local deriving relation in **D** is downloaded[2].

execution home step:

case (**D** $\subseteq$ GS):
Execution is performed on the mainframe.

case (**D** $\subseteq$ LS):
Execution is performed on the workstation.

case (|D|=2 & **D** $\cap$ LS $\neq \emptyset$):
Execution is performed on the workstation.

upload/retain step:

Retain TR at the workstation. If TR $\in$ GS, upload TR to the mainframe.

Using APD1, every access path in the mainframe is an access path which is shared by at least k workstations, and no global subaccess path is redundantly maintained in a workstation.

3.3.2. APD2- Redundant Global & Local Access Subpaths

This protocol is very similar to APD1 except for the upload/download of the operand step in the creation of a new access path. When a deriving object Ri $\in$ **D**, that is not an intermediate view but a target relation of a query, is found global, it is not removed from LS and this creates multiple bindings among global and workstation objects on the same logical access path. This implies that the switch from global to local execution can be done in more than one way. The lower the switch the more mainframe execution. Figure 3.4 shows such a path that has redundant global and local subpaths.

Using APD2, every access path in the mainframe is an access path which is shared by at least k workstations, but, workstations may maintain global subpaths in addition to their local paths.

Redundancy in global and local access subpaths allow the implementation of a dynamic execution strategy. The redundant subpath may be executed on the workstation or the mainframe depending on mainframe's load, speed, etc. In contrast, ADP1 provides a simpler but fixed execution distribution of the

[2] Note that the semijoin techniques discussed in [Ceri & Pelagatti 1984, chapter 6] for optimizing distributed joins can be used for reducing the amount of downloaded data.

subpaths.

3.4. ADMS± Deferred Update Strategy and Differential Files

The update strategy of ADMS± is similar to that of the basic ADMS, namely, all derived data objects, residing on the mainframe or a workstation are not updated until it is absolutely necessary to answer a query involving the object. This has two basic advantages: first, the system is not burden by the overhead of broadcasting every single change to every workstation that has data objects affected by the change. This keeps the communication control overhead of the system low. Second, when a derived object that has been affected by a series of updates needs to be brought up-to-date, the updates of the underlying objects can be propagated and processed in a more efficient way (optimized in a batch mode).

There are two basic techniques that are utilized for processing derived objects. The first deals with copies of objects; i.e. base relations and views stored globally and locally. When one of the copies is up-to-date and another copy needs to be updated, the dirty disk pages are transmitted. This allows replacement of the pages containing all the tuple updates. This technique requires precise maintenance of the one-to-one correspondence between the records of the data structures used.

The second technique is the viewcache technique of ADMS that is based on processing the differential files of the backlog storing updates (see section 2). Every base relation and view on the mainframe and every derived object in any workstation has a backlog. The recorded updates allow the update of all dependent access paths. This technique does not need the precision of the one-to-one correspondence and this makes it easier to implement especially on different DBMS±.

Global data objects are updated on ADMS+ using the basic ADMS algorithms. The update of the local objects involves the cooperation of both the mainframe and the requesting workstation.

Downloaded derived objects keep a pointer to the last entry (or entries) of the backlog (or backlogs) of the data objects they are derived from. When a workstation requests access to local object that is bound to some global objects, it passes along to the mainframe those pointers. The mainframe checks all the objects that the requested object depends on. If no updates have occurred since the workstation's last access, it returns a null set and access to the local structures is immediate. If updates have occurred, then the mainframe transmits only the differentials between the previous state of the deriving objects' backlogs and their current state (always maintained at the mainframe). Since the update process of derived objects considers only differences, the amount of data transmission is very small. After the transmission of the differential files, the update process is done locally at the workstation. The mainframe's only participation in the update of a downloaded object is on transferring (propagating) the differential files.

Redundant access paths, namely those created by APD2, can be either updated on the workstation or the mainframe. From there on, update requests to those can use the first technique with the dirty pages to be transmitted to the other copies of the access paths.

3.5. ADMS± Concurrency Control

In ADMS±, the access path defined by a query on a workstation consists of a global subpath on the mainframe and a local subpath on the workstation. Concurrency control is needed only on accessing the global subpath. The same concurrency protocols based on d-locks for derived objects and the standard protocol for the update of base relations on ADMS+ are used.

When the processing of the global subpath is finished, all locks are released immediately before the processing of the local subpath begins. No concurrency control is needed on the workstation. This early release of locks significantly increases the mainframe concurrency compared to the concurrency achieved when the whole access path resides on the mainframe. Furthermore, since local access paths on a workstation are mostly independent of local access paths on other workstations, and, since no overhead due to synchronization delays is incurred, overall concurrency is also much higher than that of a distributed concurrency protocol.

3.6. Global and Local Control Parameter

The parameter k is used to control globalization. When k is one, everything becomes global and this achieves maximum access path sharing at the cost of increased mainframe load and reduced concurrency. When k gets bigger and bigger, access paths become less and less global, and workstations have to do more and more computation on their own. This increases concurrency and distributes the load evenly.

