Flexible Distributed Concurrency Control
by Reasoning About Transactions

A Proposal Submitted to the National Science Foundation

Charles Elkan
Department of Computer Science and Engineering
University of California, San Diego

elkan@cs.ucsd.edu

January 14, 1991

Contents

1 Project summary

The main goal of the research proposed here is the design and implementation of a novel transaction scheduler for a distributed database system. The scheduler will have the flexibility to adapt to changing patterns of access to the shared database, and the system will be fully distributed in the sense that neither concurrency control nor any other function will necessarily be performed at a central site. Flexibility and full distributedness will be achieved by explicit agreement between sites as to which transactions may proceed, which should be delayed, and which should be restarted. Scheduling will use a distributed consensus protocol and logical reasoning about when transactions conflict. The proposed work is mainly experimental, since it will apply already known algorithms. In particular, reasoning about conflict will be based on a comprehensive theory of query and update independence developed recently by the principal investigator [Elkan, 1989; Elkan, 1990b]. NSF support is requested now to move this theory into practice.

2 Project description

2.1 Introduction

The design of database systems is one field of computer science where academic research has had a significant impact on industrial practice. The two principal research achievements have been the development of relational database systems, and the invention of protocols that allow multiple users to access a shared database concurrently [NSF, 1990]. The theory of relational databases has yielded query languages that allow data to be accessed efficiently even in ways that were not foreseen when a database was established. The theory of transaction management provides algorithms for scheduling execution of access requests from multiple users of a database system so that they do not interfere with each other.

The core problem in managing concurrent transactions is to draw inferences about whether queries and updates performed on behalf of different users may cause inconsistencies. However, little work on concurrency control uses logical reasoning about queries and updates. The main reason is that efficient algorithms for performing this reasoning have not been known. We have developed suitable algorithms in recent work, and the focus of this proposal is the implementation of these algorithms in a fully distributed database system.

The flexibility of relational database systems arises ultimately from the fact that their query languages are "value-oriented." Information to be retrieved is defined by specifying its content partially. By contrast, older navigational database systems are "item-oriented." Information can only be retrieved when the user can name parts of the database in which to find it. Relational database systems today are still typically item-oriented in how they perform concurrency control. Their schedulers do not perform any inferences about the logical relationships between the information queried and updated by two transactions.

Concurrency control by reasoning about transactions has a number of important potential advantages. These stem from the separation made possible between conflict detection, conflict resolution, and access command execution. Common knowledge of which transactions are in conflict makes possible negotiation about restarting and delaying. Once a query or update has been proved not to cause interference, it can be executed with no further checking for conflict.

This project description is organized as follows. Section 2.2 describes previous work on value-oriented concurrency control, while Section 2.3 summarizes our recent results on reasoning about queries and updates. In Section 2.4 we discuss the features and the benefits of the value-oriented, fully distributed scheduler that we wish to build. Finally, Section 2.5 outlines our research plans more precisely.

2.2 The history of concurrency control by logical reasoning

The seminal paper on controlling the simultaneous execution of transactions in a database system was written by Jim Gray and others at IBM San Jose Research Center [Eswaran *et al.*, 1976]. At the time this paper was written, its authors were working on System R, the first large-scale implementation of a relational database system. Early designs for System R envisaged a value-oriented concurrency control scheme called predicate locking, but the scheme was abandoned in favor of an item-oriented alternative [Chamberlin *et al.*, 1981]. The main reason for abandoning predicate locking was that reasoning about predicates describing portions of the database appeared impossible to do adequately. With the benefit of hindsight, the difficulties can be identified as threefold [Elkan, 1990a].

(i) Inventing expressions describing read and write sets. The transaction compiler or the scheduler of a database system using value-oriented concurrency control must invent expressions that describe the parts of the shared database queried and updated by each transaction. These are called the read and write sets of the transaction. It is not obvious how to derive logical read and write set expressions efficiently from access commands written using standard query languages, especially since answering a query often requires retrieving data not present in the final answer. Item-oriented schedulers can avoid this problem by simply observing what database items are touched while executing each access command, but then it impossible to separate command execution from conflict detection and conflict resolution.

(ii) Checking whether transactions conflict. Conflict occurs if the read and write sets of two transactions overlap; that is, if the conjunction of the corresponding read and write set expressions is satisfiable. Satisfiability of general logical expressions is undecidable, so the inventors of predicate locking proposed that expressions should be restricted to $<$, $\leq$, $=$, $\geq$, $>$, and $\neq$ comparisons between attribute variables and/or constants combined by the standard logical connectives $\vee$, $\wedge$, and $\neg$. An example of an expression in this class describing good customers for a credit card company is

$$\mathsf{account} : (age > 18) \wedge (balance > 10000) \wedge (income > balance).$$

Unfortunately, even deciding whether the conjunction of two of these restricted expressions is satisfiable is $\mathcal{NP}$-complete [Hunt and Rosenkrantz, 1979].

(iii) Reducing spurious reports of potential conflict. The conjunction of two read and write set expressions may be satisfiable even though integrity constraints preclude the existence of any tuple actually satisfying both formulas. Predicate locking may therefore prevent transactions from executing concurrently that in fact can never conflict. The concurrent execution of transactions may also be prevented incorrectly for a second reason. Consider an access command involving a join, say $\mathsf{read}\ \mathsf{R} \underset{A=B}{\bowtie} \mathsf{S}$. A tuple only belongs to the read set relative to R of this command if its component value for the attribute A is the same as the value for the attribute B of some tuple of S. No expression referring only to attributes of R can state this connection, so a predicate locking scheduler must protect the command $\mathsf{read}\ \mathsf{R} \underset{A=B}{\bowtie} \mathsf{S}$ by locks on the entire relations R and S.

