NEGOTIATING DATA ACCESS IN FEDERATED DATABASE SYSTEMS

Rafael Alonso
Daniel Barbara

CS-TR-160-88

June 1988

(Revised October 1988)

NEGOTIATING DATA ACCESS IN FEDERATED DATABASE SYSTEMS [†]

Rafael Alonso
Daniel Barbará

Department of Computer Science
Princeton University

Abstract

The ever growing need for information is putting pressure on organizations to share data with their partners. However, although the different entities would like to share information, it is clear that each individual system administrator would like to preserve his or her control over the system. This concern with each system's autonomy has led to the notion of **federated databases**. Previous work in this area has touched upon the topic of negotiating access in a federated database (i.e., determining what local information may be accessed by any particular remote user), but we feel that the topic has not been studied in depth. In this paper we propose a new scheme, based on the notion of **quasi-copies**, which may be used for interaction among autonomous databases. We also provide protocols for access negotiation, as well as simple cost models for estimating the expense involved in allowing remote access to information. One of the main advantages of our scheme is that it provides a very precise way of establishing how much autonomy is given up by the owner of the information when he or she decides to share data. Finally, our approach may be used in both the case where the databases systems in question have a common query language (or there exist facilities for query translation), and when they do not.

[†] This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and by the Office of Naval Research under Contracts Nos. N00014-85-C-0456 and N00014-85-K-0465, by the National Science Foundation under Cooperative Agreement No. DCR-8420948, and New Jersey Governor's Commission Award No. 85-990660-6, and grants from IBM and SRI's Sarnoff Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

1. Introduction

The growing need for information is putting pressure on organizations to share data with their partners. Also, individuals are ever more eager to obtain information about a wide range of topics. Some institutions are already providing such information for a fee. (Dow Jones [D84] and the Source [ED83] are two examples.) The spread of networking technology will make possible in the near future for a person to have access to a huge network of ''information stations''.

In terms of technology, this concept presents a challenge to the way database management architectures are conventionally defined. In an environment such as the one described above, databases tend to proliferate across the network, without falling under the control of any single authority and with very little standardization among them. In this situation, individual systems will want to preserve their autonomy while still being capable of sharing data efficiently with others in the network. An architecture that supports data sharing among different database systems while preserving the autonomy of the individual entities has been called a **federated database**. (See [HM85].)

In such systems, we will define for each piece of data two different classes of components. The **owner** of the data is the member (or group of members) of the federation that controls that data and is free to share it or not with the rest of the members of the federation. The **importers** are those other members of the federation that are interested in accessing the data.

The following conditions must be met for a system to qualify as a federated database.

- Any individual component of the federation has sufficient local control so that it may autonomously determine which local information it will share with the rest of the federation and in what capacity. The owner is also free to refuse access to any particular importer for any reason it deems important. For instance, a component of the federation might not want to share data with another component if that sharing implies the commitment of keeping the importer informed about updates to the data.
- Each component is free to determine which data is of interest to its users. Furthermore, each importer may negotiate with the owner of the data for whatever type of commitment that may be necessary for the intended applications of its users. For example, the importer may specify that it wishes to be kept appraised about future updates on a data item that it has just read. Or perhaps a site may stipulate that not only does it require access to the information, but that queries about the data must be completed with a certain maximum response time.

In the past, the notion of data sharing has been synonymous with letting importers query the local database. We feel, however, that there is a broad spectrum of choices in this area. For example, for performance reasons, one may consider giving the importer a copy of the data. If the importer has very stringent requirements for response time, a local copy might be the answer to efficient sharing. Placing a copy in the importer's site has the added benefit that it also provides a way of lowering the query traffic to the owner site, thus reducing the competition for the owner's resources.

Heterogeneity is a second important reason for using copies. If the systems do not share a common query language, and no copies are kept at the importer's site, all importer queries will have to be made using the owner's query language (which implies the overhead of a query translation mechanism). Of course, once data sharing has been agreed upon, the importer can always keep a copy of the data items without notifying the owner (say, by storing the result of its previous queries). But in reality this is not a feasible solution in many applications since in this case the importer can never be sure of the consistency of the data.

