Bounded Ignorance: A Technique for Increasing Concurrency in a Replicated System

NARAYANAN KRISHNAKUMAR and ARTHUR J. BERNSTEIN
State University of New York, Stony Brook

Databases are replicated to improve performance and availability. The notion of correctness that has commonly been adopted for concurrent access by transactions to shared, possibly replicated, data is serializability. However, serializability may be impractical in high-performance applications since it imposes too stringent a restriction on concurrency. When serializability is relaxed, the integrity constraints describing the data may be violated. By allowing bounded violations of the integrity constraints, however, we are able to increase the concurrency of transactions that execute in a replicated environment. In this article, we introduce the notion of an *N-ignorant* transaction, which is a transaction that may be ignorant of the results of at most N prior transactions. A system in which all transactions are *N-ignorant* can have an $N + 1$-fold increase in concurrency over serializable systems, at the expense of bounded violations of its integrity constraints. We present algorithms for implementing replicated databases in *N-ignorant* systems. We then provide constructive methods for calculating the reachable states in such systems, given the value of N, so that one may assess the maximum liability that is incurred in allowing constraint violation. Finally, we generalize the notion of *N-ignorance* to a matrix of ignorance for the purpose of higher concurrency.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; *distributed databases*; H2.4 [**Database Management**]: Systems—*concurrency*; *distributed systems*; *transaction processing*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Concurrency control, integrity constraints, reachability analysis, replication, serializability

1. INTRODUCTION

Databases are often replicated to improve performance and availability. Replication reduces the need for expensive remote accesses, thus enhancing performance. Replicated databases can also tolerate failures, hence providing

This work was supported by NSF grants CCR 8701671 and CCR 8901966. A preliminary version of this article appeared in the 10th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1991.

Authors' address: N. Krishnakumar, Bellcore, MRE-2B324, Morristown, NJ 07960; email: nkk@bellcore com.

greater availability than single-site databases. This article describes techniques for increasing concurrency, and hence throughput, among transactions executing in a replicated environment.

The traditional approach to concurrency control ensures serializability [Bernstein et al. 1987]. Serializable execution provides concurrency atomicity (one transaction does not see intermediate states produced by another) and failure atomicity (a transaction either runs to completion, or it has no effect at all on the database). The corresponding notion of correctness for replicated data is *one-copy* serializability, where the effect of the execution of a set of transactions on the replicated data is equivalent to some serial execution of those transactions on a single copy. As seen shortly, recent research has focused on exploring other models of correctness and techniques which yield higher concurrency than that realizable with the above approach.

The original work on concurrency control assumed that a transaction is a sequence of read and write operations on records. In classical serializability theory, two operations on a data item conflict unless they are both reads. Since the operations of two transactions cannot be interleaved arbitrarily if they conflict, such transactions cannot execute concurrently. One technique for improving concurrency views the database as a collection of abstract data objects and utilizes *the semantics of the abstract operations from which transactions are constructed* (as opposed to the conventional classification of operations as reads or writes on records). With this semantic view, we can restrict the notion of conflict. For instance, two commutative operations do not conflict, even if both update the same data item. Such properties have been formalized and the corresponding concurrency controls developed in Herlihy [1987; 1990], Herlihy and Weihl [1988], and Weihl [1988; 1989]. However, whether or not two operations commute might depend on the state of the database. Consider an airline reservation system with a single flight, and let res_seats denote the number of seats that have been reserved on the flight. Suppose the plane can carry at most 200 passengers, so that the integrity constraint of the system is res_seats $\leq$ 200. Two reserve operations that are each trying to reserve one seat do not commute when res_seats is 199, though they do commute when less than 199 seats have been reserved. More elaborate models (for instance, the escrow transaction idea [O'Neill 1986]) are needed to exploit the inherent concurrency in such a situation.

Several researchers have considered weakening serializability. In Garcia-Molina [1983], nonserializable schedules are allowed that preserve consistency in the sense that the integrity constraints in the system are maintained. Transactions are partitioned into disjoint subsets called types. Each transaction is a sequence of atomic steps. Each type, y, is associated with a compatibility set whose elements are the types, y', such that the atomic steps of y and y' may be interleaved arbitrarily without violating consistency. Interleaved schedules are not necessarily serializable. This idea has been generalized in Lynch [1983] and Farrag and Özsu [1990] in different ways to give a specification of allowable interleavings at each transaction breakpoint with respect to other transactions. In Fischer and Michael [1982] and Wuu and Bernstein [1984], the problem of a replicated distributed

dictionary is discussed, and nonserializable schedules are allowed so as to increase concurrency. The Grapevine system [Birrell et al. 1982] allows a naming service that does not act in a serializable fashion. Recently, built on some of these ideas, several advanced transaction models and correctness criteria have been proposed to overcome the limitations of serializability in special applications. For instance Korth et al. [1988] looks at CAD environment which frequently involves long-running transactions and cooperating tasks. Enforcing serializable behavior in such a situation implies that data items will be inaccessible for long periods of time, an unacceptable condition. The authors use a nested transaction model in which serializability may not be preserved, and in Korth and Speegle [1988] define several criteria under which nonserializable executions are termed correct. New models have also been developed to deal with the management of data in multiple autonomous databases (*federated* or *heterogeneous* databases). Among the models proposed for such applications are the ConTract Model [Reuter and Wachter 1991], relaxed atomicity [Levy et al. 1991], nested sagas [Garcia-Molina et al. 1991], multilevel transactions [Weikum and Schek 1991], polytransactions [Rusinkiewicz and Sheth 1991], and other approaches found in Elmagarmid [1991].

Most of the above approaches are concerned with maintaining the integrity constraints of the database, even though serializability is not preserved. In this article, we are interested in applications that have stringent performance requirements but which can permit bounded violations of the integrity constraints. We propose the notion of *N-ignorance* that utilizes this relaxation of correctness in order to increase concurrency. In an *N-ignorant* system, a transaction need not see the results of at most N prior transactions that it would have seen if the execution had been serial. Therefore, concurrency can be increased $N + 1$-fold with respect to a serializable system. For instance, consider the airline reservation example, and assume that each transaction is allowed to reserve at most one seat. A serializable execution would allow at most a single transaction to execute at a time in the system since all transactions modify the same data item, namely, res_seats. Suppose a maximum overbooking of 10 passengers is considered acceptable. Then, even when res_seats is 199, 11 transactions can be allowed to execute concurrently at different sites without each being aware of the other transactions. *N-ignorance* uses this observation to provide a 11-fold increase in concurrency. A related approach is taken in SHARD [Lynch et al. 1986; Sarin 1986; Sarin et al. 1988], which is a replicated database system in which deviations from the integrity constraints are also permitted. SHARD, however, makes no attempt to place a fixed bound on the size of those deviations, relying instead on the small probability of large deviations. More recently, other researchers have proposed models [Pu and Leff 1991; Sheth et al. 1991; Wong and Agrawal 1992] which deal with boundedly inconsistent databases. This article is distinct from these papers in two respects:

(1) We formalize the violations of the integrity constraints as a function of the number, N, of conflicting transactions that can execute concurrently.

To compute the extent to which the constraints can be violated, we also provide a systematic analysis of the reachable states of the system.

(2) Since we deal with a replicated database, the emphasis is on reducing the number of sites a site has to communicate necessarily with *after* a transaction is submitted for execution (for instance, majority quorum consensus requires that at least a majority of the sites be consulted). We thus attempt to increase the "autonomy" of sites in executing transactions, and this results in smaller response times.

N-ignorance results thereby in an increase in both concurrency and autonomy, and this results in increased throughput.

The article is organized in the following fashion. In Section 2, we outline the system model. A discussion of the motivation for our approach and a review of closely related work is presented in Section 3. A suite of algorithms for implementing *N-ignorance* is provided in Section 4. An analysis of *N-ignorant* systems is developed in Section 5. We discuss briefly in Section 6 the generalization of *N-ignorance* to a *matrix* of ignorance. We conclude with a discussion of future work in Section 7. The appendices have proofs of some results that are stated in the body of the article.

2. THE MODEL

We assume a fully replicated database. Let the set of site identifiers be denoted by $\mathcal{SI}$, and let each site have a unique site identifier. The state of a database is characterized by an assignment of values to the data items in the database. We refer to the state of the database at a site as the *site view*, and it need not necessarily be up to date: it may be missing the updates of (a limited number of) transactions which have executed at other sites.

A set of integrity constraints, $\mathcal{C}$, is specified for the system. A particular integrity constraint, C (in $\mathcal{C}$), is a function which maps a state, s, to *true* or *false* depending on whether or not the constraint is satisfied in that state. It is assumed that all serializable executions of transactions starting from an initial state, s_0, such that $\forall C \in \mathcal{C}\,[C(s_0) = true]$ preserve these constraints. A transaction, T, is executed atomically at a single site as follows in the following three steps [Sarin 1986]. We say that T has been *submitted* at site i when the request to execute T at site i is made (execution might, however, be delayed).

(1) The read phase of T, RP_T, occurs first. The site view, l, is read, and an update, u_T, for transforming the database is generated. We use the notation $RP_T(l) = u_T$ to indicate that u_T is generated when the read phase sees the site view l. Let $apply(u_T, l)$ denote the state resulting from applying u_T to l. Then, it is either the case that $\forall C \in \mathcal{C}\,[C(apply(u_T, l)) = \text{true}]$ (i.e., the resultant state satisfies the constraints), or u_T is the null update. We say that T has been *initiated* when the execution of RP_T has begun.

(2) The database at i is updated using u_T.

(3) A request to broadcast u_T to all other sites is made.

Note that the execution of T does not cause the site view to violate $\mathscr{C}$. Violations might occur subsequently when a site learns of updates it was ignorant of when T executed. Compensating transactions [Garcia-Molina 1983; Korth et al. 1990; Sarin 1986] can be used to restore the database to a state satisfying $\mathscr{C}$.

For example, consider the airline reservation system. $\mathscr{C}$ is {res_seats $\leq$ 200}. When a transaction T reserving a single seat is executed at site i, RP_T produces the update [res_seats := res_seats + 1] if res_seats < 200 in i's site view and the null update otherwise.

We assume that a state is represented as a history—a sequence of updates. (This is only for convenience. It is not required that the entire past history be maintained at a site. It is in fact sufficient to use a compacted version of the history, but the details are beyond the scope of this article.) We say that a transaction T (or RP_T) *sees* a particular update if the update is in the history corresponding to the site view at the time that RP_T is executed. We define a happened-before [Lamport 1978] ordering, $\rightarrow$, between transactions T and T' as follows: $T \rightarrow T'$ if and only if T' sees u_T. A transaction T is said to be *concurrent* with T', denoted T conc T' if and only if $T \not\rightarrow T'$ and $T' \not\rightarrow T$. We assume that the broadcast mechanism for disseminating updates satisfies the property that sites learn of updates which have been generated at other sites in accordance with $\rightarrow$. Thus if a site knows of an update u_T that was produced by a transaction T, then it knows also of all updates generated by transaction T' such that $T' \rightarrow T$.

A run, R, is the execution of a set of transactions and is represented by a quadruple $[\mathscr{T}_R, \rightarrow_R, <_R, g_0]$ where:

(1) $\mathscr{T}_R$ is the set of transactions that execute in R.
(2) $\rightarrow_R$ is the happened-before ordering between the transactions.
(3) $<_R$ is a total ordering of the transactions consistent with $\rightarrow_R$.
(4) g_0 is the initial site view at all sites. g_0 is also referred to as the *initial state* of R.

The ordering relation $<_R$ is constructed using timestamps. Thus, if $TS(T)$ is the timestamp of transaction T, then $T_1 <_R T_2$ if and only if $TS(T_1) < TS(T_2)$ and $T_1 \rightarrow_R T_2$ implies $TS(T_1) < TS(T_2)$. Updates might arrive at different sites in different orders (but satisfying $\rightarrow_R$). However, we assume that all sites start with the same initial state, and if the same set of updates arrives at two sites, they will yield the same site view. This can be accomplished by assuming that the state of a site is the result of executing in timestamp order the updates it has seen (perhaps implemented using an undo/redo technique as in Jefferson [1985] and Sarin et al. [1986]).