Predicate locking has always been recognized as an elegant theoretical idea, and a number of schemes have been advanced in the academic literature as attempts to overcome its difficulties. A number of commercially available database systems also use some form of predicate locking. It is worth noting that these schemes are all fixed protocols for centralized relational database systems, while we intend to construct a distributed scheduler that may use a variety of protocols for resolving conflicts.

Precision locking. The central idea of this scheme [Jordan *et al.*, 1981] is to maintain a special table where all tuples inserted or deleted by each transaction are recorded until the transaction terminates. Whenever any transaction executes a query, the scheduler applies the corresponding read set expression to all the tuples recorded by other active transactions in the table. If any of these satisfies the formula, then its transaction conflicts with the transaction executing the query. A scheduler that uses precision locking never tests the satisfiability of two read and write set expressions, because write set extensions substitute for write set expressions.

Precision locking is a step towards solving the second and third difficulties of predicate locking, since no satisfiability testing is done, and conflicts are only asserted if they actually occur. However the inventors of precision locking did not present any method of inventing expressions to describe the read sets of transactions. Moreover, satisfiability testing is replaced by the actual evaluation of read set expressions, which typically requires many extra database retrieval operations. Precision locking is therefore not a usable concurrency control scheme.

Expression locking. The principal idea of this scheme [Klug, 1983] is to require the user to provide read and write set expressions in relational algebra notation, and a tableau-based algorithm is given for satisfiability testing using this notation. Unfortunately the algorithm is inherently exponential. It uses tableaux that are incapable of representing $\leq$, $\geq$, or $\neq$ conditions, so a conjunctive query such as $\sigma_{A \leq B}(\mathsf{e})$ for example must be rewritten as $\sigma_{A<B}(\mathsf{e}) \cup \sigma_{A=B}(\mathsf{e})$, and represented by a set of two tableaux. The number of tableaux in the set representing an expression may be exponential in the length of the expression.

Expression locking has a second drawback. Checking whether the read and write set expressions provided by transaction programmers do cover its actual read and write sets is equivalent to deciding whether the set of tuples defined by one tableau is included in that defined by another, which is an $\mathcal{NP}$-hard problem [Aho *et al.*, 1979]. Expression locking is not a practical concurrency control scheme.

Other academic proposals. A number of further variants of predicate locking have been suggested in the published literature. So-called view locking [Blaauw and Duijvestijn, 1985] is similar to precision locking. The inventors of this scheme do describe how to derive read set expressions from access commands, but their notation is incapable of representing selections or projections, so describing read sets precisely is impossible.

So-called predicative optimistic concurrency control [Reimer, 1983] can be understood as an optimistic version of precision locking. Modified tuples and read set expressions are accumulated for all transactions, and each transaction is validated by checking that none of its modified tuples are in any read set of any other concurrent transaction. Unfortunately the query language of [Reimer, 1983] requires that read set expressions be coded explicitly and that they not involve joins. The problems of predicate locking are thus evaded but not solved.

In [Wheeler, 1986], extended unification algorithms are developed that can perform satisfiability testing for the original predicate locking class of expressions, but the limitations arising from the restricted nature of this class are not addressed.

Industrial applications of predicate locking. Commercially available database systems often incorporate good ideas that are never described in the academic literature. We know of three database systems that use variants of predicate locking to some degree. The UNIFY 2000 system developed by Unify Corporation (Sacramento, California) uses a combination of predicate locking and item-oriented locking where the items are tuples. The system CA-DB from Computer Associates International (San Jose, California), a company of which Eswaran was one founder, uses predicate locks. Finally, the IMAGE 3000 system of Hewlett Packard (Palo Alto, California) uses predicate locking as well as tuple and page locking.

One of the subsidiary aims of our research will be to investigate the design of the schedulers of each of these systems. We are especially interested in the reasons for the choices made by the designers of these systems. Their experience with large real-world database applications is something that academic researchers often do not have.

2.3 Reasoning about query and update independence

This section summarizes the results of our previous work on logical reasoning about queries and updates. The principal contributions are a formal study of the concept of independence, and an efficient decision procedure for testing independence that takes into account functional dependencies. Proofs of all the results below, and more, can be found in [Elkan, 1990a].

2.3.1 Sufficient conditions for independence

Intuitively, a query is independent of an update if executing the update cannot change the answer to the query. The necessary first step in studying independence is to define the notion formally. For this purpose we adopt the well-established framework of deductive databases [Ullman, 1988], letting relational databases be a special case. Let D be a collection of relations stored extensionally, and let R be a collection of rules defining relations intensionally. A query can be taken as an open logical formula $\phi(\bar{x})$ where $\bar{x}$ is the set of free variables of the query. Abstractly, the answer or meaning of $\phi(\bar{x})$ is

$$[\![\phi(\bar{x})]\!]_{D,R} = \{\bar{c} \in U^n : D, R \models \phi(\bar{c})\}$$

where n is the number of free variables in $\bar{x}$ and U is a universe of attribute values.

Definition: Let IC be a set of integrity constraints and let R be a set of rules. The query $\phi(\bar{x})$ is independent of the update $\mathsf{insert}\ Q(\bar{y}) : \psi(\bar{y})$ if for all extensional databases D satisfying IC, if $D \cup \Delta$ satisfies IC then

$$[\![\phi(\bar{x})]\!]_{D,R} = [\![\phi(\bar{x})]\!]_{D\cup\Delta,R}$$

where $\Delta = \{Q(\bar{y}) : \bar{y} \in [\![\psi(\bar{y})]\!]_{D,R}\}$. ■

Intuitively, $D \cup \Delta$ is the result of adding the set of tuples specified by the update to the original database. Independence for delete updates is defined similarly.