However, when the owner grants a copy of the data, the responsibility for data consistency is an issue that can (and must) be resolved beforehand. The choices range from giving a copy without any commitment (i.e., providing the importer with a **hint** about the data) and leaving the responsibility of the consistency to the importer, to keeping the importer posted of every update. It is clear that the choice has a tremendous impact over the amount of autonomy that the owner has over his or her data (where by autonomy we mean the degree of freedom that an owner has to update local information). For instance, in the second case above, the owner cannot proceed with an update without informing all the importers first (possibly requiring the expense of two-phase commit or some other type of transactional protocol).

As can be appreciated from the comments above, the interests of the owner and the importer of data may often conflict. The needs of both types of entities must be taken into account, and an agreement (if possible) should be reached. The process of reaching an agreement is called a **negotiation**, and is a key element to the federated system. If such negotiation is to take place automatically, the need for a protocol is evident. Such a protocol should specify the following two items:

- A way to precisely express the needs of the importers and the degree of sharing that the owner is willing to offer.
- A way to estimate the cost of a specific agreement, both for the owner and for the importer of the data, as a means for justifying such an agreement.

In this paper, we address the basic aspects of a negotiation protocol for sharing data in a federated database system. This protocol is based in the concept of **quasi-copies** (See [ABG88]). A quasi-copy is a cached value that is allowed to deviate from the central value in a controlled fashion. The degree of consistency of the quasi-copy is established by the entity where the quasi-copy is to reside. In the next section, we present some background information about quasi-copy management, and note the role of quasi-copies in data sharing. In Section 3, we explain the model of negotiation and the protocol. In Section 4, we develop techniques to estimate the cost of a proposed agreement. Finally, in Section 5 we offer our conclusions and point to some directions for future work.

2. Quasi-Copies

The notion of quasi-copies was originally developed for a centralized information retrieval system (IRS) [ABG88]. In those systems, a central node holds all the data and is responsible for all updates and management of data. The idea behind quasi-copies is to *cache* data on remote workstations to improve performance. The gains in performance come from two different sources. First, caching data may eliminate multiple requests for the same data, eliminating redundant requests to the central IRS. Secondly, it off-loads work to the remote sites.

However, although caching data is a very natural idea, it does not come for free. Every time a cached item is updated at the central node, the new value must be propagated to the copies. Moreover, if we insist in complete consistency of the copy, the propagation must be done **immediately**.

On the other hand, caching can be made less costly if a weaker type of consistency is allowed. A weaker form of consistency may be quite sufficient for some applications. For instance, if a user is interested in stock prices of oil companies, he or she may be satisfied if the prices that are given to him or her are within five percent of the real values at any time. The concept of quasi-copies is a generalization of this notion.

A quasi-copy is a cached value that is allowed to diverge from the central copy in a **controlled** way, which is determined by the actual user of the data. For instance, a user may specify that a copy should not diverge by more than 10% from the central value, or that the information be no more than one hour old. We feel that quasi-copies are a very natural way of dealing with data and are used already in many organizations in an implicit way. Consider, for example, that the CEO of a large company is not normally notified every time an employee quits, but a massive exodus of employees should probably be brought to her or his attention. Magazines and newspapers contain data that is in a way out-of-date in a controlled manner (i.e., the ''data'' is supposed to be no more than a day old).

Although quasi-copies were developed in another context, they are also very attractive for federated database management, and we have used them in our architecture as the basis of data sharing. As will be seen later, we propose that in federated databases the owner of the data can share information by giving quasi-copies to the importers, with the added commitment of honoring the consistency requirements that the importers and the owner agree upon. In the reminder of this section, we describe the relevant issues of quasi-copies. A more detailed description can be found in [ABG88].