The *global state* of the database at any time t is the state that includes all updates (in the order $<_R$) that have occurred at all sites until time t. Even though the read phase of a transaction enforces $\mathscr{C}$ on the local site view, the global state may not satisfy $\mathscr{C}$. For instance, suppose there are two sites in the airline reservation system. Assume that both sites have the same site view in which res_seats is 199. Let each site execute a transaction reserving

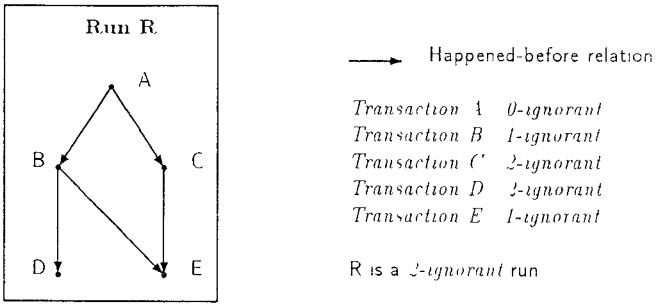

Fig. 1. A 2-ignorant run.

a single seat. If the execution is serializable, one of the transactions sees the effect of the other. However, we allow the two transactions to execute concurrently. Thus each transaction produces an update that increments res_seats in its site view, and the resulting global state has 201 reserved seats. Our goal is to limit this deviation so that all global states satisfy a weaker set of constraints, $\mathscr{C}'$. For instance, $\mathscr{C}'$ can be res_seats ≤ 210.

Informally, a *replica/concurrency control algorithm* is required to coordinate concurrent access to the replicas by several transactions and ensure that the system does not behave "incorrectly." In our case, the control algorithm must place a bound on how out of date the site view is allowed to be when a transaction is initiated. The read phase is delayed when this bound is exceeded. This limits the global states which can occur in the system and, hence, limits the extent to which the integrity constraints can be violated. A site view seen by a transaction is thus restricted by the control algorithm.

Consider a system with replica/concurrency control algorithm *RCC*. If the run R, denoted by $[\mathscr{W}_R, \rightarrow_R, <_R, g_o]$, can be produced under *RCC*, we say that R is an *RCC-run*. A state s is said to be *reachable* under *RCC* if there exists an *RCC*-run R such that the global state of the system after executing R is s. We say that *RCC* is *correct* with respect to $\mathscr{C}'$ if $\mathscr{C}'$ is preserved in all reachable global states, given that the initial state of the system satisfies $\mathscr{C}$. Thus the notion of correctness in our model is not serializability, but that of satisfying the relaxed constraints. To ensure that $\mathscr{C}'$ is preserved globally, we limit concurrency between transactions using the notion of *N-ignorance*.

Consider a run R and a transaction T in $\mathscr{W}_R$. If the number of transactions concurrent with T in R is not greater than N, we say that T is *N-ignorant* in R. If all transactions in $\mathscr{W}_R$ are *N-ignorant* in R, we say that R is an *N-ignorant* run. An *N-ignorant system* is one in which only *N-ignorant* runs are permitted.

For example, consider the run R in Figure 1. R is a 2-ignorant run since all transactions are 2-ignorant. Let z be the initial state. If $\bullet$ is the concatenation operator, and the total order on the transactions is $A <_R B <_R C <_R D <_R E$, the state $z \bullet \langle u_A, u_B, u_C, u_D, u_E \rangle$ is reachable in the given system.

We say that two transactions T_1 and T_2 *conflict* if there exists a state s such that (a) s satisfies $\mathscr{C}$, (b) RP_{T_1} reads s and produces update u_{T_1}, and

RP_{T_2} reads s and produces update u_{T_2}, and (c) either of the states $s \bullet \langle u_{T_1} \rangle \bullet \langle u_{T_2} \rangle$ or $s \bullet \langle u_{T_2} \rangle \bullet \langle u_{T_1} \rangle$ does not satisfy $\mathscr{C}$. We assume initially that all transactions conflict potentially. This is a worst-case assumption which we relax in Section 6 by taking into account transaction (or operation) semantics [Herlihy 1987].

3. MOTIVATION AND RELATED WORK

Consider the airline reservation system, with the constraint $\mathscr{C}$: {res_seats $\leq$ 200} and a transaction that reserves at most one seat. If a maximum overbooking of 10 passengers is considered acceptable, only the weakened constraint set, $\mathscr{C}'$: {res_seats $\leq$ 210}, needs to be enforced globally (each transaction enforces $\mathscr{C}$ on its view of the database). This allows a transaction executing at a replica site to be unaware of 10 concurrently executing transactions at other sites. Thus, by relaxing the constraint, 11 transactions can now run concurrently in the system, as against just one in the serializable case. The algorithms described in Section 4 guarantee that a transaction is no more than *N-ignorant* and hence that $\mathscr{C}'$ always holds.

The notion of *k-completeness*, which is similar to *N-ignorance*, was introduced in the context of the SHARD system [Lynch et al. 1986]. Both a total order is imposed on transactions executed in a run and a transaction is k-complete if it sees the results of all but at most k of the preceding transactions in that order. No algorithm is presented in Lynch et al. for enforcing a particular value of k, and hence, the extent of ignorance can only be estimated in a probabilistic sense. Furthermore, a transaction which has executed at one site may be unknown to other sites even after they have executed a large number of subsequent transactions. This is a major drawback since it implies that a particular site might remain indefinitely unaware of an earlier transaction at another site.

Suppose there are two kinds of transactions in the airline reservation system: a reserve transaction that reserves a single seat on the flight and a seat assignment transaction that assigns a passenger with a reservation to a particular seat. Although passenger P may have made the first reservation on the flight, the reservation might not be seen by a *k-complete* seat assignment transaction executing at another site after the plane becomes full. This may be a problem if the flight is overbooked since P may not even be considered for a seat. With *N-ignorance*, however, only recent reservations might be denied seats due to the overbooking. This is formalized as the locality property in Section 5.1.1.

To motivate the role of constraint violation, consider two other approaches to the airline reservation example:

(1) Though the plane accommodates 200 people, we can require that each transaction enforce on its site view the constraint set {res_seats $\leq$ 190}. In a 10-ignorant system, the weakened constraint set that is then enforced globally is {res_seats $\leq$ 200}. As a result, the system has the desirable property of never overbooking the flight. However, there is a

problem if 190 seats are booked globally, since an up-to-date site will reject reservations for the remaining 10 seats.

(2) Assume that a transaction can be concurrent with at most 10 other transactions when the number of reserved seats is small. As the seat count approaches 200, the degree of concurrency is gradually reduced, so that the total number of seats reserved globally converges to 200. Although this approach combines enhanced concurrency with strict enforcement of the original constraint, the replica control algorithm is more complex since it is state dependent. This (escrow-based) approach is described in Krishnakumar and Bernstein [1992].

Bounded constraint violation may be acceptable in applications in which transactions make incremental changes to the database (e.g., reserve a single seat) [Paige 1990; Qian and Wiederhold 1986]. A number of applications exhibit this behavior. Consider a queue which is manipulated using enqueue and dequeue transactions (which deal with one element). The queue's behavior is FIFO if elements are dequeued in the same order as they were enqueued, and no element is dequeued multiple times. We can apply *N-ignorance* to this data structure to give relaxed queues [Herlihy and Wing 1987]:

(1) The dequeue operation dequeues one of the first N elements from the queue, rather than the first element. This has been referred to as a semiqueue in the literature, with the difference that the "out-of-orderness" is bounded.

(2) The same element may be dequeued up to N times. This is a bounded stuttering queue.

In distributed problem solving [Durfee et al. 1987], used in applications such as distributed robot systems, each node may not have a completely up-to-date global picture. Nodes cooperate by generating and exchanging tentative partial solutions. Other examples include process control systems where continuously varying quantities are monitored, and name-servers in distributed operating systems where there may be at most a few out-of-date entries [Birrell et al. 1982]. A replicated relational database is another example. Consider a database with Manager and Employee relations. Suppose there is a constraint that any manager earns more than any employee. By relaxing this constraint to allow at most k exceptions (i.e., any manager earns more than all but at most k employees), a $k + 1$-fold increase in the concurrent execution of insert operations can be achieved.

The notion of setwise serializability is introduced in Sha et al. [1988]. The database is decomposed into atomic data sets, and serializability is guaranteed within a single data set. Sha et al. discuss weakening constraints but does not place any bound on the deviation. In Herman [1983] and Verjus [1983], the problem of a set of processes which can concurrently allocate R resources is considered. A particular process may be ignorant of allocations made by others, but the constraint that no more than R resources can be allocated at any time must be maintained. The algorithm ensures *k-complete-*

ness for a fixed value of k but does not place a bound on how long a particular allocation occurring at one site can remain invisible at another and, thus, is not useful in a generalized transaction environment.

Applications where replicas are allowed to diverge in a controlled fashion are discussed in Alonso et al. [1988]. In this model, however, an up-to-date central copy is present which is used to detect divergences in other copies (referred to as "quasi-copies") and triggers appropriate action. The authors do not discuss how one can extend this idea to a completely distributed environment. Barbara and Garcia-Molina [1992] and Kim et al. [1989] discuss how simple numerical relationships between data items stored at different sites may be maintained, while allowing site autonomy.

In Rusinkiewicz et al. [1991], an approach has been proposed for specifying the dependency and consistency requirements for *interdependent data* (of which replicated data is a subcase). Algorithms for maintaining the consistency of interdependent data are given in Sheth et al. [1991]. Pu and Leff [1991] have proposed a framework called *ϵ-serializability*, which parallels closely the idea of *N-ignorance*. An *ϵ-transaction* is allowed to *import* and *export* a limited amount of "inconsistency," and this entails, as in our case, an increase in concurrency by allowing bounded inconsistencies in the database. For instance, a query transaction, like *Check_balance* in a banking system, can import an inconsistency of $100 (i.e., the balance it reports can differ by at most $100 with the real balance). Pu and Leff deal mainly with query transactions (which only import inconsistency) that can return incorrect results, and update transactions (which only export inconsistency) which are serializable with respect to other update transactions. It is not clear how the global integrity constraints of the database are affected if transactions can both import and export inconsistency. Furthermore, in both Pu and Leff and Sheth et al. [1991], one approach to bounding inconsistencies rests on having a lock manager that sees all the conflicting transactions and grants locks to a bounded number of them. Thus there needs to be a site that is aware of all conflicting transactions executing in the system. In our algorithms, however, we stress the increased autonomy of the sites by requiring less synchronization between them—a lock manager need not even know that a conflicting transaction is executing elsewhere (as in the state-based algorithms discussed in Barbara and Garcia-Molina [1992] and Krishnakumar and Bernstein [1992]). Wong and Agrawal [1992] extend the notion of ϵ-serializability in the read/write model to the framework of high-level operations on single-copy abstract data types.

The background mechanism for broadcasting updates causally in this article is adapted from Wuu and Bernstein [1984]. Each site i maintains a lower bound on how up-to-date each other site j is. This information indicates to i the updates that i is sure j knows about, and is maintained as an $O(M^2)$ data structure called the *timetable* (where M is the total number of sites). Improvements and optimizations have been proposed in Wu and Bernstein and other subsequent papers that deal with gossip messages, such as the two-phase gossip techniques of Agrawal and Malpani [1991] and Heddaya et al. [1989], the lazy replication mechanism from Ladin et al. [1990], and

weak-consistency replication [Golding 1992]. The primary objective in all these improved protocols is to reduce the space required at each site for maintaining the lower-bound information, and the space required in each message to transport this information. We do not discuss the optimized versions in this article, and adopt the original $O(M^2)$ data structure. It should be noted that any of the optimized versions of the gossip message algorithm or other mechanisms such as causal broadcast (of ISIS [Birman et al. 1991]) can be used in our algorithms with minor changes. Our concern is not with the replica control algorithm but the flexible integration of the concurrency control mechanism (such as quorum locking) and the replica control mechanism to ensure *N-ignorance*.

In situations where violations can occur, it must be possible to apply compensating transactions [Korth et al. 1990; Sarin 1986] to bring the database closer to a state in which the integrity constraints are not violated (e.g., bump a passenger in the airline example). We do not address the issue of compensating transactions in this article, and the reader is referred to Krishnakumar [1992].

4. IMPLEMENTATIONS OF *N*-IGNORANCE

One of the mechanisms used by traditional concurrency control algorithms to synchronize transactions is *locking* [Eswaran et al. 1976]. Quorum locking [Herlihy 1987] is an extension of locking to a replicated system, which by limiting the amount of concurrency at any time ensures that a transaction sees the results of conflicting transactions which have already completed. In the algorithms described here, we use (a) quorum locking to control the number of transactions which can be simultaneously accessing replicas of the same data object and (b) gossip messages to limit the extent to which a site may be ignorant of the effects of prior transactions done at nonquorum sites.