Given the preceding model-theoretic definition of independence, many proof-theoretic results can be obtained. For example, the following theorem allows one to prove that a query is independent of an update by showing that each logical component of the query is independent.

Theorem 1: Let $\phi(\bar{x})$ and $\psi(\bar{x})$ be queries over the same free variables, and let U be an update. If $\phi(\bar{x})$ and $\psi(\bar{x})$ are independent of U, then so are the queries $\phi(\bar{x}) \wedge \psi(\bar{x})$, $\phi(\bar{x}) \wedge \neg\psi(\bar{x})$, $\phi(\bar{x}) \vee \psi(\bar{x})$, and $\exists \bar{x}'\ \phi(\bar{x})$ where $\bar{x}'$ is a subset of the set of variables $\bar{x}$. ■

It is important to note that the converse of Theorem 1 does not hold. It is possible for $\phi(\bar{x}) \wedge \psi(\bar{x})$ to be independent of an update when neither or only one of the subqueries is. Less obviously, the same is true of $\phi(\bar{x}) \vee \psi(\bar{x})$, even when neither subquery is subsumed by the other. Thus it is not possible to obtain a decision procedure for independence by rewriting queries into some normal form. In general, all that can be obtained are inference rules based on sufficient conditions for independence.

Definition: The query $\phi(\bar{x})$ is conjunctive in Q if it is of the form $\exists \bar{z}\ Q(\bar{x}', \bar{z}') \wedge \phi'(\bar{x}, \bar{z})$ where $Q(\bar{x}', \bar{z}')$ is a positive literal with arguments $(\bar{x}', \bar{z}')$ taken from the free variables $\bar{x}$ and the existentially quantified variables $\bar{z}$, and Q is not used in rules defining relations mentioned in $\phi'(\bar{x}, \bar{z})$. ■

A conventional conjunctive query is conjunctive in each relation that appears in it, but all that is required for the following theorem is that a query be conjunctive in the relation affected by an update.

Theorem 2: Let IC be a background set of integrity constraints. The query $\phi(\bar{x})$ is independent of the update delete $Q(\bar{y}) : \psi(\bar{y})$ if it is conjunctive in Q and

$$IC, R \vdash \nexists \bar{x}, \bar{z}\ \phi'(\bar{x}, \bar{z}) \wedge Q(\bar{x}', \bar{z}') \wedge \psi(\bar{x}', \bar{z}'),$$

where $\phi(\bar{x}) = \exists \bar{z}\ \phi'(\bar{x}, \bar{z}) \wedge Q(\bar{x}', \bar{z}')$. ■

Theorem 2 and related theorems provide the formal justification for the assertion in Section 2.2 that testing conflict between transactions is reducible to satisfiability checking. For queries involving recursively defined relations, the situation is more complicated. The next theorem provides an inductive inference rule for this case.

Theorem 3: Let IC be a set of integrity constraints, and let R be a set of rules. Let U be an update and let $\phi(\bar{x}) \wedge P(\bar{x})$ be a query where P is defined by a set of recursive definite clauses. Without loss of generality, suppose the rules in R defining P are rectified so they can be written $P(\bar{x}) \leftarrow B_j(\bar{x})$ for $j = 1, \ldots, k$ where P may be mentioned in $B_j(\bar{x})$, but not used indirectly.

Let H be the hypothesis "$\phi(\bar{x}) \wedge P(\bar{x})$ is independent of U." If for each $j = 1, \ldots, k$

$$IC, R, H \vdash \phi(\bar{x}) \wedge B_j(\bar{x}) \text{ is independent of } U$$

then $IC, R \models \phi(\bar{x}) \wedge P(\bar{x})$ is independent of U. ■

In the long run we intend to provide deductive capabilities in the distributed database system that we propose to implement, so Theorem 3 will be important in the future.

2.3.2 A decision procedure for independence

Results such as Theorems 2 and 3 do not provide efficient decision procedures. Here we outline an algorithm that decides independence for conjunctive relational queries and updates. The algorithm is best-possible in two ways. Its time complexity can only be improved by solving the reachability problem for directed graphs in sublinear time, which is a known open problem. Moreover, we have shown that deciding independence is $\mathcal{NP}$-hard for nonconjunctive queries.

The first key idea is to represent conjunctive queries using a tableau notation accommodating all six comparison operators. This is accomplished by using a set of $\neq$ constraints and a set of $\leq$ constraints. Each comparison, say (*income* $>$ *balance*), is translated into a $\leq$ and/or a $\neq$ constraint: in this case, $(\mathit{balance} \leq \mathit{income}) \wedge (\mathit{income} \neq \mathit{balance})$. Previously introduced tableaux (for example those defined in [Klug, 1983]) only supplied either $\leq$ or $<$ constraints and therefore could not represent queries involving, among others, $\neq$ comparisons.

Formally, our tableaux are quadruples $\langle \mathbf{v}, L, N, T \rangle$ where the signature $\mathbf{v}$ is a list of attribute variables and/or constants (a template), L is a set of $\leq$ constraints, N is a set of $\neq$ constraints, and the tableau proper T is a set of pairs $\langle R, A \rangle$ where R is the name of a relation, and A is a set of templates associated with R. Intuitively a tableau describes a set of tuples given a database state, which is a collection of relation instances. The next definition captures this intuition.