The two principal characteristics of quasi-copies are the **selection conditions** and the **coherency conditions**. The first refers to the set of objects that the importer wants to request and the way the copy should be maintained. The second refers to the degree of consistency requested. The set of objects involved can be expressed in a relational language [D81]. For instance, a importer that desires to have a quasi-copy of the list of flights arriving in New York may request it by issuing the following command:

SELECT: *Flight_Number*
FROM: *FLIGHT_SCHEDULES*
WHERE: *Flight_Destination = New York*

The coherency conditions may adopt a variety of forms. Three useful types are:

(1) *Delay Condition.* It states how much time a copy may lag behind the true value. For and object x, its copy x', and allowable delay of α, the condition is given as:

$$\forall \text{times } t \geq 0 \; \exists k \text{ such that } 0 \leq k \leq \alpha \text{ and } x'(t) = x(t-k)$$

(2) *Version Condition.* Instead of using time, versions are used to specify the window of allowable values. We represent this condition as $V(x) = \beta$, where x is the object and β the maximum version difference. That is, $V(x) = \beta$ is the condition

$$\forall \text{times } t \geq 0 \; \exists k, t_0 \text{ such that } 0 \leq k \leq \beta \text{ and } 0 \leq t_0 \leq t \text{ and } v(x(t)) = v(x(t_0)) + k \text{ and } x'(t) = x(t_0)$$

(3) *Arithmetic condition.* If the value of the object is numeric, the deviations can be limited by the difference between the values of the object and the copy.

$$\forall \text{times } t \geq 0 \quad |x'(t) - x(t)| < \varepsilon$$

or that

$$\forall \text{times } t \geq 0$$

$$\left| \frac{x'(t) - x(t)}{x(t)} \, 100 \right| < \varepsilon \, \%$$

We represent the first condition by $A(x) = \varepsilon$ and the second one by $A(x) = \varepsilon\,\%$.

More conditions can be built out of the elementary ones, connecting them with logical **"OR"**, **"AND"**, and **"NOT"**. For example the condition

$$V(x) = 2 \; OR \; A(x) = 10\%$$

means that the copy x' can lag two versions behind x or differ 10% from the value of x.

A crucial point regarding the use of quasi-copies in a federated environment is that their management can be jointly carried out by the owner and the importers. The main reason for an owner to offer a quasi-copy to an importer is to avoid the constant traffic of queries to its facilities. We obviously do not want the load caused by the management of quasi-copies to offset the gains of not having the constant querying of the owner's database. Therefore, it is important to distribute the work effectively.

Clearly, checking the consistency conditions at the owner's site requires massive amounts of storage and a considerable amount of processing power. To honor a consistency condition, the owner must have a model of the state of the importer's copy. For instance, consider the condition $A(x) = 5$ and $x = 10$. If an update arrives that makes $x = 15$, the owner must know the value of x'. If $x' = 10$ the condition still holds; if $x' = 8$ it does not. If the database holds m items and there are n importers, the owner's site will have to store $O(mn)$ values, plus the associated conditions. Fortunately, there are ways to substantially decrease this overhead.

First, there are some coherency conditions that are easy to check at the importers nodes themselves, without bothering the owner's database. An example is a delay condition in which the value x' is valid only for a window of time. After that period has passed, the importer simply purges the value forcing the next request to be directed to the owner.

Secondly, the storage requirements at the owner's site can be reduced substantially by the following scheme. Each time an importer receives a new (current) value for an item x from the owner, it computes a pair of values x^i_{low} and x^i_{high} based on the importer's coherency conditions. These values represent the lowest and highest values that x could reach at the owner's site, without the need to send an update to the importer. The importer sends these values to the owner which in turn computes the *tolerance interval* for x at importer i as

$$\Delta^i_x = [x^i_{low}, x^i_{high}]$$

And now the owner need only store the global tolerance interval $\Delta_x = [x_{highest\_low}, x_{lowest\_high}]$, defined as the intersection of the individual tolerance intervals for importers of item x. Whenever an update of x sets the value outside Δ_x, the owner will broadcast it to **all** the importers holding a copy of x. The storage needs are reduced now to $O(m)$, at the cost of potentially higher message rates. However, if a broadcast mechanism is available, this may not result in much added overhead for the owner. Finally, it is easy to see that the technique just described may be applied to versions as well.