We present a simplified description of quorum locking [Herlihy 1987]. Each site, i, before initiating a transaction T, locks the data items accessed by T at a quorum of sites, Q_T (the size of any quorum is denoted by $|Q|$). The quorum sites send their local site views along with the lock grant response to i. On receiving the lock grant responses, site i merges the site views it has obtained from the quorum sites with its own view, l_i, to get a new view, l_i' and determines the update, u_T on the basis of l_i'. u_T is then appended to l_i' to get l_i''. l_i'' is sent to the quorum sites to be merged into their local site views, and the lock at each quorum site is released. In a serializable execution, transactions conflicting with T should not be allowed to execute concurrent to T. This is guaranteed by ensuring that a site cannot simultaneously hold locks on the behalf of two conflicting transactions, and that the quorums of conflicting transactions intersect. (Of course, there can be transactions that do not conflict with T. For instance, a transaction T' that accesses a set of data items that are disjoint from those accessed by T does not conflict with T. In that case, a site can simultaneously hold a lock for both T and T'.) Suppose we assume that any two transactions conflict. It can be ensured that the quorums of two transactions intersect by taking $|Q|$ to be larger than

$\lfloor M/2 \rfloor$, where $M = |\mathcal{SI}|$ and $\lfloor M/2 \rfloor$ denote the largest integer smaller than or equal to $M/2$.

Gossip messages are point-to-point background messages which can be used to broadcast information. In our algorithms, a gossip message sent by site i to site j contains updates known to i (these need not be only updates of transactions initiated at site i). Gossip messages ensure that a site learns of updates in the happened-before order.

In Wuu and Bernstein [1984], a data structure called a *timetable* is used at each site to maintain information about the state of another site's knowledge. We use this data structure for the same purpose. The timetable at site i, TT_i, can be characterized as follows:

(1) $TT_i[i, i]$ is a counter incremented whenever a transaction T executes RP_T at site i.

(2) If $TT_i[j, k] = x$, then site i knows that site j knows of all transactions initiated at site k when $TT_k[k, k]$ was less than or equal to x.

We denote the kth row of TT_i as $TT_i[k]$, and the timetable at a site j at (global) time t as TT_j^t (note that we use TT_j as a variable, but TT_j^t is a constant). We assume for simplicity that at any site i, at most one of the following can occur at a time t: the execution of a transaction, the send of a gossip message, or the receipt of a gossip message. When transaction T is initiated at site i, the following actions are performed:

(A1) $TT_i[i, i] := TT_i[i, i] + 1$.

(A2) $TT_i[i]$ is assigned as the vector timestamp $TS(T)$, for T (as in Liskov et al. [1988]).

We denote the jth element of the timestamp $TS(T)$ as $TS(T)[j]$.

If site j sends a gossip message m at time t to site i, then m contains TT_j^t, and all the updates that j has seen up to time t. We refer to this as the *gossip property*. This property ensures that updates are delivered at sites in happened-before order, since if a gossip message contains an update u_{T_1}, it also contains every other update u_T such that $T \rightarrow T_1$. Note that we may optimize the amount of information [Wuu and Bernstein 1984] sent in a message by (1) using the timetable to reason about the knowledge state of the destination site (i.e., if j knows that i knows of some updates, then it need not include those updates in its gossip message to i) and (2) sending only a portion of the timetable. In a system with many sites and in which a large number of updates occur, only the optimized approach is feasible. We do not discuss such optimizations in this article.

Assume that a gossip message m is sent from site j at time t and arrives at site i at time t'. Site i merges the update information into its site view on the basis of the timestamps. TT_i is updated in the following fashion:

(M1) $\forall p \in \mathcal{SI}$ **do** $TT_i[i, p] := \max(TT_i[i, p], TT_j^t[j, p])$. This indicates that for each site p, site i knows the updates of all the transactions initiated at p that site j knew about (when j sent the gossip message).

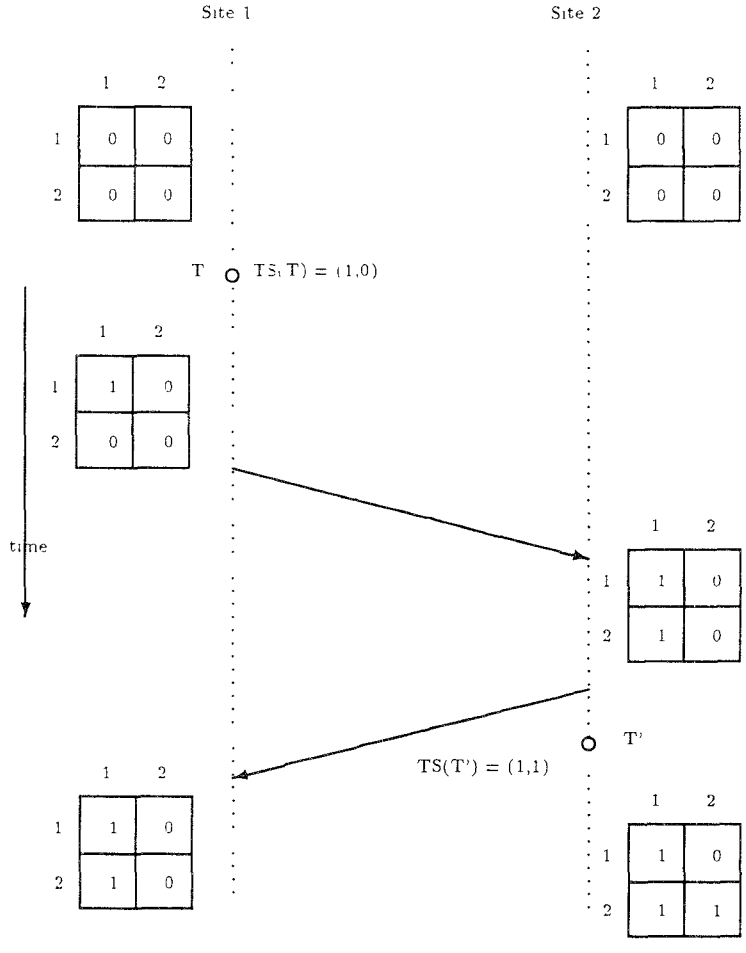

Fig. 2. Timetables and gossip messages.

(M2) $(\forall p \in \mathcal{SS})(\forall n \in \mathcal{SS})$ **do** $TT_i[p, n] := \max(TT_i[p, n], TT_j^t[p, n])$. This indicates that the transactions that j knew were initiated at n and have been seen at p (when j sent the gossip message) are also known by i to have been seen at p. Thus, i is up to date, concerning the knowledge states at the other sites.

In Figure 2, we have illustrated with a simple two-site example how timetables are manipulated. T and T' are two transactions that are executed at site 1 and 2 respectively. The two messages shown are gossip messages, and we have shown the timetable at each site at each (interesting) point in time.

4.1 Properties of the Gossip Message Scheme

The following properties of gossip messages are needed to show the correctness of the algorithms in Section 4.2. The proofs of the lemmas given below are provided in Appendix A.

Recall that the vector $TT_i[k]$ denotes the kth row of the timetable TT_i. Consider two vectors, a and b. If a is elementwise less than or equal to b, then we say $a \preccurlyeq b$.

Observation 4.1.1. For $t \leq t', \forall j \in \mathcal{SS}\, \forall i \in \mathcal{SS}\, [TT_j^t[i] \preccurlyeq TT_j^{t'}[i]]$.

This observation follows from A1, M1, and M2.

Observation 4.1.2. If a gossip message is sent from site j at time t and received and merged at site i by time t', then $TT_j^t[j] \preccurlyeq TT_i^{t'}[i]$.

This observations follows from M1.

LEMMA 4.1.3. *For any t, and for any site i, $\forall j \in \mathcal{SS}\, [TT_j^t[i] \preccurlyeq TT_i^t[i]]$.*

LEMMA 4.1.4. *Consider distinct sites i and j. Assume that $TT_i[i, s]$, for some s, is changed at time t due to either a transaction execution at i or the receipt of a gossip message at i. Then for all $t' > t$, $TT_i^t[i, s] \leq TT_j^{t'}[i, s]$ if and only if there exists a chain of gossip messages from i to j such that the first message is sent from i after t and the last message is received at j before or at t'.*

LEMMA 4.1.5. *Consider a site i and a transaction T. For any t, $TS(T) \preccurlyeq TT_i^t[i]$ if and only if u_T is in the site view of i at t.*

LEMMA 4.1.6. *Consider sites i and r and a transaction T. For any t, if $TS(T) \preccurlyeq TT_r^t[i]$, then u_T is in the site view of i at t.*

LEMMA 4.1.7. *If $T_2 \rightarrow T_1$, then $TS(T_2) \preccurlyeq TS(T_1)$. Further, if T_1 is initiated at site i, then $TS(T_2)[i] < TS(T_1)[i]$.*

LEMMA 4.1.8. *The timestamps assigned to transactions are globally unique.*

Since timestamps are globally unique, if we treat the vector of numbers as the single number obtained by concatenating the elements of the vector, then it follows from Lemma 4.1.7 that if $T_2 \rightarrow T_1$ then $TS(T_2) < TS(T_1)$. The $<$ order on the timestamps is taken as the $<_R$ order in any run R.

4.2 Algorithms for *N*-Ignorance

We integrate quorum locking and gossip messages to obtain a control algorithm for an N-ignorant system. The messages that are propagated in the system as part of the quorum algorithm are: (a) lock request, (b) lock grant, (c) lock release, and (d) lock deny. Gossip messages are piggy-backed onto these quorum messages. Gossip messages may also be transmitted periodically from a site, independent of the quorum messages. We make a liveness assumption that there is sufficient message traffic such that given an update u_T, each site will eventually learn of u_T. Furthermore, we assume that a site modifies the timetable and installs the updates that it receives on a gossip message in a single atomic step.

We assume for now that all transactions potentially conflict, so that a transaction can be ignorant of at most N other transactions in the system. Suppose site i is trying to initiate transaction T. We assume then that site i

does not try simultaneously to initiate another transaction T' (since T' conflicts potentially with T). Observe, however, that any number of transactions that access disjoint data sets should be allowed to run concurrently. We discuss the simple extension of the algorithms to this case in Section 4.3, and a description of the more elaborate extension for matrices of ignorance is given in Section 6.

Consider a transaction T submitted at site i. If site j is in Q_T, then we say that j *participates* in T. Once Q_T is established, the sites in Q_T cannot participate in another transaction until the execution of T is completed. However, $|Q|$ need not be greater than $\lfloor M/2 \rfloor$, so that (conflicting) transactions may execute concurrently. When T is initiated, site i knows of all updates known to each quorum site before the quorum site sent its lock grant response to i. When T terminates, each quorum site learns of all updates that i knows about. Hence, a tight coupling exists among these sites. The condition for nonquorum sites is more relaxed: site i is allowed to initiate T as long as nonquorum sites are ignorant of a bounded number of updates that i knows about (this is verified by waiting for a *timetable check* to be satisfied, until which the initiation is delayed). By adjusting $|Q|$ and the amount of allowed difference between the knowledge states of the initiator site and nonquorum sites, we get algorithms with different values of N. If communication delays are not uniform among sites, then it is reasonable for a site to select nearby sites to be elements of its quorum and allow a knowledge lag to exist with respect to distant sites. Thus, in the algorithms that we present below, site i must lock (less than a majority of) quorum replicas and evaluate a timetable check before it can initiate T. If gossip messages are frequent enough, it can be expected that the knowledge state of distant sites does not differ significantly from i's knowledge state. Hence it is likely that the timetable check will already be true when T is submitted, so that the initiation of T is not delayed.

4.2.1 *Algorithm A.* Assume that transaction T is to be initiated at site i. δ is an integer satisfying $\delta \geq 1$ and is an upper bound on the number of transactions initiated by site i that any other site k does not know about when T is initiated (i.e., since $TT_i[i,i]$ indicates the number of transactions initiated by $i, TT_i[i,i] - TT_i[k,i]$ should be at most $\delta - 1$ before T is initiated). Site i executes the following steps:

1. Wait until $\forall k \in \mathcal{SS}\ [(TT_i[i,i] - TT_i[k,i]) \leq \delta - 1]$.
2. Lock replicas at a quorum, Q_T, of sites.
   (After this step, i knows of all updates known to sites in Q_T at the time each sent its lock grant message).
3. Execute RP_T on i's site view, create $TS(T)$, and append u_T to the view.
4. Unlock quorum sites.

Since site i cannot initiate another transaction while it is trying to initiate T, $TT_i[i,i]$ is unchanged while i is waiting in Step 1. From Observation 1, $TT_i[k,i]$ never decreases. In fact, it increases with the arrival of gossip messages at i. Thus, from the liveness assumptions, the timetable check in Step 1 will be satisfied eventually.

THEOREM 4.2.1.1. *Consider an a-run* $R = [\mathscr{T}_R, \rightarrow_R, <_R, g_o]$ *with given* δ *and* $|Q|$. *Then the maximum number of transactions which precede a transaction* T *in* $<_R$ *and which are not in the site view seen by* RP_T *is* $N = \delta * (M - |Q|)$.

