Polyvalues: A Tool for Implementing Atomic Updates to Distributed Data[*]

Warren A. Montgomery[**]
M.I.T. Laboratory for Computer Science
545 Technology Square
Cambridge, Ma, 02139

ABSTRACT

The coordination of atomic updates to distributed data is a difficult problem in the design of a distributed information system. A common goal for solutions to this problem is that the failure of a site should not prevent any processing that does not require the data stored at that site. While this goal has been shown to be impossible to achieve all of the time, several approaches have been developed that perform atomic updates such that most site failures do not affect processing at other sites. This paper presents another such approach, one that provides a mechanism by which processing can proceed even if a failure occurs during a critical moment in an atomic update.

The solution presented is based on the notion of maintaining several potential current values (a polyvalue) for each database item whose exact value is not known, due to failures interrupting atomic updates. A polyvalue represents the possible set of values that an item could have, depending on the outcome of transactions that have been delayed by failures. Transactions may operate on polyvalues, and in many cases a polyvalue may provide sufficient information to allow the results of a transaction to be computed, even though the polyvalue does not specify an exact value. An analysis and simulation of the polyvalue mechanism shows that the mechanism is suitable for databases with reasonable failure rates and recovery times. The polyvalue mechanism is most useful where prompt processing is essential, but the results that must be produced promptly depend only loosely on the database state. Many applications, such as electronic funds transfer, reservations, and process control, have these characteristics.

1. INTRODUCTION

Many factors currently favor the distribution of data storage and processing among sites in a distributed information system. These factors include greater reliability, better administrative control, faster response, and economics.

In order to realize fully the advantages of a distributed system, that system must be constructed so as to allow it to function as a whole, yet allow the individual sites to retain autonomy. By autonomy, I mean that the storage and processing at each site can be individually controlled, and that each site is capable of independent operation if it is separated from the rest.

The processing operations in a distributed information system are known as transactions. Several transactions may be performed concurrently in a distributed information system. A difficult problem in meeting the above goals for a distributed information system is the coordination of concurrent transactions so that concurrent execution does not produce results that could not be achieved by performing all processing serially. This coordination allows the user to view the distributed system as a large fast machine that processes his transactions serially, which frees the user from the need to worry about possible interactions among concurrent transactions.

In order to take advantage of the independence of failures of individual sites in a distributed system, and to allow for the autonomy of individual sites, the transaction coordination mechanism must allow a transaction to be performed promptly whenever the sites that hold the data items accessed by that transaction are functioning and can communicate. In other words, the failure of a site should not indefinitely delay any transaction that does not access data stored at that site. A simpler condition that follows from that above is that no transaction that is entirely local to one site should be delayed indefinitely by the failure of some other site.

* This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract No. N00014-75-C-0661.

** Author's Current Address:

Bell Telephone Laboratories
Naperville, IL 60540

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1979 ACM 0-89791-009-5/79/1200/0143 $00.75

A great deal of recent research has addressed the problem of transaction coordination; specifically, coordination of updates made by the transactions to the distributed data. Several papers [1,3,5] have presented arguments to demonstrate that there can be no protocol that implements atomic updates to distributed data in such a way that a failure of some site does not postpone updates that do not involve data at that site. These arguments basically show that if a failure disrupts communication among sites during an update, there is no way to determine which of the sites have completed the update and which have not. Thus one cannot assume that a site with which communication has failed has or has not completed the update, and one must in general wait until communication with that site is restored before proceeding.

2. APPROACHES TO DISTRIBUTED UPDATES

Given that one cannot have an update mechanism that meets the desired goals of atomic transactions and independent operation of the sites, we must take an "engineering approach" to the problem, one that meets these goals most of the time. Several such approaches are possible. These can be used individually or in various combinations in designing a mechanism to coordinate transactions in a distributed information system.

2.1 Lock Avoidance

One approach that has been used is to structure the implementation of the transactions such that it avoids the need to make atomic updates wherever possible. This can be done by pre-analyzing the transactions to be performed to determine whether or not they require an atomic update. This approach was used in [2] and [5].

2.2 Window Minimization

A second, widely used approach is to structure the update algorithm such that the time period during which a failure could cause delay due to the need to synchronize transactions is small. This approach is used in the update protocols of Gray [3], Lampson and Sturgis [4], and Reed [6]. Basically, these protocols are structured so that all of the results of a transaction are computed before any updates are made. Then, a simple set of message exchanges is used to decide whether or not to carry out the updates, based on whether all sites finished the computation promptly. If a failure occurs during the presumably much longer computation part of the protocol, the transaction can be aborted and no updates will be made.