Thus, the only drawback of this approach is that the frequency of update propagation may be increased by taking the intersection of the intervals. This may happen if a particular user sets a very narrow interval or the importers start to cache an item at different times. Regrettably, there is no way of telling at the negotiation time whether a new importer may narrow the interval just by examining the coherency conditions. To see this, consider an owner that has granted a quasi-copy to an importer with condition $A(x) = 10\%$ and is negotiating a request with an importer that is asking for $A(x) = 10$. In a situation when $x = 10$ and it is then updated to $x = 11$ the new importer condition will have no effect and there will be no need to propagate the new value. However, if $x = 1000$ and it is updated to $x = 1100$, the second condition would force a broadcast. On the other hand, this problem may not arise in practice, because if all the importers of the information are running similar applications, it is not inconceivable that they will also share similar coherency requirements.

Finally, an important point in the implementation of quasi-copies is how to deal with failures and transmission delays. To illustrate, consider the condition $A(x) = \varepsilon$ and assume that an update of more than 2ε is about to occur. The condition indicates that the difference between the value in the central node and the quasi-copy must "never" exceed ε. However, this would imply that the

update to the image must be performed ''at the same time'' as the update in the owners site. Strictly speaking, this is not possible because of transmission delays. The owner of the data, when operational, will make sure that the importer receives a message every δ seconds. If T_D is the maximum message delay, this condition implies that if no value has been propagated to the importer in $\delta - T_D$ seconds, the owner must send a ''null'' message. Therefore, the semantics of a coherency condition $C(x)$ must be interpreted in the following way

$$C(x) \vee W(x) = \delta$$

This means that all conditions will have an implicit delay window of δ.

Failures can be treated in a similar way. In fact, when the importer notices that δ seconds have passed without a message from the owner, it declares a local variable $FAILED(x)$ as true. Then the condition can be interpreted as:

$$C(x) \vee W(x) = \delta \vee \sim FAILED(x)$$

The absence of a message after δ seconds can be attributed to one of two causes. Either the owner site has failed or the communication between owner and importer has been interrupted (i.e., a partition of the network has occurred). In the first case, obviously the value of the item will not change, since the updates to x will stop. Thus, the condition may remain in effect. In the second case, there are two alternatives.

a) The owner site can honor the contract with the importer by refraining itself from installing new updates that violate the condition $C(x)$. In many applications, however this alternative is not feasible and will cause the owner to surrender an intolerable amount of autonomy over its own data. For instance, the price of stocks cannot be stopped from fluctuating just because the customers cannot communicate with the source of the data. Or, in a system that controls a physical process, a measured parameter cannot be prevented from changing. (I.e., a temperature will change without regard to partitions in the network.)

b) The condition will be declared without effect for the duration of the partition. In this case, the variable $FAILED(x)$ will be declared as true. This will be the most commonly used alternative.

In practice, however it may be difficult to distinguish between a partition and the crash of the owner site. In this case, we must content ourselves with invalidating the condition for the interval of the failure.

In the next section, we make use of all these concepts in describing a precise protocol with which to carry out the negotiation for data sharing.

3. The Protocol

In this section we describe the actions involved in negotiating for access. Throughout, we refer to the site which owns of the information as the ''owner site'', and we will denote as ''importer site'' that machine which wishes to share the data.

The first step in our protocol is for the owner to publish its export schemas (i.e., the list of data items it chooses to make available). The decision of to which items access will be allowed is orthogonal to our discussion, but clearly will be related to the types of interactions anticipated by the owner. For example, a stockbroker company will be eager to publish information about stock prices, but will never want to publish internal information like its own payroll.

At some point in the future, a importer site may become interested in a piece of information. That site then tries to find an owner site that will provide the data. For the moment we will assume that this can be done by examining a federated dictionary. (In a truly large system, there will not exist a single federated dictionary containing pointers to all the information on the network. Rather, there will be a service analogous to a name server. Currently, another aspect of our research involves analyzing the structure of such a naming service.)

Once the importer site determines which owner site to contact, it will do so by sending a message containing an access request, as well as a few other parameters. (Those parameters will be used to determine the cost to the owner site of providing access to the data. This point is discussed in more detail in the next section.)