Definition: The tableau $\mathbf{T} = \langle \mathbf{v}, L, N, T \rangle$ defines a mapping $I \mapsto \mathbf{T}(I)$ from database states to sets of tuples such that

$$\mathbf{T}(I) = \{\mathbf{v}\theta : \forall \langle x, y \rangle \in L \;\; x\theta \leq y\theta \;\wedge\; \forall \langle x, y \rangle \in N \;\; x\theta \neq y\theta \\ \wedge\; \forall \langle \mathsf{R_k}, A \rangle \in T \;\; \mathbf{a} \in A \;\rightarrow\; \mathbf{a}\theta \in I_k\},$$

where θ ranges over all mappings from variables to attribute values, and I_k is the instance of the relation named $\mathsf{R_k}$ in the database state I. ∎

The tableau corresponding to any conjunctive query has size proportional to the length of the query. It can be computed in linear time, and the tableau representing the intersection of two conjunctive queries is easy to compute. Given such a tableau $\mathbf{T}$ and a collection of functional dependencies, the task is to decide whether $\mathbf{T}(I) = \emptyset$ whatever the database state I such that the functional dependencies hold. The second key idea of the decision procedure for independence is that the following three conditional rewrite rules are sufficient.[1]

- *Equality inference.* If there exist two variables or constants x and y such that both $x \leq y$ and $y \leq x$ are deducible from the $\leq$ constraints of $\mathbf{T}$, then $\mathbf{T} \rhd \mathbf{T}[x \equiv y]$.
- *Functional dependency propagation.* Suppose $\mathsf{R}_k{:}F_1 \cdots F_p \rightarrow G_1 \cdots G_q$ is an imposed functional dependency. If there exist templates $\mathbf{a} \in A$ and $\mathbf{b} \in A$ with $\pi_{F_1 \cdots F_p}(\mathbf{a}) = \pi_{F_1 \cdots F_p}(\mathbf{b})$, where $\langle \mathsf{R}_k, A \rangle$ belongs to the tableau proper of $\mathbf{T}$, then $\mathbf{T} \rhd \mathbf{T}[\pi_{G_1 \cdots G_q}(\mathbf{a}) \equiv \pi_{G_1 \cdots G_q}(\mathbf{b})]$.
- *Inconsistency detection.* If $c \leq d$ is deducible from the $\leq$ constraints of $\mathbf{T}$, where c and d are two constants such that $c > d$, or $x \neq x$ is deducible from the $\neq$ constraints of $\mathbf{T}$, where x is any variable or constant, then $\mathbf{T} \rhd \perp$.

[1] The notation $\mathbf{T} \rhd \mathbf{T}[A \equiv B]$ means $\mathbf{T}$ is rewritten to $\mathbf{T}$ with A and B identified.

It is possible to prove that these three rules are confluent and terminating. Also, if all variables range over dense domains, there exists some valid database state I such that $\mathbf{T}(I) \neq \emptyset$ if and only if $\mathbf{T}$ cannot be rewritten to $\perp$. Applying the rules repeatedly until none are applicable is thus a decision procedure for unsatisfiability.

The task that remains is to find a good representation for tableaux and a good strategy for ordering application of the rules. This is discussed in detail in [Elkan, 1990a]. L is represented as a directed acyclic graph, so applying the equality inference rule becomes reachability testing by depth-first search. Applying the functional dependency propagation rule is reduced to computing a congruence closure [Downey *et al.*, 1980], and all three rules are coordinated by propagating equalities [Oppen, 1980]. The outcome is an algorithm that uses $O(\|N\| + \|L\|^2 + \|T\| \log \|T\|)$ time on an input tableau $\langle \mathbf{v}, L, N, T \rangle$ if the number of functional dependencies applicable to each relation is a constant.

Imagine a relational database system storing information about stockbrokers, their offices, and their customers. Suppose that the database contains, among others, two relations specified using a tabular data definition language as follows.

name	*schema*	*interpretation*
client	PB	person P is a client of broker B
office	BC	broker B is based in city C

Suppose further that two functional dependencies are imposed: every person is known to be a customer of at most one broker, and every broker has an office in at most one city. (These assertions are not realistic in the United States in the 1990s, but they might well be legally enforced in some other jurisdiction.) Now suppose one user of the database system wants to update the information recorded in the client relation by deleting all $\langle P, B\rangle$ pairs such that the office of the broker B is located in Los Angeles. Using relational algebra notation, this update can be expressed as

$$\mathsf{delete\ client:}\ \pi_{PB}\left(\sigma_{C=LosAngeles}(\mathsf{client} \bowtie \mathsf{office})\right).$$

Suppose that at the same time another user wants to see a list of all customers of brokers whose offices are in New York. This query can be expressed as

$$\mathsf{read}\ \sigma_{C=NewYork}(\mathsf{client} \bowtie \mathsf{office}).$$

Is it safe to answer this query immediately? Or must the query be delayed until it is certain that the update will not be undone?

The answer is that the query may be executed immediately, because undoing the update would not change the answer to the query. Here is an informal argument.

> "Suppose $\langle P_1, B_1, C_1\rangle$ is a tuple in the answer to the query. Then $C_1 = NewYork$. Now suppose $\langle P_2, B_2\rangle$ is any tuple deleted from the relation client. Because of the join operation that was performed to find which tuples to delete, there must exist a matching tuple $\langle B_2, C_2\rangle$ in the relation office with $C_2 = Los\ Angeles$. Clearly $C_1 \neq C_2$ always. Every broker B has a unique office location C, so $B_1 \neq B_2$. Every person has a unique broker, so $P_1 \neq P_2$. Therefore, if $\langle P_1, B_1, C_1\rangle$ is in the answer to the query, it is present whether or not $\langle P_2, B_2\rangle$ is removed from client."

The algorithm sketched in Section 2.3.2 automates reasoning such as the above.

An example of independence reasoning.

2.4 Towards fully distributed, logical concurrency control

The theory of independence sketched in the previous section solves the difficulties that made predicate locking and its variants impractical. However concurrency control for centralized database systems is no longer an active research area. What is still an important open problem is the design of fully distributed database systems, those where all system functions can be performed at multiple sites.