A very interesting distribution is achieved by giving k its other extreme value, that is k becomes bigger than the number of workstations. In this case, only base relations are maintained on the mainframe and all access paths are locally maintained. This may be a reasonable approach on less capable mainframes which due to load, speed, storage, etc., become the bottleneck of the system. But more importantly, no other overhead is incurred but the absolutely minimum required to maintain the database consistent in a multi-user environment. All other processing is independent.

4. Conclusions

We have presented a new architecture that fully integrates local and global database management in a transparent for the user fashion. The architecture utilizes the workstation's local processing and uses the global mainframe for sharing and maintenance of consistency. The access path distribution protocols described distribute the overhead by localizing uncommon paths to their requesting workstations while avoiding repetition of globally shared paths in workstations. The concurrency control protocol used has its foundation on the deferred update strategy, the concept of differential files, and the new lock for derived objects.

We have started the implementation of ADMS± using a VAX 8600 as a mainframe and a series of VAX 750s and SUN IIs as workstations. Since all the ADMS family software runs on any of the above machines under Berkeley UNIX 4.3, the roles of the mainframe and workstations can be interchanged.

ADMS± can be further enhanced to provide a fully distributed environment among several mainframes. We believe that the concepts presented in this paper are easily extendible to replicated data and that they bring a fresh look to the difficult problem of managing and controlling distributed databases.

5. References

[Adobe 1985]
"Adobe Manual: XDE Project Management Tools User Guide" Xerox Corporation, September 1985.

[Bancilhon & Spiratos 1981]
Bancilhon, F.M., and N. Spiratos "Update Semantics of Relational Views" ACM Trans. Database Systems, Dec. 1981.

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

[Date 1983]
Date, C.J., "An Introduction to Database Systems," Volume II, Addison-Wesley, 1983.

[Dayal & Bernstein 1978]
Dayal, U., and P. Bernstein, "On Updatability of Relational Views," Proceedings 4th International Conference on Very Large Data Bases Berlin, pp. 368-377, 1978.

[Eswaran et al 1976]
Eswaran, K.P., J.N. Gray, R.A. Lorie and I.L. Traiger "The Notions of Consistency and Predicate Locks in a Database System," Commun. ACM 19,11 Nov. 1976 pp. 624-633

[Furtado and Sevcik 1977]
Furtado, A.L. and Sevcik, K.C., "Permitting Updates Through Views of Data Bases," Departamento de Informatica, Pontificia Universidade Catolica, Rio de Janeiro, Brazil, 1977.

[Roussopoulos 1982a]
Roussopoulos, N. "View Indexing in Relational Databases," *ACM Trans. on Database Systems,* Vol. 7, No. 2, pp. 258-290, June 1982.

[Roussopoulos 1982b]
Roussopoulos, N., "The Logical Access Path Schema of a Database," *IEEE Trans. on Software Engineering,* Vol. SE-9, No. 5, September 1982.

[Roussopoulos et al 1984]
Roussopoulos, N., Bader, C., and O'Connor, J., "ADMS: An Advanced Database Management System: Design Document" Department of Computer Science, University of Maryland, College Park, January 1984.

[Roussopoulos 85]
Roussopoulos, N. "VIEWCACHE: A System to Support Rapid Access to Database Views," Department of Computer Science, University of Maryland, College Park, September 1985.

[Roussopoulos & Kang 1985]
Roussopoulos, N., and H. Kang, "ADMS View System: Algorithms and Cost Analysis," Department of Computer Science, University of Maryland, December 1985.

[Schroeder et al 1985]
M.D. Schroeder, D.K. Gifford, and R.M. Needham "A Caching File System For a Programmer's Workstation" ACM SIGOPS Operating Sys. Review 19,5 Dec. 1985

[Stonebraker 1975]
Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification" ACM SIGMOD International Conference on Management of Data, pp. 65-78, 1975.

[Stonebraker 1979]
Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data of Distributed INGRES" *IEEE Trans. Software Engineering,* May 1979.

[Traiger et al 1979]
Traiger, I.L., J.N.Gray, C.A. Galtieri and B.G. Lindsay "Transactions and Consistency in Distributed Database Systems" IBM Research Report RJ2555 June 1979.

[Yalamanchili, et al]
Yalamanchili, S., M. Malek, and J.K. Aggarwal "Workstations in a Local Area Network Environment" Computer vol.17 #11 Nov. 84, pp. 74-86.