PROOF. Assume that T is to be initiated at site i. No transaction (involving the locked data items) can be initiated at any site, $k \in Q_T$, between the time the lock grant message is sent from k to i and the time the lock release message is received by k. Thus all transactions, T', initiated at k satisfy $T' \rightarrow_R T$ or $T \rightarrow_R T'$. Since T can only be ignorant of concurrent transactions, it follows that transactions concurrent to T must be initiated at nonquorum sites. We now argue by contradiction. Suppose that T is concurrent to at least $\delta + 1$ transactions initiated at a site j, where $j \notin Q_T$. Let T' and T'' be the first and the last of those transactions, respectively. From Step 1 it follows that T'' could not have been initiated until site j knew that site i knew of $u_{T'}$. Site i may learn of $u_{T'}$ at two points.

(1) Before RP_T is executed: in this case $T' \rightarrow_R T$, which is a contradiction, since by assumption $T' \mathsf{conc}_R T$.
(2) After RP_T has completed: let i know of $u_{T'}$ by time t, after RP_T has completed. From Lemma 4.1.5, $TS(T') \preccurlyeq TT_i^t[i]$. Suppose site j is trying to initiate T''. After Step 1 in the algorithm, $TT_i^t[i,j] \leq TT_j[i,j]$. Then from Lemma 4.1.4 and the gossip property, $T \rightarrow_R T''$, which is again a contradiction, since we assume $T'' \mathsf{conc}_R T$.

Thus, T is concurrent to at most δ transactions initiated at each nonquorum site. Since all these transactions might have smaller timestampes than T, the result follows. □

4.2.2 *Algorithm B.* Algorithm A does the timetable check first and then locks a quorum. Algorithm B does the operations in reverse order. Unlike Algorithm A, replicas at quorum sites are locked while a site is waiting for a more restrictive timetable condition to be satisfied. Hence, less concurrency is permitted. This results in smaller values of N.

The algorithm used by site i when executing T follows. In this case, δ is an integer satisfying $\delta \geq 1$ and is an upper bound on the number of transactions *known* to site i that another site, j, does not know about when T is initiated (instead of the number of transactions *initiated* at site i as in Algorithm A).

1. Lock replicas at a quorum, Q_T, of sites.
   (After this step, i knows of all updates known to sites in Q_T at the time each sent its lock grant message).
2. Choose a time t after Step 1. Let $<'_R$ denote the total order on updates seen by i until t, and let $\mathscr{T}$ denote the transactions in $<'_R$ except the last $\delta - 1$ transactions.
   Wait until $(\forall k \in (\mathscr{S}\mathscr{T} - Q_T))\ (\forall T' \in \mathscr{T})\ [TS(T') \preccurlyeq TT_i[k]]$.
3. Execute RP_T on i's site view, create $TS(T)$, and append u_T to the view
4. Unlock the quorum sites.

Note that the timetable condition in Step 2 will eventually be satisfied due to the liveness assumption and since $\mathscr{T}$ is a fixed set of transactions.

THEOREM 4.2.2.1. *Consider a B-run* $R = [\mathscr{W}_R, \rightarrow_R, <_R, g_0]$ *with given* δ *and* $|Q|$. *Then the maximum number of transactions which precede a transaction* T *in* $<_R$ *and which were not in the view seen by* RP_T *is* $N = \delta * (\lfloor M/Q \rfloor - 1)$.

PROOF. Let T be initiated at a site i, and suppose it is ignorant of some set of transactions, Ig_T, having smaller timestamps than $TS(T)$. Then $(\forall T' \in Ig_T) T' \mathsf{conc}_R T$.

After Step 2 (from Lemma 4), each nonquorum site knows of the updates of all transactions in $\mathscr{T}$ after Step 2 and hence is ignorant of at most $\delta - 1$ transactions that quorum sites participated in.

Let M satisfy $M = k * |Q| + c$, where c and k are integers and $0 \leq c < |Q|$. Let $N = \delta * (\lfloor M/|Q| \rfloor - 1)$. Note that any transaction T_1 executed by a site in Q_T after T completes satisfies $T \rightarrow_R T_1$ and therefore $TS(T) < TS(T_1)$. Any transaction T_2 that a site in Q_T participates in before it participates in T satisfies $T_2 \rightarrow_R T$. Thus, all transactions in Ig_T are executed at sites in $(\mathscr{S} - Q_T)$.

Assume that $|Ig_T| > N$, and consider any transaction T_3 in Tg_T. It has to accumulate a quorum Q_{T_3} of $|Q|$ sites. Then $Q_T \cap Q_{T_3} = \phi$. (If this were not so, then either $T_3 \rightarrow_R T$ or $T \rightarrow_R T_3$. In either case, T is not concurrent to T_3, and so $T_3 \notin Ig_T$.)

Now choose T' to be the transaction with the largest timestamp in Ig_T, and let T' be executed at site j. At the time T' was initiated, site j determined that site i was ignorant of at most $\delta - 1$ transactions that sites in $Q_{T'}$ knew about at that time. This implies that i was ignorant of at most $\delta - 1$ transactions that sites in $Q_{T'}$ had participated in up to that time. Let K be the set containing those transactions and T' as well. Then $|K| \leq \delta$. At most, all elements of K are in Ig_T.

Let $T'' \in (Ig_T - K)$. Then $TS(T') > TS(T'')$. Furthermore, $Q_{T''}$ cannot intersect the quorums of either T or T'. (If $Q_{T''} \cap Q_T \neq \phi$, then $T'' \notin Ig_T$, which contradicts the assumption $T'' \in (Ig_T - K)$. Similarly, if $Q_{T''} \cap Q_{T'} \neq \phi$, then $T'' \rightarrow_R T'$ from Lemma 4.1.7. Since $T'' \notin K$, it follows that T'' should be known to T which means that $T'' \notin Ig_T$.) Since T'' is an arbitrary element of $(Ig_T - K)$, there must be at least $(|Ig_T| - \delta)$ transactions in Ig_T which the sites in Q_T and Q'_T have not participated in. It follows that the quorums of all transactions in $Ig_T - K$ are confined to $M - 2 * |Q|$ sites. By selecting T'' to be the transaction in $Ig_T - K$ with the largest timestamp we are in a position to repeat the earlier argument. Note that the cardinality of the set $Ig_T - K$ is at least $N + 1 - \delta$. By successive application of the same argument, we arrive at the result that the quorums of at least $N + 1 - (k - 1) * \delta$ transactions in Ig_T will have to be confined within $(M - k * |Q|) = c$ sites. But

$$N + 1 - (k - 1) * \delta = \delta * \left\lfloor \left(\frac{M}{|Q|} - k \right) \right\rfloor + 1,$$

and $\lfloor M/|Q| - k \rfloor = \lfloor M - k * |Q|/|Q| \rfloor$ which is 0 since $c < |Q|$. Thus one transaction must have been initiated with a quorum of size c, and this is a contradiction. □

We have shown that N is an upper bound on ignorance. We now show a run in which this upper bound is attained. Let the system quiesce, and wait until all sites know of all updates. Then all sites will have the same site view and timetable. Assume that the sites are partitioned into disjoint sets of size $|Q|$ (with c sites left over, if $M = k * |Q| + c$, where c and k are integers and $0 \leq c < |Q|$). Assume also that a transaction initiated at a site uses the set to which the site belongs as its quorum. We thus have $\lfloor M/|Q| \rfloor$ disjoint quorums. The algorithm permits δ transactions to execute within each quorum with no exchange of gossip messages between quorums. Any transactions at a particular quorum is concurrent to $\delta * (\lfloor M/|Q| \rfloor - 1)$ transactions.

4.3 Discussion

The parameters δ and $|Q|$ can be varied to get a range of values for N. If $|Q| > \lfloor M/2 \rfloor$, all runs are serializable. If serializability can be relaxed, the $|Q|$ can be reduced and performance improves because (1) it takes less time to gather a quorum and (2) conflicting transactions can execute concurrently. If gossip messages are exchanged frequently enough, the timetable checks at Step 1 and Step 2 in Algorithms A and B, respectively, may already be satisfied when a transaction is submitted.

Algorithms A and B can be optimized in several ways without changing the value of N. For instance, assume that during the time a site j is locked due to the execution of a transaction, j can send out only those gossip messages that have updates seen by it before it was locked. Thus any gossip message containing the updates seen at j after j was locked can be sent out only after it is unlocked. If in Step 1 of Algorithm A, $\forall k \in \mathcal{SS}$ is replaced by $\forall k \in S$, where S is any subset of $\mathcal{SS}$ such that $|S| = M - |Q| + 1$ then it can be shown [Krishnakumar and Bernstein 1990] that the value of N derived in Theorem 4.2.1.1 does not change. The corresponding modification for Algorithm B is to replace $\forall k \in \mathcal{SS} - Q$ in Step 2, with $\forall k \in S$, where S is any subset of $\mathcal{SS}$ with $M - 2 * |Q| + 1$ nonquorum sites. Using S instead of $\mathcal{SS}$ reduces the waiting time (on the average) in Steps 1 and 2 of Algorithms A and B respectively, since the new timetable check allows a transaction to be initiated without having to wait for messages from $|Q| - 1$ sites.

The system model that we have described up to this point has not dealt with failures of any kind. Assume now that we allow site crashes, message delays, and communication link failures and that we rely on a standard commit protocol [Gray 1978; Skeen 1982] to abort transactions interrupted by failures of quorum sites. In this context, the above transformed algorithms are fault tolerant: they can allow up to $|Q| - 1$ nonquorum site failures (since the timetable check need not take into account up to $|Q| - 1$ sites), and transactions can still be initiated and executed successfully at the sites that have not failed. Furthermore, observe that as with quorum locking, Algorithms A and B might result in deadlocks. We assume that standard techniques [Bernstein et al. 1987] are used to breaking deadlocks.

We now discuss briefly how Algorithms A and B can be extended to allow the concurrent execution of transactions accessing disjoint sets of data items. Each transaction declares, at the time it is submitted, the data items that it

might access. Then, a site is allowed to hold locks simultaneously on behalf of transactions that are accessing disjoint data sets. If a site is trying to initiate transaction T, it can simultaneously try to initiate another transaction T' if T and T' access different sets of data items. The timetable checks in both Algorithms A and B are also weakened. For instance, in the extension to Algorithm A, T is initiated (at Step 1) when all other sites are ignorant of at most $\delta - 1$ transactions initiated at i, where *each transaction accesses some data item also accessed by* T. Notice that this check is less restrictive than the one in the original algorithm. A similar weaker check can be given for Algorithm B.

5. ANALYSIS OF *N*-IGNORANCE

Let $\mathscr{S}$ be the set of all possible global states of the database. Given an *N-ignorant* system, a subset, $\mathscr{S}'$, of $\mathscr{S}$ may not be reachable. For instance, in a 0-*ignorant* system, if $C(s)$ is false for some C in $\mathscr{C}$, then s is not reachable in the system. To verify that a given *N-ignorant* system is correct (according to our relaxed definition of correctness), it is necessary to determine its reachable states, and demonstrate that they do not violate the relaxed constraints, $\mathscr{C}'$. This section reasons about which states in $\mathscr{S}$ are reachable. We first provide an analysis of the system in which we make no assumptions about which replica/concurrency control algorithm is being used. Using this analysis, a superset of the reachable states of the system can be determined. We then show the "locality" property of *N-ignorance* that asserts that only the most recent updates in the total order are not seen by a transaction. To arrive at tighter characterizations of the set of reachable states, we reason about the actual runs of the *N-ignorant* system for Algorithms A and B, and provide constructive methods of deriving the reachable states.

We make the following assumptions:

Assumption 5.1. In any run, R, a transaction T in $\mathscr{W}_R$ is of the form:

if P_T then u_T

where P_T is the read phase predicate evaluated on the site view during RP_T, and u_T, the update, is a sequence of assignment statements.

If P_T is true of the site view, then u_T is generated; otherwise the null update is generated. If C is in $\mathscr{C}$, let $P_{T,C}$ satisfy $P_{T,C} \Rightarrow wp(u_T, C)$, where $wp(u_T, C)$ is the weakest precondition of u_T which guarantees its termination in a state satisfying C. Let P_T^1 denote the assertion $\bigwedge_{C \in \mathscr{C}} P_{T,C}$, and P_T^2 denote the assertion $\bigwedge_{C \in \mathscr{C}} C$.

Assumption 5.2. $P_T \equiv P_T^1 \wedge P_T^2$.

Informally, a compensating transaction is one that takes the system from a "bad" state (one in which $\mathscr{C}$ is not satisfied) to a "better" state. Assumption 5.2 rules out compensating transactions (see Krishnakumar [1992] where Assumption 5.2 is removed).

5.1 An Implementation-Independent Analysis of *N*-Ignorance

This section present properties of an *N-ignorant* system that hold irrespective of the control algorithm being used.