In a fully distributed system no aspect of data management or transaction execution is permanently centralized at a particular location. In systems that are not fully distributed, some failures at some sites cannot be tolerated. Also, there must be some bottleneck in transaction processing that cannot be alleviated by increasing the number of processing sites. To avoid the so-called wandering bottleneck phenomenon, where the binding constraint on transaction throughput changes as the system is expanded, a fully distributed database system should have a uniform architecture: every site should be capable of performing all system functions. Then it should be possible to increase the processing capacity of the system incrementally just by duplicating processors and software.

System architecture. We discuss here in general terms the design of the database system and scheduler that we propose to implement. Distributed database systems are a classical research area [Bernstein and Goodman, 1981; Bernstein *et al.*, 1987], but most work has focused on systems where the individual processing sites are relatively powerful, and communication is over a point-to-point network. The environment that we are more interested in is a collection of single-user workstations linked by a broadcast network.[2]

The relative cost of different system resources is quite different in such an environment, compared to mainframes sending messages to individual neighbors. In a workstation environment it can be worthwhile to duplicate computation at many sites if communication is thereby avoided. When communication is necessary, it is important to keep messages short, because only one processor can transmit at a time. In a fast point-to-point network, on the other hand, the cost of initiating a message is typically high but the cost of then transmitting a large quantity of data is low.

We propose that each site at which a transaction initiates a query or update should broadcast the text of that request. Using heuristics to accelerate common special cases and to deal with nonconjunctive commands, one or more sites will perform logical reasoning to decide whether the new command conflicts with commands forming part

[2]The obvious example of a broadcast network is an Ethernet. However we anticipate that organizations such as Federal Express will in the future use wireless broadcast networks. Already Federal Express has thousands of handheld computers that read the barcode labels on packages as they pass checkpoints, and then download this information to a central database.

of other transactions. All sites will then execute a distributed agreement protocol to reach common knowledge of whether a conflict exists. The sites concerned by a conflict will then negotiate about resolving the conflict. It should be possible to make intelligent choices about restarting and delaying transactions, given common knowledge of what parts of the database each wants to query and update. Finally, the sites at which relevant data resides (which can also be determined by logical analysis) will cooperate in executing query and update requests.

As far as possible, our system will be built as independent software modules. The scheduler will act as a filter on query and update requests: some will be passed on for execution and some will be blocked. Inside the scheduler, conflict detection and resolution will be separate functions. All reasoning done by the scheduler will concern only the text of queries and updates, so scheduling will require no disk operations and no broadcasting of long messages.

Research starting points. No fully distributed database system of the type that we plan exists at present. At the University of Maryland, Nick Roussopoulos has a prototype implementation of a relational database system running on multiple Suns linked by a standard Ethernet []. In this system, the software to execute query and update commands is fully distributed. However the scheduler of the system is centralized. We intend to use Roussopoulos's algorithms for evaluating queries and performing updates as inspiration in the design of our system.

At Harvard University, Mei Hsu and Va-On Tam have proposed a fault-tolerant protocol for maintaining a directory of access rights [Tam and Hsu, 1990]. They view an access right as a token that permits its holder to perform some operation on a data item. In the computer architecture context access rights are called capabilities. The work of Hsu and Tam is item-oriented, but the correctness of their protocol does not rely on this assumption. It should therefore be possible to use it to maintain common knowledge of access rights to parts of a database designated logically.

The protocol of Hsu and Tam is not fully distributed. They impose a tree topology on the sites that need to agree as to who holds what access rights. Essentially, the directory of access rights is maintained at the root site of this tree. However the protocol is robust in the face of any site failing, and in particular in the face of a failure at the root site. Intuitively, by telling all other nodes that the root is down, the protocol could become fully distributed.

At the University of Texas (Austin) Don Batory has developed a system called GENESIS for generating specialized relational database systems []. The user of GENESIS makes architecture and implementation choices on menus, and then the system automatically assembles appropriate software modules from its library. GENESIS produces single-user centralized database systems running on Apple Macintosh II workstations. We intend to base our software development on GENESIS and we expect to be able to make our work available to others in the form of additions to the GENESIS library.

Anticipated benefits. Subject to the over-riding correctness requirement of preventing interference between transactions, the main aim of a scheduler is to provide high throughput: to allow transaction query and update requests to be executed as soon as possible. A scheduler that maximizes the number of transactions allowed to proceed increases the opportunities for another system component to manage the degree of multiprogramming.

We anticipate that separating conflict detection, conflict resolution, and access command execution will enable our scheduler to maximize throughput. Depending on circumstances, such as the loading of the system and the priority of different request, conflicts can be resolved in various ways. Transactions can be delayed or restarted on a case-by-case basis. For example, one of each pair of conflicting transactions can be queued and its execution resumed later unless the frequency of deadlock is too high, in which case the least advanced conflicting transaction can be chosen as the one to be aborted and restarted after a random wait.

We expect that it will normally be cheaper to check whether a query or update request causes a conflict than to execute the request, because the former requires only computation and some communication, while the latter requires disk transfers. However, our scheduler will be able to switch to optimistic concurrency control if wanted. Checking whether a transaction is involved in a conflict can be delayed until the transaction has to commit because it is terminating or communicating with the outside world.