Once the owner site receives the request, it decides whether to allow access or not. It then notifies the importer site of its decision. If the owner decides to grant access in the form of a quasi-copy, it will then commit itself to maintain the coherency conditions associated with the quasi-copy. At this point the two sites are said to have ''negotiated a contract''.

The contract will be in effect until the owner decides to terminate it. (If the importer site ever wants to terminate the contract it can simply disregard all owner messages with quasi-copy updates. Notifying the owner is then only a matter of efficiency, not correctness.) For the present we will assume that the owner will always reserve the right to deny access to its information at some time in the future, although there may be applications where this is not desirable. However, if the owner decides to terminate the contract it must then notify the importers of this fact. If the communication links between owner and importers have failed, all contracts are assumed to be invalid until the failure is repaired. (Although, as we suggested in the previous section, another possibility is for the owner to refrain from any updates that violate coherency conditions.) All sites may be quickly appraised of network failures by making sure that periodic "I'm alive" messages are sent by all the sites involved.

Clearly, the only difficult aspect of the interaction we have just outlined is making the determination of whether to grant access or not. There are two possible situations to consider, and we will examine both of them in this paper. The first is the **heterogeneous** case, i.e., where we assume that the sites in question do not share a common database schema, and that sharing information may only take place by providing the importer site with a copy of the owner's data. (In other words, the importer site may not generate direct queries to the database of the owner site.) In that case the only decision for the owner to make is whether to grant a quasi-copy or not. We will consider this case in detail in Section 4.1 below.

The other possibility is that the two sites indeed share a database query language. We call this the homogeneous case. Now the owner node may consider the following possibilities. Either it grants access or not. If access is given, the owner must decide if it will let the importer site query the database directly or whether a quasi-copy will be provided. We consider this more complex case in Section 4.2.

4. Computing the Costs

A crucial aspect of the negotiation protocol involves the computation of the cost incurred by allowing access to importers. In this section we describe precisely how to compute those costs. In Section 4.1, we consider the heterogeneous case, and in Section 4.2 the homogeneous case.

4.1. The Heterogeneous case.

We begin this section by describing the information that must be provided by importers and owners of data in order to compute the cost of sharing. We will start by discussing the information required from the importer. In Table 4.1 we present a set of three parameters with which the importer defines precisely its needs for the data controlled by the i-th owner. Each parameter is subscripted with i, indicating that it is going to be used in the process of negotiation with the i-th owner.

Parameter	Symbol
querying rate	λ_i^q
coherency condition	$P(X)$
selection condition	S_i

Table 4.1. Importer Parameters

The querying rate is the importer's estimate of the number of queries per second that will involve the requested data. The coherency condition was explained in Section 2. The selection condition represents the importer's way of specifying the data it wishes to import from a particular owner i. This condition can be written as a query in an SQL-like language, as explained in Section 2. The entire set of data that any site desires to use from other databases is defined by the union of the S_i and corresponds to the definition of *import schema* in [HM85].

From the point of view of the owner, there are two types of information that must be specified in the negotiation process. The first is a list of the data items that the component is willing to share with the rest of the system. This is called the *export schema* in [HM85]. The second kind refers to the internal parameters that the owner uses to decide whether the negotiation will culminate in an agreement or not. These parameters are shown in Table 4.2.

Parameter	Symbol
update frequency	μ_x
number of quasi-copies of the item	k_x
average load	$\overline{W}$
maximum load	$W_{\max}$
number of bookkeeping instructions	I_b
number of instructions per broadcast	$I_s(k_x)$
number of instructions per query	I_q
probability of triggering a broadcast	$P_x(k_x)$
system capacity	μ