Another example of this approach is given by Thomas's algorithm for synchronizing updates to distributed data [7]. This algorithm uses a complicated voting protocol to decide when to install the results of the transactions. While the voting may take place over an extended period of time, the window in which a site failure may leave the functioning sites in doubt as to whether or not to complete the transaction is small.

2.3 Relaxed Consistency

A third approach is to have each site that is involved in a transaction when a failure occurs make an arbitrary decision as to whether to complete or abort the transaction. This introduces the possibility that a transaction may be performed incorrectly (some but not all of the updates performed) if a failure occurs at a critical moment. Each site can, however, continue processing and need not wait for failure recovery to proceed. Because correctness is more important than timeliness for most applications, this approach has not been widely used.

2.4 Polyvalues

A fourth approach, extensively described in the remainder of this paper, is based on the notion of maintaining two or more possible current values for the items involved in an update interrupted by a failure. If both new and old values are known for those items, frequently that information is sufficient to allow processing of those items by subsequent transactions. The polyvalue scheme described in the following section is a generalization of this notion to allow continued transaction processing by sites involved in a transaction interrupted by a failure.

3. THE POLYVALUE MECHANISM

The following terms will be used in discussing the polyvalue mechanism for distributed updates:

- A database is a set of data items.
- A database state corresponds to an assignment of a particular value for each item in the database.
- A transaction is a mapping from one database state to another database state.

In a distributed database, each item is stored at one of the sites. (An item that is replicated at several sites can be viewed as a set of individual items, one for each site.) Each transaction involves directly only those sites that hold the data items accessed by the transaction (the items whose values are changed by the transaction, and those needed to compute the changes made by the transaction).

This provides an adequate framework for discussing the problem of coordinating transactions. The constraint that transactions be atomic can now be expressed as meaning that the database state reached by an execution of a set of transactions must be the same as that reached by some serial execution of the transactions, in which one transaction is completed before the next is begun.

Polyvalues are a simple extension to this model that allow each item to have several possible current values. The polyvalue mechanism specifies how transactions act on such items.

A polyvalue is a bookkeeping tool for keeping more than one value for an item. A polyvalue is a set of pairs <v,c>, where v is a simple value, and c is a condition which is a predicate. The variables in a condition stand for transactions, and are known as transaction identifiers. For example, the condition T1 (T2 T3) would be true if T1 and at least one of T2 and T3 were completed. The value of c in the pair <v,c> indicates the condition under which v is the correct value represented by the polyvalue. The conditions on the pairs in each polyvalue must be complete and disjoint. (One and only one of the predicates must be true under any assignment of truth values to the transaction identifiers).

There are two circumstances that cause an item to be updated with a polyvalue. A polyvalue is assigned to an item if a failure delays a transaction that is updating that item, or a polyvalue may be produced as one of the results of a transaction that accesses an item that has a polyvalue. These circumstances are explained in the next two sections.

3.1 The Update Protocol

Transactions are performed with a two-phase protocol [3]. In the first (compute) phase, each site computes the new values (updates) for the items that it holds that are specified by the execution of the transaction. If a failure delays the completion of the compute phase of a transaction at some site (by preventing communication with some other site), then that site simply discards the computation performed for the transaction and continues processing transactions as if the transaction interrupted by the failure had never occurred.

When each site finishes its compute phase, it then reports that it is ready to a site that has been designated as the transaction coordinator for that transaction by sending a ready message, and enters the second (wait) phase. After the transaction coordinator has received ready messages from all sites involved in the transaction, it sends out complete messages to all of those sites. If ready messages are not promptly received by the coordinator, then the coordinator sends out abort messages to all sites.

When a site receives a complete message, it installs (makes current) all of the values computed for the results of the transaction, thus completing the execution of the transaction. If a site receives an abort message, it discards any computation done by the transaction, and continues processing other transactions.

If neither a complete nor an abort message is received by a site promptly after entering the wait phase, then that site knows that some failure has interfered with the completion of the transaction, but does not know whether the eventual decision of the coordinator will be to complete or to abort the transaction. This decision is referred to as the outcome of the transaction. When a site is uncertain about the outcome of a transaction, it installs polyvalues for the results of that transaction.