In addition to flexibility in adapting to changing patterns of database access, doing scheduling by reasoning about queries and updates should allow more efficient execution of these commands. The reason is that the retrieval procedures of a database system often examine more tuples than are necessary to determine the result of a query. This can happen because indices are missing, because selections are not fully propagated, or for other reasons. By default an item-oriented scheduler protects any extra tuples retrieved. However the answer to a query cannot be changed by an update given independence as defined in Section 2.3.1, even if the update does modify tuples that are retrieved while evaluating the query. Therefore the command execution component of our database system will be able to treat all queries and updates passed for execution as if they were requested by one transaction. For example, caches will not have to be flushed between different transactions, and it will be possible to read whole relations into memory once for use by multiple access commands. Reasoning about independence will ensure correctness regardless of how access commands are executed.

2.5 Proposed research

2.5.1 Topics of investigation

As progress is made in research, new questions come to the fore, so it is not possible to provide an exhaustive list of research topics here. Nevertheless, it is important to be precise in identifying issues to investigate. We believe that is feasible to resolve the issues listed below, and that doing so will be sufficient to enable us to build a useful fully distributed database system.

- Data structures for implementing independence inference. The work summarized in Section 2.3.2 concentrated on obtaining asymptotically best-possible algorithms for deciding when conjunctive queries and updates are independent. In practice it may be preferable to do independence reasoning using simpler data structures. More complex structures may involve higher constant factors, despite better asymptotic performance. Also, it may be easier to ensure that simpler algorithms are implemented correctly. Another constraint on the choice of data structures will be that they must be compatible with those already used in the GENESIS system.

- Heuristics for nonconjunctive queries. While the theorems of Section 2.3.1 are general, algorithms developed so far apply only to conjunctive query expressions. Deciding independence for nonconjunctive queries is $\mathcal{NP}$-complete, so we intend to look for heuristics accommodating them. It is also important to deal with queries that are not strictly relational, for example those involving aggregating operators such as sum.[3]

- Selecting distributed agreement algorithms. We intend to compare alternative protocols for attaining common knowledge, and we may choose to implement more than one. Two important questions are whether to use an asynchronous or a synchronous protocol, and whether to attempt to limit the number of sites involved in each conflict detection operation.

- Quantifying tradeoffs between network bandwidth, disk seek and transfer speeds, and processing power. Of these network bandwidth will be the only scarce central

[3] Coping with the complexities of real database query languages will be challenging, but a number of ideas are available. Consider SQL cursors for example. A cursor ranges over the set of tuples defined by a query expression that is not evaluated when the cursor is defined. Instead, further SQL commands advance the cursor and incrementally obtain tuples. If a cursor selects tuples of a relation R ordered by increasing values of an attribute A, then concurrency can be maximized by using nested read set expressions. When the cursor is advanced from $A = c_1$ to $A = c_2$, the read set expression $\sigma_{A \leq c_1}(\mathsf{R})$ can be replaced by $\sigma_{A \leq c_2}(\mathsf{R})$. Since each new expression subsumes previous ones, the previous ones can be discarded immediately without jeopardizing correctness.

resource. We hope to be able to design an eclectic protocol for resolving conflicts based on a quantitative analysis of these costs.

- Performing experiments to measure performance. The first aim of these experiments will be to ensure that the system we build is implemented efficiently, so that conclusions drawn about its performance are not artefacts of low-level design errors. The second aim will be to document that there really are no bottlenecks preventing performance from increasing with the addition of extra processing sites. For these experiments we will try to use the standard TP100 transaction processing benchmark [Anon. *et al.*, 1985].

The first two research topics listed above fall squarely within the area of expertise of the principal investigator. For the other topics, expert advice is available in the UCSD computer science department. The choice of distributed agreement algorithms will draw on the expertise of Michael Saks and Christos Papadimitriou, while the investigation of tradeoffs between different system resources will benefit from interaction with a research project on redundant arrays of inexpensive disks under initiation by Walter Burkhard. Advice on performance measurement will be available from George Polyzos.

2.5.2 Research plan

In the summer of 1991, we intend to study distributed agreement algorithms and the facilities provided by the GENESIS toolkit. We will also contact the designers of all the database systems that we can find using variants of predicate locking. The 1991/92 academic year will be devoted to the detailed design of our new database system, and coding and testing on one workstation. In the summer of 1992 we expect to be able to run in genuine multiuser mode on two workstations.

In 1992/93 we will concentrate our efforts first on measuring the performance of the system. We shall perhaps be led to revise some design choices, and to reimplement some components of the system. In 1993 we will also search for funding to continue our work. We should have a working system running on multiple workstations, so we expect to find a company interested in supporting our work under the University of California "Cal Micro" program of research cooperation with industry. The aim of this future work will probably be the provision of a more sophisticated object-oriented and deductive query language.

2.6 Bibliography

References

[Aho *et al.*, 1979] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, 8(2):218–246, May 1979.

[Anon. *et al.*, 1985] Anon. *et al.* A measure of transaction processing power. *Datamation*, 31(7), 1985. Written by Jim Gray and others.

[Bernstein and Goodman, 1981] Philip A. Bernstein and Nathan Goodman. Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2):185–221, 1981.

[Bernstein *et al.*, 1987] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, 1987.

[Blaauw and Duijvestijn, 1985] G. A. Blaauw and A. J. W. Duijvestijn. View locking. In G. Bucci and G. Valle, editors, *Computing 85: A Broad Perspective of Current Developments*, pages 157–165. Elsevier Science Publishers B.V. (North-Holland), 1985.

[Chamberlin *et al.*, 1981] D. D. Chamberlin *et al.* A history and evaluation of System R. *Communications of the ACM*, 24(10):632–646, October 1981.

[Downey *et al.*, 1980] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, October 1980.

[Elkan, 1989] Charles Elkan. A decision procedure for conjunctive query disjointness. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 134–139, Philadelphia, March 1989.

[Elkan, 1990a] Charles Elkan. *Flexible Concurrency Control by Reasoning about Queries and Updates.* Ph.D. thesis, Cornell University, 1990. Available from University Microfilms International, Ann Arbor, Michigan.