THEOREM 5.1.1. *Consider an N-ignorant system and a run R with final state g. If the updates of every pair of transactions commute then there exists a state g′ satisfying* $\mathscr{C}$ *such that g can be obtained by concatenating at most N updates to g′.*

PROOF. Consider the last transaction, T, in $<_R$, such that RP_T produces a nonnull update. Let i be the site of initiation of T, and l be the view seen in RP_T. When T executes it is unaware of at most N transactions in $\mathscr{U}_R$. Since all updates commute, g may be obtained by concatenating the updates of those N transactions to the state $l \bullet \langle u_T \rangle$, which is the required state g'. Further, $l \bullet \langle u_T \rangle$ satisfies $\mathscr{C}$, otherwise a null update would have been produced by RP_T from Assumption 5.2. □

It follows from Theorem 5.1.1 that, if the updates of every pair of transactions commute, a superset of the set of reachable states in the system can be obtained by taking every state satisfying the constraints and concatenating all sequences of N or fewer updates to them.

Example 5.1.2. We extend the airline system example. The initial state is given by res_seats = 0. Let the constraints $\mathscr{C}$ be as follows:

(C1) res_seats ≥ 0

(C2) res_seats ≤ 200

The transactions in the system are RESERVE and CANCEL:

T1. RESERVE:
P^1_{T1}:
$P_{T1,C1}$: [True]
$P_{T1,C2}$: [res_seats < 200]
u_{T1}: res_seats := res_seats + 1

T2. CANCEL:
P^1_{T2}:
$P_{T2,C1}$: [res_seats > 0]
$P_{T2,C2}$: [True]
u_{T2}: res_seats = res_seats − 1

$P_{T1,C1}$ and $P_{T2,C2}$ are actually [res_seats ≥ -1] and [res_seats ≤ 201] respectively. For brevity, we have simplified $P_{T1,C1}$ and $P_{T2,C2}$ to [True] since P^2_{T1} and P^2_{T2} check whether $C1$ and $C2$ hold respectively.

Since every pair of updates commute, it follows from Theorem 5.1.1 that in an *N-ignorant* system, res_seats = $200 + N$ is the most overbooked state with respect to $C2$. □

In order to relax the assumption that updates commute, we say that a set of updates *nc-commutes* if each update, u_T, in the set can be decomposed into two parts:

(1) a commutative part, denoted u^m_T, satisfying the condition that for any two updates u_T and $u_{T'}$, in the set, u^m_T and $u^m_{T'}$ commute.

(2) a noncommutative part, denoted u^n_T, satisfying the following three restrictions:

(NC1) Any data item updated in u^n_T is overwritten with a constant.

(NC2) If any two updates in the set, u_T and $u_{T'}$, are such that u_T^n and $u_{T'}^n$ do not commute, then all data items updated in u_T^n are also updated in $u_{T'}^n$.

(NC3) For any two updates in the set, u_T and $u_{T'}$, u_T^n and u_T^m, commute.

Consider a run R and assume that the set of updates of the transactions in $\mathscr{W}_R$ *nc-commutes*. Then the transactions in $\mathscr{W}_R$ can be partitioned into classes called *nc-classes* such that the updates of transactions from different *nc-classes* commute.

Example 5.1.3. The airline reservation system of Example 5.1.2 is augmented with two more transactions that change the size of the plane. The state of the system is represented by the tuple (size,res_seats). The initial state is (Small,0). Let the constraints for the system be:

(C1) res_seats $\geq$ 0.

(C2) [size = Small) $\Rightarrow$ (res_seats $\leq$ 200)].

(C3) [(size = Large) $\Rightarrow$ (res_seats $\leq$ 300)].

The transactions in the system are RESERVE, CANCEL, ENLARGE, and SHRINK:

T1. RESERVE:
- P_{T1}^1:
  - $P_{T1,C1}$: [True]
  - $P_{T1,C2}$: [(size = Large) $\vee$ (res_seats < 200)]
  - $P_{T1,C3}$: [(size = Small) $\vee$ (res_seats < 300)]
- u_{T1}^m: res_seats = res_seats + 1
- u_{T1}^n: null

T2. CANCEL:
- P_{T2}^1:
  - $P_{T2,C1}$: [res_seats > 0]
  - $P_{T2,C2}$: [True]
  - $P_{T2,C3}$: [True]
- u_{T2}^m: res_seats = res_seats − 1
- u_{T2}^n: **null**

T3. ENLARGE:
- P_{T3}^1:
  - $P_{T3,C1}$: [True]
  - $P_{T3,C2}$: [True]
  - $P_{T3,C3}$: [res_seats $\leq$ 300]
- u_{T3}^m: null
- u_{T3}^n: size = Large

T4. SHRINK:
- P_{T4}^1:
  - $P_{T4,C1}$: [True]
  - $P_{T4,C2}$: [res_seats $\leq$ 200]
  - $P_{T4,C3}$: [True]
- u_{T4}^m: null
- u_{T4}^n: size = Small

In any run, the SHRINK and ENLARGE transactions are in the same *nc-class*. RESERVE and CANCEL can be assigned to arbitrary *nc-classes*.

THEOREM 5.1.4. *Consider an N-ignorant system and a run R with final state g. If the set of updates nc-commutes then there exists a state g′ satisfying $\mathscr{C}$ such that g can be obtained by concatenating at most N of either updates or commutative parts of updates to g′.*

PROOF. Consider the last transaction T in $<_R$ that produces a nonnull update. Let $\mathscr{T}$ be the set of transactions in R concurrent to T. Let l be the view seen by RP_T and g' be the state $l \bullet \langle u_T \rangle$. g' satisfies $\mathscr{C}$ since the null update was not produced by RP_T.

g' is a subsequence of g. Thus, g can be obtained by inserting the updates of all the transactions from $\mathcal{T}$ into the sequence g', at the positions specified by $<_R$. For each transaction T' in $\mathcal{T}$, define $p_{T'}$ as follows:

(1) If $RP_{T'}$ produces a nonnull update and is the last transaction from its *nc-class* in $<_R$, then $p_{T'} = u_{T'}$.
(2) If $RP_{T'}$ produces a nonnull update and is not the last transaction from its *nc-class* in $<_R$, then $p_{T'} = u^m_{T'}$.
(3) If $RP_{T'}$ produces a null update, then $p_{T'} = \mathsf{null}$.

Then g can be obtained by appending, in any order, to g' the updates $p_{T'}$ for each T' in $\mathcal{T}$. This follows because

(1) For any T' and T'' in $\mathcal{T}$, $p_{T'}$ and $p_{T''}$ commute.
(2) If $p_{T'} = u^m_{T'}$, then it follows from (NC2) that all the data items updated by $u^n_{T'}$, are updated by another transaction (from the same *nc-class*) which occurs after T' in $<_R$. Thus, the values in g of the data items updated by $u^n_{T'}$ are not influenced by $u^n_{T'}$. Therefore, $u^n_{T'}$ need not form a part of $p_{T'}$.

It follows from Theorem 5.1.4 that any reachable state can be reached by starting in a state g' satisfying $\mathcal{C}$ and appending at most N of either updates or commutative parts of updates. For example, the most overbooked state with respect to constraint (C2) in Example 5.1.3 is $(\mathsf{Small}, 200 + N)$.

5.1.1 *The Locality of N-ignorance.* *N-ignorance* has the property that in any run R, the update of a transaction T is unknown to at most N transactions that follow T in the $<_R$ order (since all these transactions are concurrent to T). We refer to this as the *locality* of *N-ignorance*. Consider the following example. Assume that there are two sites, i and j, and $N = 2$. Let both sites have the same initial site view $\langle u_E \rangle$. Consider a 2-ignorant run, R, in which each site executes two transactions, A and B at i, and C and D at j, such that $A \rightarrow_R B$ and $C \rightarrow_R D$. Assume that the timestamps of the transactions are such that the final global state, g, is $\langle u_E, u_A, u_B, u_C, u_D \rangle$. The state l at site j immediately after the execution of D is $\langle u_E, u_C, u_D \rangle$.

Define the transaction sequence of a state s, tr-seq(s), as the sequence of all transactions T (such that $u_T \in s$) ordered in the same way as the corresponding updates in s. For instance, tr-seq $(\langle u_A, u_B \rangle)$ is $\langle A, B \rangle$. For two sequences, s_1 and s_2, define lcp(s_1, s_2) as the longest common prefix of s_1 and s_2. If s'_1 is a prefix of s_1, define suff(s_1, s'_1) such that $s_1 = s'_1 \bullet \mathsf{suff}(s_1, s'_1)$. In the example above, lcp $(g, l) = \langle u_E \rangle$ and suff$(g, \langle u_E \rangle) = \langle u_A, u_B, u_C, u_D \rangle$. Denote the length of a sequence s by $|s|$. Then note that $|\mathsf{suff}(g, \langle u_E \rangle)| = 4 \leq 2 * N$. Lemma 5.1.1.1 formalizes this relationship.

LEMMA 5.1.1.1. *Consider an N-ignorant system and a run R with final state g. Consider the site view l seen the last transaction, T, in* $<_R$, *and let* $g' = l \bullet \langle u_T \rangle$. *Then* $|\mathsf{suff}(g, \mathsf{lcp}(g, g'))| \leq 2 * N$.

PROOF. Assume the lemma is not true, and thus if lcp(g, g') is denoted by p, $|\mathsf{suff}(g, p)| \geq 2 * N + 1$. The situation is shown in Figure 3. Let tr-seq(g') be denoted by $<'_R$. Consider transactions T_1 and T_2 in $<'_R$. Since $<_R$ is the

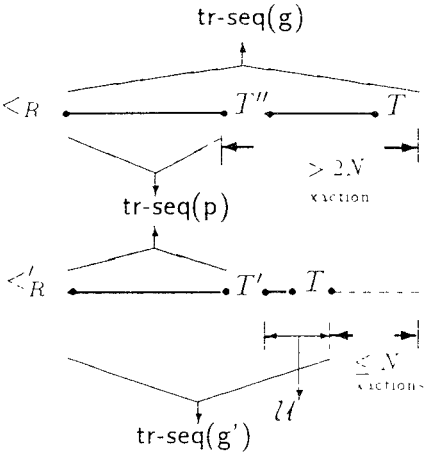

Figure 3

global total ordering of transactions, T_1 precedes T_2 in $<_R$ iff it precedes T_2 in $<'_R$.

Let the first transaction in tr-seq(suff(g, p)) and tr-seq(suff(g', p)) be denoted by T'' and T' respectively, and let the transactions in $<'_R$ that occur after T' be denoted $\mathscr{U}$. Note that by assumption, T'' precedes T' in $<_R$. T'' cannot be in $\mathscr{U}$, for if it were, then T'' would follow T' in $<'_R$, which is a contradiction. Thus T'' does not occur in $<'_R$. Furthermore, consider any transaction $U \in \mathscr{U} \cup \{T'\}$. T'' is concurrent with U, since

(1) $U \not\rightarrow_R T''$: if this were not the case, U would precede T'' in $<_R$, and therefore it would precede T' in $<'_R$.
(2) $T'' \not\rightarrow_R U$: if this were not the case, T'' would precede U in $<'_R$, since sites learn of updates in $\rightarrow_R$ order.

Since l is the site view of T, at most N updates in g are not in g'. Since $|\text{suff}(g, p)| \geq 2 * N + 1$, it follows that $|\mathscr{U} \cup \{T'\}| \geq N + 1$. However, since T'' is concurrent with all transactions in $\mathscr{U} \cup \{T'\}$, it is concurrent with at least $N + 1$ transactions. This is not allowable in an *N-ignorant* system. □

From Lemma 5.1.1.1, it follows that a transaction might not see in its site view at most N of the preceding $2 * N$ updates in $<_R$. To illustrate this, consider a run, R, of transactions from Example 5.1.3 with initial state (Small,0). Let $\mathscr{U}_R$ include a single enlarge transaction and a single shrink transaction, and let the remaining elements be reserve transactions. Assume that in $<_R$, the enlarge transaction is followed by the shrink transaction.

Assuming *N-completeness*, reserve transactions that execute after the shrink in $<_R$ can be ignorant of the shrink update (and $N - 1$ others), and keep reserving seats assuming that the size is Large. The final state can then be (Small,300 + $N - 1$) (since the last reserve transaction in $<_R$ has to preserve (C3) and can be ignorant of the shrink update and $N - 1$ other reserve updates). However, Lemma 5.1.1.1 ensures that in an *N-ignorant* system the read phase of the $(2 * N + 1)$st reserve transaction after the

shrink will see the shrink update, and will thus produce the null update. Hence the value of res_seats in the worst state is $200 + 2 * N$. (In fact, in the previous section, we utilized the properties of the updates to show that it is impossible to reach a state in which the size is Small and the value of res_seats exceeds $200 + N$.)

In a general situation where the updates do not commute (and do not satisfy any of the special cases discussed earlier), we can use the locality property to compute the set of reachable states.