Each such polyvalue is constructed as {<v,T>, <v'¬T>}, where v is the new value computed by the transaction, v' is the previous value, and T is a transaction identifier for the transaction. This polyvalue indicates that if T is completed, then v is the correct value, otherwise v' is correct. Before installation, polyvalues are simplified by:

1. Expanding any pair <v,c>, where v is a polyvalue to a set of pairs $<v_i, c_i \wedge c>$ where $<v_i, c_i>$ are the pairs in v. This eliminates nesting of polyvalues which can occur when polyvalues are updated with polyvalues.

2. Combining any two pairs $<v_1,c_1>$ and $<v_2,c_2>$ where $v_1 = v_2$ to form a single pair $<v_1, c_1 \vee c_2>$. This type of simplification occurs when the same value is computed under two sets of conditions.

3. Reducing each predicate to sum-of-products form, and discarding any pair <v,c> for which c is logically false.

The simplification procedure insures that all of the polyvalues in the database have a minimal number of pairs, each pair with a simple value.

Figure 1 shows a state diagram for this simple protocol. Each site can be in one of three states: idle, compute, or wait. In the idle state, a site is ready to begin a new transaction and enter the compute state. In the compute state, a site computes the results of a transaction. If those results are promptly computed, the site enters the wait state, sending a ready message to the transaction coordinator. If the results cannot be promptly computed due to a failure, or if the site receives an abort message, the site moves from the compute state to the idle state, discarding any computation done by the transaction. From the wait state the site can return to the idle state in one of three ways. If the site receives a complete or abort message from the coordinator, the site either installs or discards the results of the transaction, and returns to the idle state. If neither message is promptly received, the site installs polyvalues for the items updated by the transaction and returns to the idle state.

3.2 Transactions that Access Polyvalues

A polyvalue describes the possible values of an item whose correct exact value is not known, due to a failure. Such items may participate in transactions. A transaction that accesses an item with a polyvalue becomes a polytransaction. The compute phase of a polytransaction differs from a transaction that operates only on simple values, in that a polytransaction must compute new values based on polyvalued inputs.

Each polytransaction T consists of a set of alternative transactions $\{T_c\}$, each of which performs the transaction T on a different database state. Each alternative transaction T_c is tagged with a condition, c, which is derived from the conditions on the input values read by T_c. Whenever a transaction begins execution, it has a single alternative T_{true}. When an alternative transaction T_c accesses an item with a polyvalue $\{<v_i,c_i>\}$, T_c is partitioned into a set of alternative transactions $\{T_{c \wedge ci}\}$. $T_{c \wedge ci}$ has the same execution history as

T_c, and accesses the value vi for the item with the polyvalue.

For each item updated by T, a polyvalue is constructed as {<v1,c1>, ..., <vn,cn>} where vi is the value computed by alternative transaction T_{ci}, or is the previous value of the item if transaction T_{ci} does not compute a new value for the item. This polyvalue is simplified and installed according to the protocol for the wait phase of T as described above. The wait phase of a polytransaction does not begin until all of the alternative transactions have finished computation.

The rules for evolving the conditions on the alternative transactions insure that the conditions on the alternative transactions, and therefore the conditions in the polyvalues produced by T, will be complete and disjoint. Substantial improvements can be made in the efficiency of the computation performed by polytransactions, by recognizing alternatives T_c for which c is logically false. Any such alternative transaction can be discarded, as its results can never contribute to the values assigned to items by T. One can also recognize cases where the actual value of an item accessed by a transaction does not affect the computation performed by the transaction, and thus need not cause partitioning.

This mechanism allows computation by transactions to proceed in a database where the exact values of the items may not be known, due to the updates that may be made by transactions that have been suspended by failures. The transactions propagate the uncertainty in the values in the database by computing polyvalues as outputs. Any transaction whose outputs do not depend on the exact correct value of a polyvalued input item produces simple values and thus does not propagate uncertainty.

3.3 Failure Recovery

When a failure that has interrupted the wait phase of some transaction is recovered, the sites involved in that transaction can complete the wait phase, deciding whether to complete or abort the transaction. This knowledge can be used to reduce the polyvalues that depend on the outcome of the transaction. The value of the transaction identifier for such a transaction can be replaced by true or false in the predicates in the polyvalues, depending on whether the transaction is completed or aborted, respectively. The polyvalues can then be simplified, and when the outcome of every transaction is known, a single value pair will be left in each polyvalue, eliminating all uncertainty from the database.

Because of the propagation of polyvalues by polytransactions, the sites that may hold polyvalues dependent on the outcome of a transaction T, are not limited to the sites involved in T.