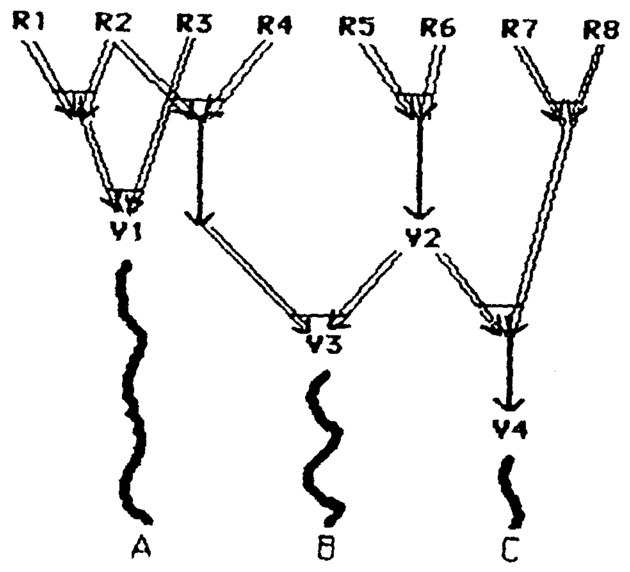
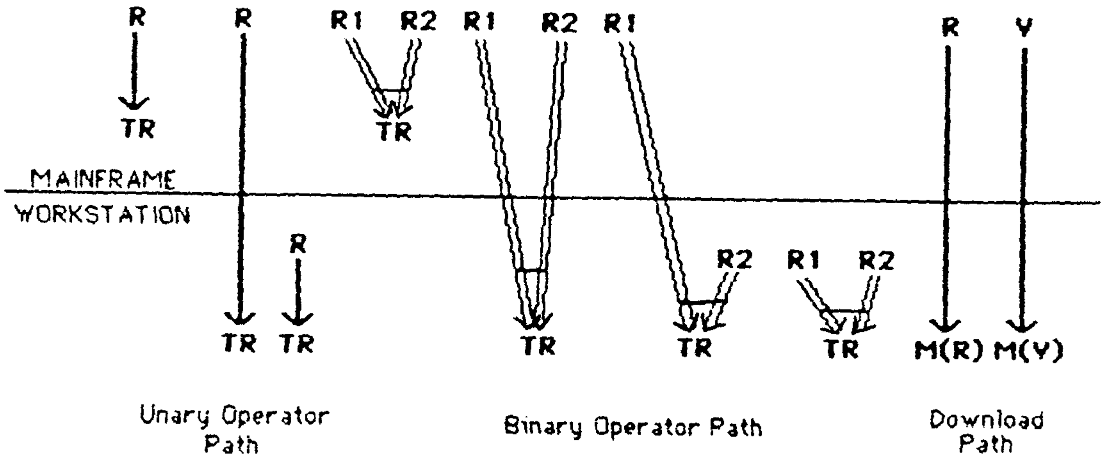


Figure 2.1



Bindings generated by the relational operators and downloads of relations and views

Figure 3.1

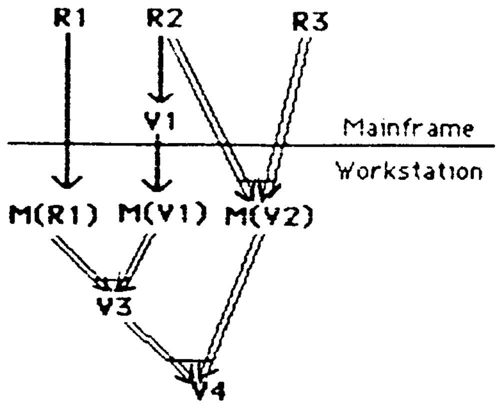


Figure 3.2

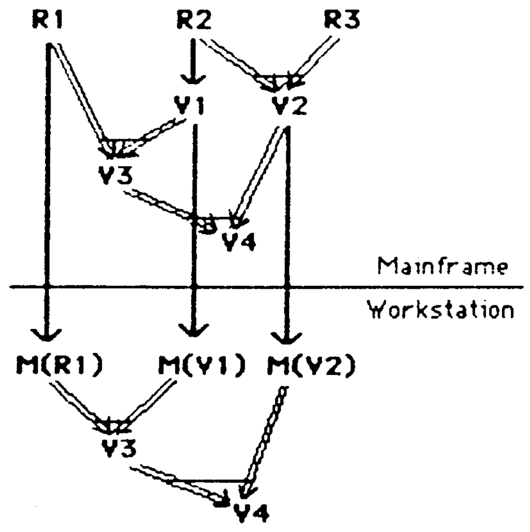
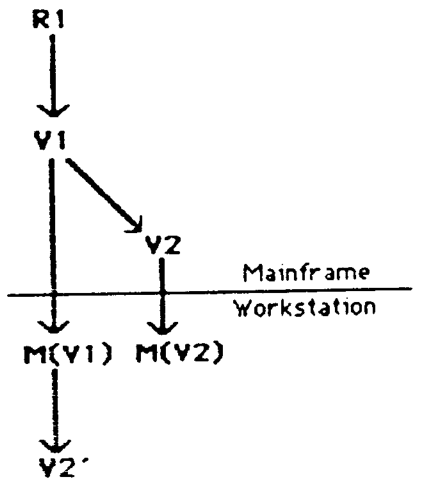


Figure 3.3



Subpaths (V1,V2) on the mainframe and (M(V1),V2') on the workstation are redundant.

Figure 3.4