THEOREM 5.1.1.2. *Consider an N-ignorant system and a run R with final state g. Then there exists g′ satisfying* $\mathscr{C}$ *such that g can be obtained by removing at most N updates from g′ and then concatenating an update sequence u such that (a) the updates that were removed from g′ occur in u in the same order as g′ and (b) at most N other updates are inserted among these removed updates.*

PROOF. Consider the last transaction T in $<_R$ that produces a nonnull update, and let RP_T see g'' in R. Then $g' = g'' \bullet \langle u_T \rangle$ satisfies $\mathscr{C}$. g' can be missing at most N updates from g. From the proof of Lemma 5.1.1.1, at most $2 * N$ updates need to be appended to $\mathrm{lcp}(g, g'')$ to obtain g, and these updates include those that are in g but are missing g'. □

5.2 Analysis of N-Ignorance Using Algorithms A and B

The results of Theorems 5.1.1 and 5.1.4 do not discuss whether the bounds they prescribe are reachable. For instance, consider a flight which can have at most 200 passengers, and assume that there are two types of reserve transactions: one that reserves a single seat and another that reserves 60 seats. If $N = 4$, then Theorem 5.1.1 implies that $200 + 4 * 60 = 440$ is an upper bound on the number of reserved seats. If $M = 2$, however, a global state with 440 reserved seats is not reachable, since it is not possible to execute four transactions which each reserve 60 seats at the same site. The most overbooked reachable state has actually 381 reserved seats. (This is possible in a run with an initial state of 19 seats reserved at both sites, and then each site executes a sequence of three 60-seat transactions and one 1-seat transaction. Each of the four transactions at one site is ignorant of all the 4 transactions at the other site.) Thus, we must examine the feasibility of a run to provide a better upper bound. We do this by considering particular implementations.

In addition to Assumptions 5.1 and 5.2, we assume that all updates commute with one another. This allows us to treat an update sequence as a set, and we represent the addition and removal of updates from an update sequence by the operators $\cup$ and $-$.

Further, we make two more assumptions.

Assumption 5.2.1. $P_{T,C} = wp(u_T, C)$.

Assumption 5.2.1 states that if all constraints are true in a state $l \cup \{u_T\}$, then RP_T generates u_T if it executes on l.

Example 5.2.2. To illustrate Assumption 5.2.1, consider the airline reservation system in Example 5.1.2. Assume that there is a second kind of reserve transaction: one which reserves two seats if res_seats < 150. Then Assumption 5.2.1 is invalid since an attempt to reserve two seats fails when the site view indicates that 190 seats are reserved. This is so since the state with 192 reserved seats satisfies the constraint, but the read phase predicate is false since the site view seen by the transaction does not have less than 150 seats.

Consider a constraint $C \in \mathscr{C}$. Let $\mathscr{S}$ be the set of all possible states. We define an update, u_T, to be *increasing* [Lynch et al. 1986] with respect to C if and only if it satisfies the condition:

$$\neg(\exists s \in \mathscr{S})[C(s) = \mathit{false} \wedge C(s \cup \{u_T\}) = \mathit{true}].$$

We define an update, u_T, to be *decreasing* with respect to C if and only if it satisfies the condition:

$$\neg(\exists s \in \mathscr{S})[C(s) = \mathit{true} \wedge C(s \cup \{u_T\}) = \mathit{false}].$$

A transaction T is defined to be increasing or decreasing with respect to C according to whether u_T is increasing or decreasing with respect to C.

Assumption 5.2.3. All transactions in the system are either increasing or decreasing with respect to each constraint $C \in \mathscr{C}$.

Example 5.1.2 illustrates a system that satisfies all of the above assumptions. $T1$ is an increasing transaction with respect to constraint $C2$ but decreasing with respect to constraint $C1$. On the other hand, $T2$ is increasing with respect to $C1$ and decreasing with respect to $C2$. In Example 5.1.3, $T3$ is increasing with respect to $C3$ and decreasing with respect to $C2$, whereas $T4$ is increasing with respect to $C2$ and decreasing with respect to $C3$.

There may exist transactions that are both increasing and decreasing with respect to a constraint C, and they will be considered to be increasing transactions with respect to C in the following analysis.

For simplicity, we assume that we are dealing with one constraint, C, and hence, will refer to the read phase predicate as P_T. In the general case we can reason separately about each constraint. Thus given that Algorithm A or B is being used, our goal is to determine all the reachable states of the system for a particular value of N. The maximum liability incurred due to constraint violation can then be assessed.

We simplify the problem by assuming that we deal only with runs where all transactions produce nonnull updates. (In Krishnakumar and Bernstein [1990] it is shown that this assumption does not change any of the results in this section.)

Consider an A-run, R, in an *N-ignorant* system. Let the set $Last_j$ have as its elements the last δ transactions that are initiated at site j. Define $Incr_j$ to be the subset of $Last_j$ satisfying $\{T | T \in Last_j \wedge T \text{ is increasing}\}$, and $Decr_j$ to be the subset $Last_j - Incr_j$. Define $\mathscr{U} = \{T | (\forall j \in \mathscr{SF})\ (T \notin Last_j)\}$. Denote

$\mathscr{D} = \bigcup_{j \in \mathscr{J}^{\mathscr{D}}} Decr_j$, and $\mathscr{I} = \bigcup_{j \in \mathscr{J}^{\mathscr{I}}} Incr_j$. Similarly, the sets $\mathscr{U}$, $\mathscr{D}$, and $\mathscr{I}$ can be defined for a B-run, and the reader is referred to Appendix B for the definitions. Note that for any run R (using either Algorithm A or B), $\mathscr{W}_R = \mathscr{U} \cup \mathscr{D} \cup \mathscr{I}$.

If S is a set and $\trianglelefteq$ a partial order on the elements of S, we say that the pair $\langle S, \trianglelefteq \rangle$ is a *partially ordered set*. If $\trianglelefteq$ is a total order, then $\langle S, \trianglelefteq \rangle$ is a *chain*. $|S|$ denotes the *cardinality* of the chain $\langle S, \trianglelefteq \rangle$. The *least* element (if it exists) of a chain $\langle S, \trianglelefteq \rangle$ is the element h in S such that $h \trianglelefteq h'$, for all $h' (\neq h)$ in S. If S_1 and S_2 are disjoint sets, and $\langle S_1, \trianglelefteq_1 \rangle$ and $\langle S_2, \trianglelefteq_2 \rangle$ are chains, then $\langle S_1, \trianglelefteq_1 \rangle$ and $\langle S_2, \trianglelefteq_2 \rangle$ are said to be *disjoint*, and their *union* is given by $\langle S_1 \cup S_2, \trianglelefteq_1 \cup \trianglelefteq_2 \rangle$ (since $S_1 \cap S_2 = \phi$, this is not a chain).

If S is a subset of $\mathscr{W}_R$, let $\mathsf{upd}(R, S)$ denote the set of updates due to the execution of transactions from S in R. Since every pair of updates commute, we abuse the concatenation operator $\bullet$: if l is a state, then $l \bullet \mathsf{upd}(R, S)$ denotes the state resulting from concatenating to l the updates of transactions of S in any order.

THEOREM 5.2.4. *Consider an N-ignorant system and assume that every pair of updates commutes. For every A-run (B-run) R with initial state g_0 and final state g, there exists an A-run (B-run) R' with the following properties*:

(1) *R' has final state g.*
(2) *R' has initial state $g'_0 = g_0 \bullet \mathsf{upd}(R, \mathscr{U} \cup \mathscr{D})$, and $C(g'_0) = true$.*
(3) *$\mathscr{W}_{R'} = \mathscr{I}$ and the partially ordered set $\langle \mathscr{W}_{R'}, \rightarrow_{R'} \rangle$ is the union of at most $\lfloor M/|Q| \rfloor$ chains of transactions, such that any two chains are disjoint.*
(4) *The read phase of the least element of each chain sees g'_0.*
(5) *Furthermore,*
   (*a*) *if the algorithm is A, $|\mathscr{W}_{R'}| \leq \delta * M$.*
   (*b*) *if the algorithm is B, $|\mathscr{W}_{R'}| \leq \delta * \lfloor M/|Q| \rfloor$, and the cardinality of each chain is at most δ.*

Theorem 5.2.4 shows that the final state of any run of an arbitrarily large number of transactions can be produced by a run which has at most $\delta * M$ increasing transactions for Algorithm A or at most $\delta * \lfloor M/|Q| \rfloor$ increasing transactions for Algorithm B. Thus if we take each state s, such that $C(s) = true$, as the initial state of a run (s is the starting state at all sites), and execute at most $\delta * M$ increasing transactions for Algorithm A or at most $\delta * \lfloor M/|Q| \rfloor$ increasing transactions for Algorithm B, we can generate all the reachable states of the system. Furthermore, the transactions are executed in disjoint chains, so that *no gossip messages are sent in the new run, and the quorums are assigned statically and are disjoint*. Thus no details of quorums and gossip messages occur in this construction. We have provided thereby a constructive procedure for determining the exact set of reachable states. The proof of Theorem 5.2.4 is given in Appendix B. Theorem 5.2.4 is extended in Krishnakumar and Bernstein [1990] to the case where the set of updates *nc-commutes*, and where Assumption 5.2.2 can be discarded.

6. A MATRIX OF IGNORANCE

We have assumed that each transaction conflicts with every other transaction. This was done for simplicity, and in this section we relax that assumption. Consider, for example, a queue having two transaction types: Enq that enqueues a single element at the tail of the queue and Deq that removes a single element from the head of the queue. If we assume that an Enq transaction can be ignorant of any number of preceding Enq or Deq transactions and that a Deq transaction can be ignorant of the updates of at most 25 preceding Enqs and at most 10 preceding Deqs, we get the following relaxed behavior:

(1) A Deq can dequeue any of the first 25 elements of the queue.

(2) The same element of the queue can be dequeued at most 11 times.

This is the specification for a "bounded stuttering semiqueue" that we saw briefly in Section 3.

In general, consider two transaction types $\mathcal{T}_1$ and $\mathcal{T}_2$. Suppose a transaction of type $\mathcal{T}_1$ can be ignorant of any number of transactions of type $\mathcal{T}_1$ and at most $N_2(>0)$ transactions of type $\mathcal{T}_2$. Similarly, suppose a transaction of type $\mathcal{T}_2$ can be ignorant of at most $N_1(>0)$ transactions of type $\mathcal{T}_1$ and at most $N_2'(>0)$ transactions of type $\mathcal{T}_2$. These constraints can be combined into an *ignorance matrix*, *IM* (where a blank in the position $[r, c]$ indicates that a transaction of type $\mathcal{T}_r$ can be ignorant of an unbounded number of transactions of type $\mathcal{T}_c$):

	T_1	T_2
T_1		N_2
T_2	N_1	N_2'

An ignorance matrix is a generalization of a standard conflict relation: a conflict is recorded in the ignorance matrix by a zero and the absence of a conflict by a blank. There is a relationship between $IM[r, c]$ and $IM[c, r]$. If $IM[r, c]$ is zero then a transaction of type $\mathcal{T}_r$ cannot be concurrent with any transaction of type $\mathcal{T}_c$. In that case, even if $IM[c, r]$ is greater than zero, a transaction of type $\mathcal{T}_c$ cannot be concurrent with a transaction of type $\mathcal{T}_r$. Thus concurrency can be realized only if $IM[r, c]$ *and* $IM[c, r]$ are greater than zero.

Algorithms A and B can be extended to implement an ignorance matrix. In the following we describe how this is done for the above ignorance matrix using Algorithm B. (The extension to more than two transaction types is straightforward.)

A timetable TT_i is maintained at each site i, which carries information about both types of transactions. Two $\mathcal{T}_1$-type locks do not conflict (two transactions of type $\mathcal{T}_1$ can hold locks at the same time at a site), but a $\mathcal{T}_1$-type lock conflicts with a $\mathcal{T}_2$-type lock, and vice versa. A transaction holds a lock until it completes. Suppose the quorum sizes for types $\mathcal{T}_1$ and $\mathcal{T}_2$ are

$|Q_1|$ and $|Q_2|$ respectively. As in Algorithm B, it is not necessary that $|Q_1| + |Q_2| > M$. (The quorums need not intersect as in a serializable case [Herlihy 1987]. Note, however, that if $IM[1,2] = 0$ or $IM[2,1] = 0$, the quorums must intersect.)