Each site with a polyvalue dependent on the outcome of a transaction T must be informed of that outcome, and any data structures used to keep track of the transaction outcome should be quickly deleted when no longer needed. The responsibility for informing the sites with polyvalues dependent on T of the outcome of T among those sites can be distributed among the sites. Each site maintains a table recording, for each transaction T whose outcome is unknown a list of the polyvalues held by the site that depend on T, and a list of other sites to which polyvalues dependent on T have been sent. When a site learns the outcome of a transaction T, it can reduce the polyvalues that it holds that are dependent on the outcome of T, by consulting the table. The site must inform all of the sites listed in its table entry for T. Once this is done, that site can forget the outcome of T and the table entry for T. This scheme quickly eliminates the polyvalues dependent on T from the database. The data structures used in the mechanism are also quickly removed.

3.4 External outputs of a system using polyvalues

The preceding sections have dealt with the mechanisms for maintaining the internal state of the database during a failure. The effect of a failure on a database using polyvalues is to introduce uncertainty into the values maintained for the database items. This uncertainty may or may not be reflected in the outputs produced by the system. For many applications, such as authorizing reservations or credit transactions, the outputs of the database visible to the users do not depend on the exact values of the data, so that uncertainty in the values may not be reflected in the outputs of the database system.

When uncertainty is reflected in the outputs of the database system, two options are available: present the uncertain outputs to the user, or withhold those outputs until the uncertainty is resolved. For many applications, presenting uncertain outputs to the user would be appropriate. Most of the time, a ticket agent would not be bothered by an uncertain answer to a request for the number of seats remaining on a flight. For some applications, such as those requiring a yes or no answer, uncertain outputs are useless. Both choices (waiting, or presenting the uncertain outputs), are available in a system using polyvalues whereas without polyvalues, one must wait for the failure to be recovered.

4. ANALYSIS OF THE POLYVALUE MECHANISM

The polyvalue mechanism presented in the previous section allows transactions to be performed on data items that were involved in a transaction suspended by a failure, at the cost of additional storage and processing in performing transactions. One concern in using this scheme is that the number of items with polyvalues, and thus the number of polytransactions, will become large and expensive. This section presents an analysis of the expected number of polyvalues in a database using this mechanism, and a simulation of a database using the polyvalue mechanism.

4.1 A Model for Polyvalue Creation and Deletion

We can express the net rate at which polyvalues in the database are created as the rate at which new polyvalues are created by failures, plus the rate at which they are created by polytransactions, less the rate at which failure recovery eliminates polyvalues, less the rate at which transactions overwrite polyvalues for items by updating those items with simple values. These can in turn be expressed in terms of the following parameters of the database.

- I The number of items in the database.
- $P(t)$ The number of items with polyvalues at time t.
- U The number of updates made per second.
- F The probability that an update will fail.
- R The proportion of failures recovered each second.
- D The average number of items on which the new value assigned to an updated item depends.
- Y The probability that the new value of an updated item will not depend on its previous value.

The expected rate of change of the number of polyvalues in the database can be expressed using these parameters. The model presented here is a first order model in that the term $(1-P(t)/I)$ has been replaced by 1 in all of the equations. This model is valid, so long as the proportion of items with polyvalues remains small, and greatly simplifies the mathematics.

$$P'(t)=UF + UD\frac{P(t)}{I} - UY\frac{P(t)}{I} - R\,P(t)$$

Solving this simple linear differential equation yields:

$$P(t)=\frac{UFI}{IR+UY-UD} + C\,e^{-\left(\frac{IR+UY-UD}{UFI}\right)t}$$

The number of polyvalues can be expected to approach a constant, $P = (UFI)/(IR+UY-UD)$, as time progresses. This is the number of polyvalues that we would expect to find in the database, averaged over a long period of time, provided that the parameters accurately describe the operation of the database.

Two points about this solution should be noted. First, it is stable in that if the number of polyvalues temporarily becomes larger than the predicted (steady-state) number, then the number of polyvalues can be expected to decrease with time. A serious failure causing the introduction of many polyvalues does not cause the number of polyvalues to grow without limit.

A second point is that this solution is only valid when the number of polyvalues is small compared to the number of database items. When this condition is not met, the approximations made invalidate the solution. For our purposes, this solution is adequate, as we would not want to operate a system with parameters such that the number of items with polyvalues becomes significant compared to the total number of items. Thus the fact that the equations predict a very large number of polyvalues for some parameter values suggests that one would not wish to operate a database with such values, but does not accurately predict the number of polyvalues that would result.