[Elkan, 1990b] Charles Elkan. Independence of logic database queries and updates. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 154–160, Nashville, Tennessee, April 1990.

[Eswaran *et al.*, 1976] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, November 1976.

[Hunt and Rosenkrantz, 1979] Harry B. Hunt and Daniel J. Rosenkrantz. The complexity of testing predicate locks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 127–133, May 1979.

[Jordan *et al.*, 1981] J. R. Jordan, J. Banerjee, and R. B. Batman. Precision locking. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 143–147, May 1981.

[Klug, 1983] Anthony Klug. Locking expressions for increased database concurrency. *Journal of the ACM*, 30(1):36–54, January 1983.

[NSF, 1990] Database systems: Achievements and opportunities. *Computing Research News*, 2(3):18–19, July 1990. Summary of the Report of the NSF Invitational Workshop on Future Directions in DBMS Research.

[Oppen, 1980] D. C. Oppen. Complexity, convexity, and combination of theories. *Theoretical Computer Science*, 12(3):291–302, November 1980.

[Reimer, 1983] M. Reimer. Solving the phantom problem by predicative optimistic concurrency control. In *Proceedings of the International Conference on Very Large Databases*, pages 81–88, September 1983.

[Tam and Hsu, 1990] Va-On Tam and Meichun Hsu. Token transactions: Managing fine-grained migration of data. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 344–356, Nashville, Tennessee, April 1990.

[Ullman, 1988] Jeffrey D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1. Computer Science Press, Rockville, Maryland, 1988.

[Wheeler, 1986] Traci Ann Wheeler. Unification and predicate locking. M.S. thesis, University of California, Los Angeles, 1986. Published as UCLA Technical Report CSD-880023.

3 Impact statement

Technical contributions. Our work will demonstrate a novel approach to the design of fully distributed database systems, which is a long-standing open problem in computer systems research. The research will also be a step towards increasing the intelligence of database systems by applying automated reasoning algorithms. In future systems providing object-oriented and deductive query languages, where individual transactions have long lives, item-oriented concurrency protocols will not know exactly what information must be protected on behalf of each transaction, and they will not be able to resolve conflicts flexibly enough. Concurrency control based on logical analysis of queries and updates, as we shall provide, will be essential.

Societal contributions. Our work will help reduce the distance between industrial database system technology and academic research. We will be drawing lessons from the design of commercial database systems, and the software that we develop will be made available as new components for the GENESIS toolkit, so it will be easy for other researchers to use.

The proposed research is expected to constitute the core of the Ph.D. thesis of Kimberly Claffy, a top graduate student at UCSD. Thus the research proposed here will help increase the population of computer scientists who are women and U.S. citizens. With additional funding from the UCSD Committee on Research, which is expected to be granted, the proposed research will also contribute to the training of undergraduates in computer science.

4 Biographical sketch

CHARLES ELKAN

ACADEMIC POSITIONS

- Assistant Professor, Department of Computer Science & Engineering, University of California, San Diego, since September 1990.
- Research Associate, Knowledge Representation and Reasoning Group, Department of Computer Science, University of Toronto, August 1989 to August 1990.
- Lecturer, University of Pennsylvania, June 1989 and February 1990.
  Co-teacher of short courses for the US Army Research Office.

EDUCATION

- **Cornell University.** Ph.D. in computer science, conferred August 1990. Thesis title *Flexible Concurrency Control by Reasoning about Queries and Updates.* Committee members Prakash Panangaden, Dexter Kozen and (representing economics) George Staller.
- **Stanford University.** Coursework in artificial intelligence and economics, 1986.
- **Fitzwilliam College, Cambridge University.** B.A. (Honours) in mathematics, 1984.

ACADEMIC AWARDS

- Special fellowship, Department of Computer Science, Cornell University, 1987/88.
- Visiting scholarship, Stanford University, 1986.
- Andrew D. White fellowship, Cornell University, 1984/85/86.
- Rouse-Ball travelling studentship awarded by Trinity College (Cambridge), 1984/85.

INDUSTRIAL EXPERIENCE

- Research Intern, Intelligent Systems Laboratory, Xerox Palo Alto Research Center, summer 1988.
- Member, Technical Staff, Mathematics Research Center, ATT Bell Laboratories (Murray Hill), summer 1986.

- Statistical Consultant, Katalco Corporation (Chicago), summer 1983.
- Applications Programmer, Compusoft S.A. (Geneva, Switzerland), summer 1982.

PROFESSIONAL ACTIVITIES

- Program committee member, 1991 National Conference on Artificial Intelligence (AAAI-91), 1990 Canadian Conference on Artificial Intelligence (CSCSI-90).
- Referee for *ACM Transactions on Computer Systems*, *ACM Transactions on Database Systems*, *Annals of Pure and Applied Logic*, *Artificial Intelligence*, *IEEE Transactions on Data and Knowledge Engineering*, and *Information Processing Letters*.
- Reviewer for the ACM Ph.D. dissertation contest, the ACM POPL conference, the IEEE FOCS conference, IJCAI-89, and other conferences.
- National Science Foundation proposal reviewer.