Note that Algorithm B was developed under the assumption that all transactions conflict. This need not be true. For example, $\mathcal{T}_1$-type transactions do not conflict with each other, and as a result it should be possible for an arbitrary number of them to execute concurrently *as long as no* $\mathcal{T}_2$*-type transaction is executing in the network*. Furthermore, the following situation should not occur: more than N_1 $\mathcal{T}_1$-type transactions are executing concurrently, and a $\mathcal{T}_2$-type transaction is initiated (its quorum need not intersect with those of the $\mathcal{T}_1$-type transactions, so it might not be aware of those transactions). If the ignorance level in Algorithm B is set to N_1, this situation can be avoided since the initiator of the $\mathcal{T}_2$-type transaction cannot be ignorant of more than N_1 concurrent transactions. Unfortunately, in order to ensure this, Algorithm B restricts pessimistically the level of "multiprogramming": whether or not an attempt is being made to initiate a $\mathcal{T}_2$-type transaction, a bound is placed on the number of $\mathcal{T}_1$-type transactions that can be executing concurrently.

To overcome this limitation we extend the gossip message technique to distribute information about active (as well as completed) transactions. Hence, a site learns of the existence of some concurrent transactions and can place an upper bound on the number of concurrent transactions of which it might be ignorant. As a result, a site trying to initiate a $\mathcal{T}_2$-type transaction can place a bound on the number of $\mathcal{T}_1$-type transactions with which it might be concurrent and thus decide whether to initiate the transaction or not.

The extension is implemented as follows. When a $\mathcal{T}_1$-type transaction T is submitted at site i, it tries to acquire a quorum and thus tries to lock i. If site i can be locked (i.e., no transaction of type $\mathcal{T}_2$ has site i in its quorum), a dummy transaction T_d that is associated with T and having a null update is created. T_d represents the fact that T has been submitted. T_d is treated as if it is of type $\mathcal{T}_1$ (except that no quorum is required) and is assigned a timestamp as in Section 4 by using $TT_i[i]$. The null update is logged in i's history, and the timetable is also updated. $TS(T_d)$ is associated with T as its *initial timestamp*. We assume that the creation and assignment of an initial timestamp to a transaction is done atomically, so that concurrent transactions at the same site get unique timestamps.

As T_d's update propagates across the network in gossip messages, sites learn that a transaction T (with initial timestamp $TS(T_d)$) of type $\mathcal{T}_1$ has been submitted. Furthermore, since T_d is assigned a timestamp, site i can determine from TT_i the sites at which T_d is known.

Let a *candidate* transaction of type $\mathcal{T}_1$ at site i be either

(1) a transaction T' of type $\mathcal{T}_1$ that has completed and whose update is known at i or

(2) a dummy transaction T_d associated with a transaction T of type $\mathcal{T}_1$ such that T has not completed, but the null update of T_d has been logged at i.

Notice that it is not necessary that either T' or T be submitted at i. Furthermore, the number of candidate dummy transactions seen at i is exactly the number of transactions of type $\mathcal{T}_1$ whose submission but not completion is known to i. Let δ_1 satisfy $\delta_1 > 1$. Before initiating a transaction of type $\mathcal{T}_1$, i accumulates a quorum of size $|Q_1|$ and, using TT_i and its log, determines that the following assertion is true: *there does not exist a nonquorum site that is ignorant of more than* $\delta_1 - 1$ *candidate transactions of type* $\mathcal{T}_1$ *that i has seen.*

Suppose a transaction T' of type $\mathcal{T}_2$ is submitted at site j. Consider a site i not in $Q_{T'}$. Since the quorum of a $\mathcal{T}_2$-type transaction need not intersect the quorum of a $\mathcal{T}_1$-type transaction, i can be executing transactions of type $\mathcal{T}_1$ without j knowing of their existence. However, there can be at most δ_1 transactions of type $\mathcal{T}_1$ seen by site i that are concurrent to T' such that j has not even seen the corresponding dummy transactions. This observation allows us to relate the value of δ_1 to N_1. There are at most $M - |Q_2|$ sites that are not in T''s quorum. The proof of Algorithm B (Theorem 4.2.2.1 in Section 4.2.2) provides the intuition that mutually disjoint quorums of size $|Q_1|$ allow the largest number of transactions of type $\mathcal{T}_1$ to be concurrent to T'. There can be at most $\lfloor (M - |Q_2|)/|Q_1| \rfloor$ such quorums. Since T' can be concurrent to at most δ_1 transactions of type $\mathcal{T}_1$ within each such quorum. T' is concurrent to at most $\delta_1 * \lfloor (M - |Q_2|)/|Q_1| \rfloor$ such transactions. Suppose further that j knows of c candidate dummy transactions of type $\mathcal{T}_1$. Each such dummy transaction corresponds to a transaction T of type $\mathcal{T}_1$ such that T is concurrent to T'. Thus the maximum number of $\mathcal{T}_1$-type transactions that can be concurrent to T' is given by $c + \delta_1 * \lfloor (M - |Q_2|)/|Q_1| \rfloor$. If N_1 is greater than or equal to the value, T' can be initiated.

Assume that before initiating a transaction of type $\mathcal{T}_2$, a site must determine that there does not exist a nonquorum site that is ignorant of more than $\delta_2 - 1$ candidate transactions of type $\mathcal{T}_2$. Using similar reasoning, it follows that a transaction T'' of type $\mathcal{T}_1$ can be concurrent to at most $\delta_2 * \lfloor (M - |Q_1|)/|Q_2| \rfloor$ transactions of type $\mathcal{T}_2$ whose dummy transactions it has not seen. T'' can be initiated only if it has seen at most $N_2 - \delta_2 * \lfloor (M - |Q_1|)/|Q_2| \rfloor$ candidate dummy transactions of type $\mathcal{T}_2$. Similarly, a transaction of type $\mathcal{T}_2$ needs to verify that at most $N_2' - \delta_2 * \lfloor (M - |Q_2|)/|Q_2| \rfloor$ candidate dummy transactions of type $\mathcal{T}_2$ have been seen before it can be initiated.

We summarize the three conditions under which a transaction T of type $\mathcal{T}_1$ can be initiated at site i:

(1) $|Q_1|$ sites have been locked in Q_T.
(2) There does not exist a nonquorum site that is ignorant of more than $\delta_1 - 1$ candidate transactions of type $\mathcal{T}_1$ at i.
(3) i sees at at most $N_2 - \delta_2 * \lfloor (M - |Q_1|)/|Q_2| \rfloor$ candidate dummy transactions of type $\mathcal{T}_2$.

The conditions under which a transaction T' of type $\mathcal{T}_2$ can be initiated at site i are as follows:

(1) $|Q_2|$ sites have been locked in $Q_{T'}$.

(2) There does not exist a nonquorum site that is ignorant of more than $\delta_2 - 1$ candidate transactions of type $\mathcal{T}_2$ at 1.

(3) i sees at most $N_2' - \delta_2 * \lfloor (M - |Q_2|)/|Q_2| \rfloor$ candidate dummy transactions of type $\mathcal{T}_2$, and at most $N_1 - \delta_1 * \lfloor (M - |Q_2|)/|Q_1| \rfloor$ candidate dummy transactions of type $\mathcal{T}_1$.

In general, we associate an integer δ_i with each transaction type i. Suppose a transaction of type $\mathcal{T}_i$ can be ignorant of $N_{i,j}$ transactions of type $\mathcal{T}_j$, and $|Q_j|$ and $|Q_i|$ are the quorum sizes for type $\mathcal{T}_j$ and $\mathcal{T}_i$, respectively. Let T be of type $\mathcal{T}_i$. The i can see at most $N_{i,j} - \delta_j * \lfloor (M - |Q_i|)/|Q_j| \rfloor$ candidate dummy transactions of type $\mathcal{T}_j$ when it initiates T.

7. CONCLUSIONS AND FUTURE WORK

We have introduced the notion of *N-ignorance* in replicated systems as a mechanism for increasing concurrency among transactions, with the tradeoff that the integrity constraints of the system may be violated to a bounded extent. *N*-ignorance has the nice property that only the more recent updates in the total order are unknown to a transaction. The technique is useful in systems where transactions make incremental changes to the database state. Furthermore, as demonstrated in the airline reservation example, the idea of ignorance can be useful even though updates like size changes are not incremental.

We have described algorithms for implementing *N*-ignorance, and have investigated the relationship between the value of N and the extent to which a constraint is violated, so that one may verify that the behavior of the system is acceptable. We have provided some implementation-independent results that help us identify a superset of the reachable states. By taking the specific implementations into account, we give results that enable us to obtain the exact set of reachable states under certain assumptions.

N-ignorance is most useful when many transaction types conflict. To improve concurrency when few transaction types conflict, we have extended the algorithms to permit the use of a matrix of ignorance, so that the allowable ignorance of each transaction type with respect to each other type can be individually specified. With this extension *N-ignorance* allows higher concurrency that one-copy serializability, since (1) nonconflicting locks can be recognized and (2) concurrency restrictions based on conflicts can be relaxed using ignorance. The actual performance, however, of the algorithms used to implement *N-ignorance* depends largely on how frequently gossip messages are propagated and how much time is involved in processing these messages. Any system that is based on ignorance will have to be tuned to determine the appropriate frequency of gossip messages.

Several other problems remain open. Krishnakumar [1992] discusses how compensating transactions can be included for analysis in the model. A compensating transaction is normally used to bring a system that does not satisfy the constraints back to a "good" state. However, the concept of semantic compensation has been difficult to capture [Korth et al. 1990]. For the analysis in Krishnakumar [1992], a compensating transaction is simply

modeled as one that does not make a "bad" state even "worse." We define a partial order on reachable states to formalize the notions of "bad" and "worse." It is important to develop a lucid theory of compensation in *N-ignorant* systems, since compensation is integral to such a system. Issues that have to be addressed by such a theory include the following questions: (a) when to compensate, (b) whether to compensate on the "bad" state as a whole or to compensate only for an "errant" transaction, (c) how to avoid overcompensation in a replicated environment, and so on.

Another area that needs to be investigated is in how to determine the value of N (or the ignorance matrix) given the original and relaxed constraints. This problem is difficult in general [Krishnakumar 1991], and a case study of how the matrix can be derived automatically in some cases (such as constraints over relational databases) is shown in Krishnakumar [1992].

Finally we would like to identify classes of applications (beyond those where the transactions do only incremental changes) where the idea of ignorance is useful in increasing concurrency. One example is that of the airline reservation system with size changes. It is important to characterize such systems accurately.

APPENDIX A

Properties of the Gossip Messages Scheme

LEMMA A.1. *For any t, and for any site i, $\forall j \in \mathcal{SI}$ $[TT_j^t[i] \preccurlyeq TT_i^t[i]]$.*

PROOF. The nontrivial case is when $i \neq j$. Let us refer to transaction executions and the receipts of gossip messages collectively as (interesting) *events* (the send of a gossip message is not considered since it does not affect the sender's timetable). The lemma follows by induction on the number, n, of events in the system.

Base Case. $n = 0$. The timetables remain unchanged forever, so that at any time $t, TT_i^t[i, k] = 0 = TT_j^t[i, k]$, for all i, j, and k in $\mathcal{SI}$.

Induction Hypothesis. Assume that the lemma holds when n events take place in the system.

Inductive Step. Consider the case when $n + 1$ events take place in the system. We have to show that the $n + 1$st event preserves the lemma. Assume that this event occurs at time t'. From the induction hypothesis, we know that the lemma is true for all $t < t'$. We consider now the cases for the $n + 1$st event:

(1) Transaction execution at site i: in this case, only $TT_i[i, i]$ is incremented at A1, so $TT_j[i]$ is unchanged.
(2) Receipt of a gossip message at site i: this can either increase $TT_i[i, k]$ for some k or not alter $TT_i[i, k]$ for any k. $TT_j[i]$ remains unchanged.
(3) Receipt of a gossip message at $j(\neq i)$: assume that the gossip message was sent at time t'' from site l such that $t'' < t'$. $TT_j[i]$ can be updated

only at Step (M2). From the induction hypothesis, for all $t < t', TT_j^t[i] \preccurlyeq TT_i^t[i]$. Furthermore, $TT_l^t[i] \preccurlyeq TT_i^t[i]$. Observe that TT_l^t is sent in the gossip message from l. Then for any k in $\mathcal{FS}$, $TT_j[i,k]$ cannot be updated to a value larger than $TT_i^{t'}[i,k]$ at (M2).

In all the above cases, the lemma is preserved for any time greater than or equal to t'. □

LEMMA A.2. *Consider distinct sites i and j. Assume that* $TT_i[i,s]$, *for some* s, *is changed at time* t *due to either a transaction execution at* i *or the receipt of a gossip message at* i. *Then for all* $t' > t$, $TT_i^t[i,s] \leq TT_j^{t'}[i,s]$ *if and only if there exists a chain of gossip messages from* i *to* j *such that the first message is sent from* i *after* t, *the last message is received at* j *before or at* t'.

PROOF.

If. Given that a chain of messages exists from i to j as in the statement of the lemma, show that at $t', TT_i^t[i,s] \leq TT_j^{t''}[i,s]$.