The predicted number of polyvalues for some particular values of the parameters is given in Table 1. The parameters for the first table entry were chosen to reflect a typical database to which polyvalues may be applied. The remaining table entries show how varying each of the parameters individually effects the predicted number of polyvalues. Space limitations in this paper prevent a thorough exploration of the parameter space, however the individual effects of the parameters can be clearly seen from the equations and the data.

4.2 Simulation of the Polyvalue Mechanism

In order to gain confidence that the model accurately describes the behavior of the system, and to discover the behavior of the system when the number of polyvalues is large, a simulation of a database system using polyvalues was performed.

The simulation maintained a description of the items of the database having polyvalues, and the transactions on which those items depended. Transactions were introduced at a rate U. Each transaction updated a single item chosen at random from the database. This update depended on a set of d items, also selected at random, where d was chosen from an exponential distribution with mean D. The previous value of the updated item was included in its new value with probability (1-Y).

For the results reported here, a uniform distribution was used for the random selections of items from the database. In a real system, the selection of items to participate in transactions is not likely to be uniform. Some items may participate in transactions much more frequently than others. This has the effect of reducing the effective size of the database.

Transactions were chosen to fail with probability F. For a failed transaction, a polyvalue was created for the item that it updated and a recovery time was chosen from an exponential distribution with a mean value of 1/R. As noted above, each item with a polyvalue is tagged with the identity of all transactions on which the polyvalue depends. When a failure is recovered, the tag for the recovered transaction is removed from all polyvalues, and any polyvalue with no remaining tags is converted to a simple value.

The number of polyvalues for a particular set of parameters can be obtained by running the simulation with that set of parameters until the number of polyvalues has remained stable for some time, and then taking the average number of polyvalues in the database during such a stable period. The implementation of the simulation restricted the range of the parameters for which simulations can be performed to relatively small databases. Table 2 reports the results of several simulation runs with different sets of parameters. The results agree well with the predictions of the model in the area where the number of polyvalues is small. The number of polyvalues obtained in the simulation is in general smaller than predicted. This is a result of the approximations made in obtaining the prediction.

This model and simulation demonstrate that the number of polyvalues in the database is expected to be small for reasonable failure rates and recovery times. Thus the cost of the computation and storage to support polyvalues should remain relatively small.

5. POSSIBLE APPLICATIONS FOR POLYVALUES

The polyvalue mechanism is best suited to applications where rapid processing of some transactions is essential, and where the most important results depend only loosely on the values of the data items in the database. If this is the case, the important transactions will frequently produce simple output values, even when the database contains polyvalues. There are many such applications now planned for distributed information systems.

Electronic funds transfer or credit authorization are good examples. The important transactions in such a system are those that authorize transfers of "real" money or goods, such as transactions to cash checks or authorize credit purchases. To satisfy customers, such transactions must be performed promptly, even if failures in the database system have interfered with other transactions. Such transactions depend very loosely on the state of the database in that the important effect (distribution of funds or goods) depends only on the fact that the relevant accounts contain enough funds, not on exactly how much. Such a system can tolerate much uncertainty in the database, so long as the uncertainty is eventually resolved when failures are recovered.

Another example is a reservations system. In this case, the important transactions and effects are the granting of reservations to customers. This can frequently be done without knowing the total number of such reservations granted. If the number of reservations granted is a polyvalue, then a new reservation can be granted so long as the largest value in that polyvalue is less than the number of available rooms or seats. This will be discovered when the reservation-granting transaction is run as a polytransaction: All alternative transactions of such a polytransaction will decide to grant the reservation.

Such applications as inventory or process control also seem ideal candidates for the polyvalue mechanism. Again, real time operation is important; however, the exact values of the items in the database are frequently not needed for the important real time effects.

6. CONCLUSIONS

The polyvalue mechanism presented in this paper is an effective solution to the problems presented by the need to make atomic updates to distributed data without disrupting the processing of transactions by the system if a failure delays such an update. The mechanism appears to have many applications in currently planned uses of distributed information systems.

The polyvalue mechanism can be combined with other atomic distributed update protocols to decrease the chance that polyvalues will be created. The mechanism as presented uses the two-phase commit protocol of Gray [3], but is compatible with other such schemes, such as the protocol of Lampson and Sturgis [4], or that of Reed [6].