Table 4.2. Owner Parameters

The update frequency for item x is based on accumulated experience or on an estimation of the number of times the item x experiences an update during a given period of time. The number of quasi-copies of the item reflects the number of importers that have been already granted a quasi-copy of x. The average load in the system is related to the average amount of work that the users of the owner site are expected to generate. It is measured here as mean response time of user queries. The maximum load represents the highest delay that the owner is willing to tolerate. (The difference $W_{\max} - \overline{W}$ is the spare load that can be used to handle extra work due to importer requests.) The average number of instructions to perform quasi-copy bookkeeping represents the cost of checking the predicate conditions once an update has occurred. The average number of instructions to broadcast is defined as the number of instructions executed by the network protocols to broadcast a value. It is shown as a function of the number of granted quasi-copies to account for the case in which there is no broadcast facility in the system and extra quasi-copies mean extra messages to send. (If a broadcast facility is available the cost remains constant, i.e., $I_s(k_x) = I_s$.) The average number of instructions to query the local database is based on path-lengths on the owner's database. The probability that once the request is accepted an update of x will trigger a broadcast depends on whether there are other importers of the same data and how strict is the coherency condition requested by the importer with which the negotiation is being carried out. The capacity of the system is defined as the number of instructions per second the owner's CPU is able to deliver.

With these parameters, we can compute the amount of extra work that the owner site will encounter if the quasi-copy is granted.

$$\lambda_{extra}^{qc} = \mu_x [I_b + P_x(1) I_s(1)] \quad \text{if } k_x = 0$$

$$\lambda_{extra}^{qc} = \mu_x [P_x(k_x + 1) I_s(k_x + 1) - P_x(k_x) I_s(k_x)] \quad \text{if } k_x \neq 0 \qquad (1)$$

Using an *M/M/1* model [K75], we can compute the rate of the work (instructions per second) that arrives to the system without considering the new request:

$$\lambda = \mu - \frac{1}{W} \qquad (2)$$

If the owner accepts the negotiation, the new rate will be:

$$\lambda_{new} = \lambda + \lambda_{extra}^{qc} \qquad (3)$$

$$W_{new}^{qc} = \frac{1}{\frac{1}{W} - \lambda_{extra}^{qc}} \qquad (4)$$

To compute the cost, the only remaining difficulty is the estimation of P_x. Clearly, the stricter the coherency conditions, the larger P_x will be. A simple, worst case model would set $P_x = 1$ for all k_x. In general, $P_x(k_x)$ will be a monotonically increasing function of k_x. Below, we present two methods for computing the function P_x. The first is suitable for numerical data, and the second becomes useful when the coherency conditions are

expressed in terms of versions.

(1) Numerical data. A simple technique for estimating P_x may be derived from the assumption that update values follow a normal distribution with mean equal to the quasi-copy value value x', and variance σ. If $\Phi(x)$ is the standard normal distribution function, the distribution function for the update random function X would be:

$$P(X \leq x) = \Phi(\frac{x - x'}{\sigma}) \quad (5)$$

Now, the probability that an update results in a broadcast may be computed as $Prob[X \leq x_{highest\_low} \text{ or } X \geq x_{lowest\_high}]$ (see Figure 4.1), with $x_{highest\_low}$ and x_{lowest_high} representing the extremes of the tolerance interval Δ_x, as defined in Section 2. Let Δ_x be the current tolerance interval, with k_x quasi-copies granted. (If no quasi-copies have been granted so far, we may define Δ_x as infinite.) At this point we can compute Δ_x' (the tolerance interval that would apply if we grant the new quasi-copy) by applying the coherency condition $P(x)$ to the value x'. That is, we can compute the values $x'_{highest\_low}$ and x'_{lowest_high}. We can use those two values in turn to compute now $P_x(k_x + 1)$ as follows:

$$P_x(k_x + 1) = 1 - \quad (6)$$
$$[\Phi(\frac{x'_{lowest\_high} - x'}{\sigma}) - \Phi(\frac{x'_{highest\_low} - x'}{\sigma})]$$

(2) Versions. Let us define the value $\beta_{min}(x)$ as the minimum version difference requested so far by the importers to whom a quasi-copy of x has been granted. (If none has been granted, $\beta_{min}(x)$ is defined as infinite.) Let $P(x) = V(x)$ be the coherency condition under negotiation. Define $\beta'_{min}(x) = min[\beta_{min}(x), V(x)]$. We may now compute $P_x(k_x + 1)$ as follows:

$$P_x(k_x + 1) = \frac{1}{\beta'_{min}(x)} \quad (7)$$

Equation (7) follows from the fact that the object must be updated $\beta'_{min}(x)$ times before a broadcast is triggered. For instance, if $\beta'_{min}(x) = 3$, the broadcast will occur after the third update. Thus, over time the probability that an update triggers a broadcast will be $\frac{1}{3}$.