INVITED TALKS

- *Concurrency Control by Reasoning About Transactions: Adaptive Locking.* Stanford University, May 1988.
- *Not the Last Word on Explanation-Based Learning Algorithms.* Annual meeting of the AI Society of New England, Brown University, November 1988.
- *Conspiracy Numbers and Caching in a Prolog-Technology Theorem Prover.* AAAI Symposium on High-Performance Theorem-Proving, Stanford University, March 1989.
- *Automated Inductive Reasoning about Logic Programs.* UCSB, UCLA, UCR, University of Virginia, University of British Columbia, University of Massachusetts (Amherst), University of Toronto, University of Pennsylvania, University of Chicago, March to May 1989.
- *Incremental, Approximate Planning (Preliminary Report).* Stanford University, November 1989. AAAI Symposium on Planning in Uncertain Environments, Stanford University, March 1990.
- *A First-Order Solution to the Frame Problem*, UCSB, UCLA, UCI, UCSD, ATT Bell Laboratories, SRI International, April 1990. USSR Academy of Sciences, Moscow State University, September 1990.

REFEREED PAPERS

Charles Elkan, David Lubinsky, and Daryl Pregibon. Automated descriptions of data. In *Proceedings of the Fifth International Symposium on Data Analysis and Computer Science*, pages 165–170. Versailles, France, September 1987.

Charles Elkan and David McAllester. Automated inductive reasoning about logic programs. In *Proceedings of the Fifth International Conference Symposium on Logic Programming*, pages 876–892. Seattle, August 1988. MIT Press.

Charles Elkan. A decision procedure for conjunctive query disjointness. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 134–139. Philadelphia, March 1989. ACM Press.

Charles Elkan. Conspiracy numbers and caching for searching and/or trees and theorem-proving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 341–346. Detroit, August 1989. Morgan Kaufmann.

Charles Elkan. Logical characterizations of nonmonotonic TMSs. In *Proceedings of the Symposium on Mathematical Foundations of Computer Science*, pages 218–224. Porąbka-Kozubnik, Poland, August 1989. Springer Verlag LNCS 379.

Charles Elkan. Independence of logic database updates and queries. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 154–160. Nashville, Tennessee, April 1990. ACM Press.

Charles Elkan. A Rational Reconstruction of Nonmonotonic Truth Maintenance Systems. *Artificial Intelligence* **43** (1990) 219–234.

Charles Elkan. Incremental, approximate planning. In *Proceedings of the National Conference on Artificial Intelligence* (AAAI-90), pages 145–150, August 1990. Morgan Kaufmann.

Alberto Segre, Charles Elkan, and Alex Russell. A Critical Look at Experimental Evaluations of EBL. *Machine Learning* **6** (1991) 183–195.

TECHNICAL REPORTS

Charles Elkan. Adaptive Locking. Technical Report 87-829, Department of Computer Science, Cornell University. April 1987.

Charles Elkan and Alberto Segre. Not the Last Word on EBL Algorithms. Technical Report 89-1010, Department of Computer Science, Cornell University. May 1989.

Charles Elkan. Incremental, Approximate Planning (Preliminary Report). Technical Report KRR-89-12, Department of Computer Science, University of Toronto. November 1989.

Alberto Segre, Charles Elkan, and Alex Russell. On Valid and Invalid Methodologies for Experimental Evaluations of EBL. Technical Report 90-1126, Department of Computer Science, Cornell University. May 1990.

AREAS OF RESEARCH

My research interests are in database systems and artificial intelligence. In database systems, my previous research and current interests are described in this proposal.

In artificial intelligence I have worked on a number of topics united by the theme of efficient reasoning. I have proposed a first-order solution to the notorious frame problem, and a new planning algorithm which is incremental and approximate: it can find a plausible candidate plan quickly, and it can revise its preliminary plan if necessary when allowed more time. Another topic on which I am working is theorem proving with induction. David McAllester and I have proposed and tested a new method of inventing induction hypotheses in reasoning about logic programs. This method may be useful in independence reasoning about DATALOG queries and updates.

Alberto Segre and I have also done work on explanation-based learning (EBL) algorithms, which increase the speed of a problem-solver by analyzing solutions to example problems. We have developed a family of domain-independent EBL algorithms that subsume all other EBL algorithms. We have also shown that the conclusions of previous experimental studies of EBL algorithms are unreliable, and we have proposed a new methodology that does yield experimental conclusions with predictive power.

CAREER RESEARCH GOALS

Database systems and general theorem provers are extreme points on a spectrum of automated reasoning systems. I want to increase the efficiency of traditional AI problem-solvers, and I want to increase the intelligence of traditional database systems.

5 Budget explanation

We are requesting summer support for the principal investigator and two academic years of support for a research assistant. The research described in this proposal is expected to constitute the core of the Ph.D. thesis of the RA, and we feel it is important to allow that person to work full-time on this project. Given the $60000 budget limit imposed by the NSF, the only way to provide two full academic years of support for an RA is to reduce support for the PI. Accordingly, the budget only provides for the PI at 90% of regular salary.

The budget does not include any funding for equipment, although it will be necessary to purchase two Apple Macintosh IIsi workstations for the research. An equipment grant awarded already by the Powell Foundation will be used for this purpose. The Powell grant will also be used to acquire database system software developed elsewhere: the GENESIS toolkit from the University of Texas, and versions of industrial systems. Technology transfer in both directions between academia and industry is an important aim of our proposed research.

The budget includes a minimal request for travel funding. This will be used for collaboration with other researchers, and for presenting the results of our work at conferences. Our previous work on independence reasoning was presented at the 1989 and 1990 PODS conferences, and an article for the *Journal of the ACM* is in preparation. We expect that the work proposed now will be reported at SIGMOD conferences and in *ACM Transactions on Database Systems*.

We anticipate that the requested funding for a research assistant will be used to support Kimberly Claffy, a Ph.D. candidate in computer science who is one of the top graduate students at UCSD. Kimberly Claffy graduated in 1989 with a B.A. in computer and cognitive science from Stanford University, with honors. At UCSD, she has passed the Ph.D. comprehensive examinations in computer science with distinction. The NSF support requested now will enable her to write an excellent Ph.D. thesis.