Analysis and simulation have shown that the extra storage and processing required to support this mechanism are small, given reasonable failure rates and repair times.

ACKNOWLEDGEMENTS

I would like to thank the members of the Computer Systems Research group of the M.I.T. Laboratory for Computer Science for their support during the research that lead to this paper. I would like to thank my wife, Carla, who suggested the term "polyvalue", and improved the expression of many of the ideas presented in the paper. I would also like to thank the reviewers for their suggestions, which substantially improved the final content of the paper.

REFERENCES

[1] Akkoyunlu, E. S, Ekandham, K., Huber, R.V., "Some constraints and tradeoffs in the design of network communications", Proc. Fifth Symposium on Operating Systems Principles, (November 1975) (Operating Systems Review, Vol. 9, No. 5).

[2] Bernstein, P.A., Shipman, D.W., Rothnie, J.B., and Goodman, N., "The concurrency control mechanism of SDD-1: A system for distributed databases (The general case)", Computer Corporation of America Technical Report CCA-77-09, (December 1977).

[3] Gray, J.N., "Notes on data base operating systems", in Operating Systems, An Advanced Course, in Volume 60 of Lecture Notes in Computer Science, Springer-Verlag (1978) pp. 393-481

[4] Lampson, B., and Sturgis, H., "Crash recovery in a distributed data storage system", Xerox Palo Alto Research Center, (To appear in comm. ACM).

[5] Montgomery, W. A., Robust concurrency control for a distributed information system", M.I.T. Laboratory for Computer Science Technical Report 207 (December 1978).

[6] Reed, D. P., "Naming and synchronization in a decentralized computer system, M.I.T. Laboratory for Computer Science Technical Report 205", (September, 1978).

[7] Thomas, R.H., "A solution to the update problem for multiple copy data bases which used distributed control", Bolt Beranek and Newman Inc. Technical Report No. 3340 (July 1976).

Figure 1: The Update Protocol States

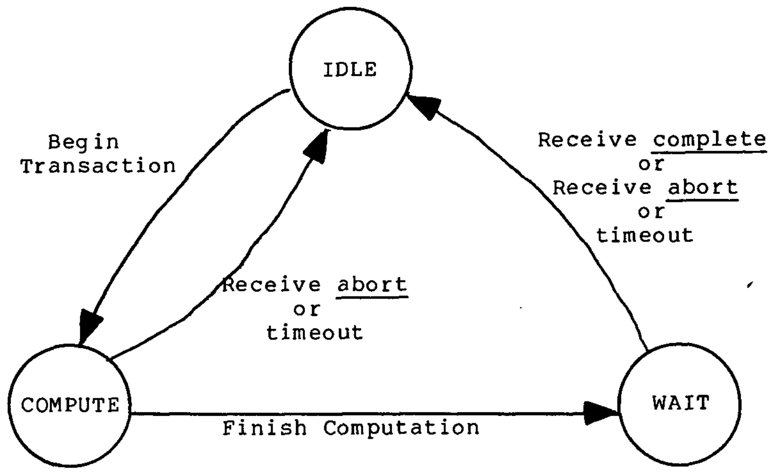

Table 1

Typical Predictions of the Number of Polyvalues in a Database

Parameters (see text)						Expected Number of Polyvalues
U	F	I	R	Y	D	P
10	0.0001	1,000,000	0.001	0	1	1.01
100	0.0001	1,000,000	0.001	0	1	11.11
10	0.0001	100,000	0.001	0	1	1.11
10	0.0001	100,000	0.001	0	5	2.00
10	0.0001	100,000	0.001	0	7	3.33
10	0.0001	100,000	0.001	1	1	1.00
10	0.0001	20,000	0.001	0	1	2.00
10	0.0001	11,000	0.001	0	1	11.00
10	0.001	1,000,000	0.001	0	1	10.10
10	0.005	1,000,000	0.001	0	1	50.50
10	0.0001	1,000,000	0.0001	0	1	11.00

Table 2

Results of Simulating the Polyvalue Mechanism

Parameters (see text)						Predicted	Actual
U	F	I	R	Y	D	P	P
2	0.01	10,000	0.01	0	1	2.04	2.00
5	0.01	10,000	0.01	0	1	5.26	2.71
10	0.01	10,000	0.01	0	1	11.11	9.5
10	0.001	10,000	0.01	0	1	1.11	0.74
10	0.01	10,000	0.01	0	5	20	19.8
10	0.01	10,000	0.01	1	5	16.7	15.8