To recap the results of this section, in the heterogeneous case the importer and owner sites

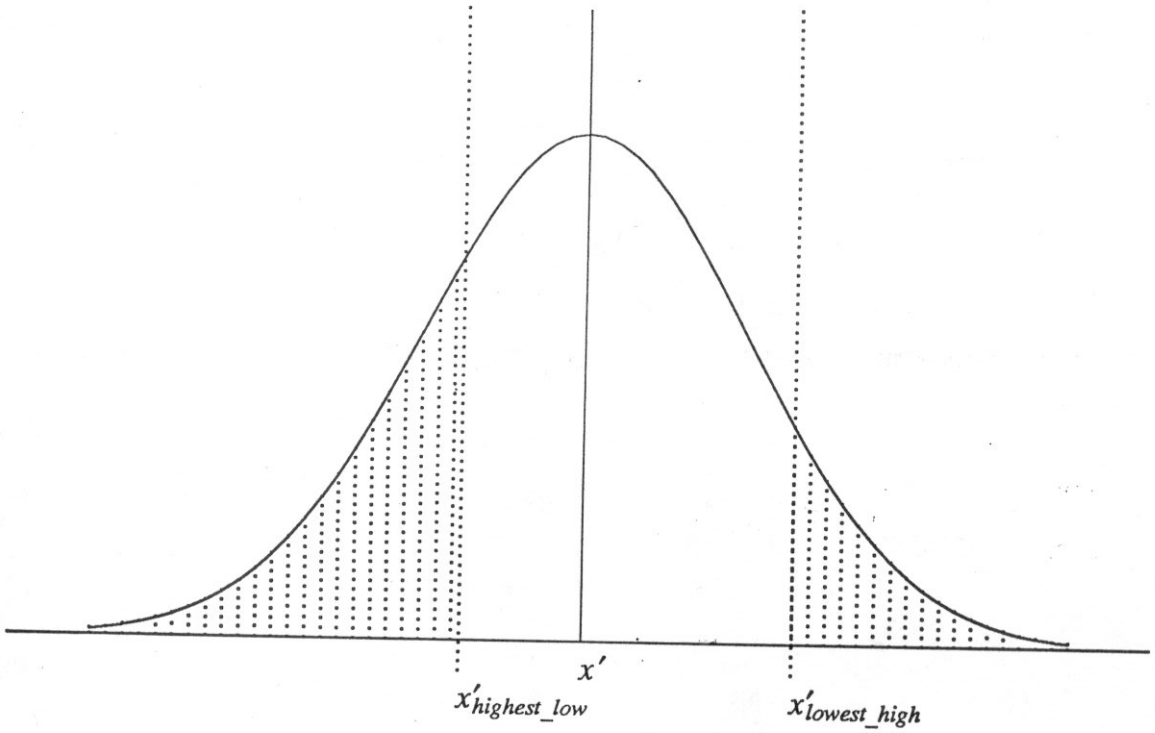

Figure 4.1

may only share information via quasi-copies. When the owner site receives the quasi-copy request, it compares W_{new}^{qc} to $W_{\max}$ using Equation (4). If $W_{new}^{qc} \leq W_{\max}$, the quasi-copy is granted, otherwise access to the data is denied.

4.2. The Homogeneous case.

In some federated databases, some of the databases in the network may share the same query language (alternatively, a query translation mechanism exists). If so, we will allow the owner site the freedom to give the importer either a quasi-copy or the right to query directly the database. The cost of granting a quasi-copy in this situation would be identical to that of the heterogeneous case which we have already discussed in the previous section. Thus, all that remains to be considered is the cost of granting direct access to the owner's database.

Using the same parameters that we have already defined in Section 4.1, the extra cost of direct access would be:

$$\lambda_{extra}^{da} = \lambda_i^q (I_q + I_s(1)) \tag{8}$$

Using this result we can compute

$$\lambda_{new} = \lambda + \lambda_{extra}^{da} \tag{9}$$

and

$$W_{new}^{da} = \frac{1}{\frac{1}{W} - \lambda_{extra}^{da}} \tag{10}$$

Using the equations (1)-(4), the owner can compute the cost of granting a quasi-copy and then compare W_{new}^{qc} with W_{new}^{da} and $W_{\max}$. The possible outcomes are:

(1) Access is denied, if $W_{new}^{qc} > W_{\max}$ and $W_{new}^{da} > W_{\max}$.

(2) A quasi-copy is granted, if $W_{new}^{qc} < W_{new}^{da}$ and $W_{new}^{qc} \leq W_{\max}$.

(3) Query rights are granted, if $W_{new}^{da} < W_{new}^{qc}$ and $W_{new}^{da} \leq W_{\max}$.

The careful reader will note that in this section we have assumed that the importer will have no preference for direct access over obtaining a quasi-copy, and thus the owner is free to choose the option that will result in the least expense to him or her. If the importer will rather have direct access, we can still compute direct access costs as before, but grant the importer direct access as long as $W_{new}^{da} < W_{\max}$. Only if $W_{new}^{da} > W_{\max}$ will it be worthwhile considering a quasi-copy.

On the other hand, some importers will rather have a quasi-copy than direct access. This is not so strange if we consider that, in the case of a network failure, direct access customers are left without service, while quasi-copy importers at least have some partially correct data available for their users. For these importers the sequence of cost comparisons changes in the obvious way.

5. Conclusions

We have presented a protocol for negotiating access to data in a federated database. This protocol deals with several important aspects of data sharing in an environment where databases belonging to different organizations coexist and cooperate. This aspects are:

- Allowing the owner of the requested data to evaluate precisely the expense it will incur if sharing is accepted.
- Allowing the sites involved to deal more efficiently with heterogeneity problems, since owners may now give a quasi-copy to the importer. That way the problems of query translation disappear.
- Dealing in a systematic and precise way with the problem of data consistency. The importer specifies exactly what manner of consistency it needs, and the owner decides whether it can commit itself to maintain such a degree of consistency over the shared data.
- The use of quasi-copies is also an effective way of off-loading work from the owner site. By giving the importer a quasi-copy, much of the querying processing costs are off-loaded to the importer site.

As shown in Section 4, a key question in evaluating the cost of sharing is the estimation of the probability of triggering a broadcast. We have presented two techniques that can be used for numerical data and for copies that rely on version numbers. We believe that these two ideas encompass many of the interesting cases in practice. However, it is also possible to fine tune the estimation of this function once the exact semantics of the data involved are known.

We are currently working on a prototype implementation of a federated database system in which these ideas can be tested in practice.

6. References

[ABG88] Alonso, Rafael, Daniel Barbará, Hector Garcia-Molina, and Soraya Abad, ''Quasi-Copies: Efficient Data Sharing for Information Retrieval Systems,'' *Proceedings of the International Conference on Extending Database Technology,* Italy, 1988.

[D81] Date, C.J. An Introduction to Database Systems. Vol. I, 3rd Ed. Addison-Wesley. 1981.

[D84] Dunn, Bill, ''Bill Dunn of Dow Jones: The Data Merchant,'' *Personal Computing,* pp. 162-176. December 1984.

[ED83] Edelhart, Mike and Owen Davies, *OMNI Online Database Dictionary,* Collier MacMillan Publishers, 1983.

[HM85] Heimbigner, Dennis, and Dennis McLeod, ''A Federated Architecture for Information Management,'' *ACM Transactions on Office Information Systems,* vol. 3, no. 3, pp. 253-278, July 1985.

[K75] Kleinrock, Leonard, Queueing Systems. Vol. 1: Theory. Wiley-Interscience 1975.