This follows from (M1), (M2), and the gossip property.

Only If. Given that at $t, TT_i[i,s]$ changes, and at $t' > t, TT_i^t[i,s] \leq TT_j^{t'}[i,s]$. Show that there exists a chain of gossip messages from i to j as in the statement of the lemma.

The follows by induction on the number, n, of events in the system (as in Lemma 4.1.3).

Base Case. $n = 0$. The timetables remain unchanged forever, so the lemma is trivially true.

Induction Hypothesis. Assume that the only-if part of the lemma holds when n events take place in the system.

Inductive Step. Consider the case when $n + 1$ events take place in the system. We have to show that the $n + 1$st event preserves the only if part of the lemma. Assume that the $n + 1$st event occurs at time t' at site j. Assume that $TT_j^{t'}[i,s] \geq TT_i^t[i,s]$. If for some $t''(< t')$, it is true that $TT_j^{t''}[i,s] \geq TT_i^{t'}[i,s]$, then the inductive hypothesis ensures that there exists a chain of gossip messages as in the statement of the lemma. If not, observe that only a gossip message from some site l can cause the update of $TT_j[i,s]$ at t'. This implies that $TT_l^{t'}[i,s] \geq TT_i^t[i,s]$. From the inductive hypothesis, there exists a chain of messages from i to l as in the statement of the lemma, and by appending the message from l to j, we get the required chain from i to j. □

LEMMA A.3. *Consider a site* i *and a transaction* T. *For any* t, $TS(T) \preccurlyeq TT_i^t[i]$ *if and only if* u_T *is in the site view of* i *at* t.

PROOF. Let T be initiated at site j at $t'(< t)$.

If. Given that u_T is in the site view of i at t, show that $TS(T) \preccurlyeq TT_i^t[i]$.

(1) $i = j$: follows from (A1), (A2), and Observation 4.1.1.

(2) $i \neq j$: let u_T be in the site view of i at t. There then exists a chain of gossip messages, each containing u_T, such that the first message in the

chain is sent from site j after t', and the last message is received at site i before or at after t. We know that $TS(T) = TT_j^{t'}[j]$. From Observations 4.1.1 and 4.1.2 and the existence of the chain above, it follows that $TS(T) \preccurlyeq TT_i^t[i]$.

Only if. Given that $TS(T) \preccurlyeq TT_i^t[i]$, show that u_T is in the site view of i at t.

(1) $i = j$: from Lemma A.1, (M1), and (M2), $TT_i[i,i]$ is not altered when a gossip message arrives at i. Thus $TT_i[i,i]$ is incremented only when site i initiates a transaction. Thus if $TS(T)[i] \leq TT_i^t[i,i]$, then T has been initiated at i before t, so that T is in the site view at i at time t.

(2) $i \neq j$: observe that $TS(T) = TT_j^{t'}[j]$, and note that $TT_j[j,j]$ changes at t' due to the execution of T at j. From our assumption, $TT_j^{t'}[j] \preccurlyeq TT_i^t[i]$, and specifically $TT_j^{t'}[j,j] \leq TT_i^t[i,j]$. From Lemma A.2, $T_j^{t'}[j,j] \leq TT_i^t[i,j]$ only if there exists a chain of gossip messages such that the first message in the chain is sent from j after T has been initiated; the last message is received at i before or at t. From the gossip property, all the messages in the chain will contain u_T. Since the last message in the chain is received before or at t, u_T will be in the site view of i at t. □

LEMMA A.4. *Consider sites i and r and a transaction T. For any t, if $TS(T) \preccurlyeq TT_r^t[i]$, then u_T is in the site view of i at t.*

PROOF. From Lemma A.1, $TT_r^t[i] \preccurlyeq TT_i^t[i]$, and we are given that $TS(T) \preccurlyeq TT_r^t[i]$. Thus $TS(T) \preccurlyeq TT_i^t[i]$, so that, from Lemma A.3, u_T is in the site view of i at t. □

LEMMA A.5. *If $T_2 \rightarrow T_1$, then $TS(T_2) \preccurlyeq TS(T_1)$. Further, if T_1 is initiated at site i, then $TS(T_2)[i] < TS(T_1)[i]$.*

PROOF. Let T_2 be initiated at site j at time t' and T_1 at site i at time t, where $t' < t$. By definition, u_{T_2} is in i's view at some time $t'' < t$. From Lemma A.3, $TS(T_2) \preccurlyeq TT_i^{t''}[i]$. Note that at t, both $TT_i[i,i]$ is incremented in (A1) and the vector $TT_i^t[i]$ is assigned as the timestamp of T_1 in A2. From Observation 4.1.1 and the above results, the lemma follows. □

LEMMA A.6. *The timestamps assigned to transactions are globally unique.*

PROOF. Assume not, so that for two transactions T_1 and T_2, $TS(T_2) = TS(T_1)$. Then from Lemma A.5, $T_1 \mathsf{conc}\ T_2$. Let T_1 be initiated at site i at time t. $TS(T_2) \preccurlyeq TT_i^t[i]$ by assumption. It follows from Lemma A.3 that u_{T_2} is in the site view of i when T_1 is initiated. But then $T_2 \rightarrow T_1$, which is a contradiction. □

APPENDIX B

PROOF OF THEOREM 5.2.4 First we define the sets $\mathscr{V}$, $\mathscr{D}$, and $\mathscr{S}$ in the case of a B-run, R. Choose at most $\lfloor M/|Q| \rfloor$ transactions as in Figure 4 to form a set $\mathscr{S}$.

```
If T is the last transaction in <_R, then include T in L
While ( |L| < ⌊M/Q⌋ ) do
begin
    Choose the transaction T' with the largest timestamp such
        that Q_T' does not intersect any of the quorums of
        transactions in L
    If such a T' cannot be found,
    then exit
    else add T' to L
end
```

Fig. 4. Procedure to construct $\mathcal{L}$.

Notice that the quorums of any two transactions in $\mathcal{L}$ are disjoint. Let $\mathcal{Q}$ denote the set of quorums, Q_T, such that $T \in \mathcal{L}$. If $T \in \mathcal{L}$, consider the last $\delta - 1$ transactions in the $<_R$ order that RP_T sees. Denote the set that includes T and these $\delta - 1$ transactions as $Last_{Q_T}$. For each $q \in \mathcal{Q}$, define $Incr_q$ to be the subset of $Last_q$ satisfying $\{T' | T' \in Last_q \wedge T'$ is increasing$\}$. The set $Decr_q$ is defined to be $Last_q - Incr_q$. Finally, let $\mathcal{U} = \{T' | (\forall q \in \mathcal{Q}) T' \notin Last_q\}$, $\mathcal{D} = \bigcup_{q \in \mathcal{Q}} Decr_q$, and $\mathcal{I} = \bigcup_{q \in \mathcal{Q}} Incr_q$.

Before proving Theorem 5.2.4, we prove two important lemmas.

LEMMA A.7. *Consider the execution of an A-run (B-run) R. Denote the updates of all transactions in $\mathcal{U}$ by $\mathcal{U}^u$. Then either of the following is true of any transaction $T \notin \mathcal{U}$:*

(1) *RP_T sees all the updates in $\mathcal{U}^u$, or*
(2) *there exists a transaction $T' (\notin \mathcal{U})$ such that $RP_{T'}$ sees all the updates in $\mathcal{U}^u$ and u_T.*

PROOF. First assume that R is an A-run. Let T be initiated at site i. T, by assumption, is in $Last_i$. Assume that RP_T does not see all the updates in $\mathcal{U}^u$. Assume further that if T_i is the last transaction initiated at i, RP_{T_i} does not see all the updates in $\mathcal{U}^u$. (If RP_{T_i} does see all updates in $\mathcal{U}^u$, then the lemma follows trivially.)

There then exists a site j and a transaction T'' initiated at j such that both $T'' \in \mathcal{U}$ and $u_{T''}$ is known to i only after i has executed T_i. i must learn of $u_{T''}$ eventually; otherwise the last transaction in $Last_j$, T_j, cannot be initiated. From Lemmas A.2 and A.3 and the gossip property, T_j sees the updates of all transactions initiated at site i (Figure 5). Now there are two cases:

(1) RP_{T_j} sees all updates in $\mathcal{U}^u$, and also u_T. The second condition in the statement of the lemma is satisfied by substituting T_j for T'.
(2) Otherwise, the argument can be repeated with respect to T_j. Thus there exists another site k such that some transactions not in $Last_k$ is not known at j until T_j has been executed. Furthermore, if T_k is the last transaction initiated at k, then RP_{T_k} will have to see the updates of all

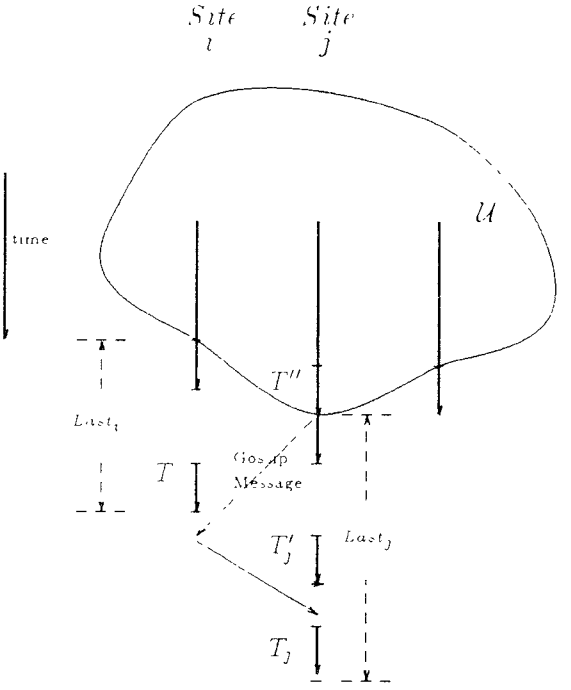

Figure 5

the transactions initiated at both i and j. This situation is similar to the one encountered earlier. Hence a similar reasoning may be used. Note that k cannot be i, since RP_{T_k} sees all the transactions in $Last_j$, and $T_i \rightarrow_R T_j$.

Since there are a finite number of sites, there must exist a site m such that the last transaction in $Last_m$ is the required transaction T'.

The proof of the lemma for Algorithm B follows the same lines as for Algorithm A. Let $T \in Last_q$ for some $q \in \mathcal{Q}$. Consider the transaction T_q in $\mathcal{L}$ whose quorum is q (by construction, T_q is unique). There are two cases for T_q:

(1) RP_{T_q} sees all the updates in $\mathcal{U}^u$. Then the second condition in the statement of the lemma is satisfied by substituting T_q for T'.
(2) Otherwise, there exists a transaction T'' in $\mathcal{U}$ such that $u_{T''}$ is known at any of the sites in q only after T_q has executed. Furthermore, by construction, T'' is seen by the read phase of some transaction $T_{q'}$ in $\mathcal{L}$. The argument can now be repeated for $T_{q'}$. □

LEMMA A.8. *Consider a system using Algorithm A, with given* δ *and* $|Q|$. *Consider a set of transactions* $\mathcal{T}$ *and a set of sites* $\mathcal{F}$ *such that* $|\mathcal{F}| \geq |Q|$ *and* $|\mathcal{T}| \leq \delta * |\mathcal{F}|$. *Assume that L is a linear order on the transactions in* $\mathcal{T}$. *Then*

there exists an A-run R of the transactions in $\mathcal{T}$ executed over the sites in $\mathcal{F}$ such that $\rightarrow_R = L$.

PROOF. Number the sites from 0 to $|\mathcal{F}| - 1$. Label the transactions in $\mathcal{T}$ from T_0 to $T_{|\mathcal{T}|-1}$, such that for T_i and T_j, $i < j$ if and only if $T_i \, L \, T_j$. Assume that all the sites if $\mathcal{F}$ are at the same initial state. The execution of R is done as follows. The transactions labeled T_0 to $T_{\delta-1}$ are initiated at site 0 in succession, with a quorum of sites 0 to $|Q| - 1$. Then the transactions labeled T_δ to $T_{2\ \delta-1}$ are all initiated at site 1, with quorum from 1 to $|Q|$, and so on. Thus the set of transactions from $T_{m\ \delta}$ to $T_{(m+1)\ \delta-1}$ are all initiated at site m, with the quorum being the sites m through $(m + |Q| - 1) \bmod |\mathcal{F}|$. Furthermore, these transactions are initiated only after $T_{(m-1)\ \delta}$ to $T_{m\ \delta-1}$ have finished execution.

In run R, $\rightarrow_R = L$. Furthermore, the timetable check in Algorithm A for any transaction is trivially satisfied since at most δ transactions are initiated at each site. □

THEOREM A.9. *Consider an N-ignorant system, and assume that every pair of updates commutes. For every A-run (B-run) R with initial state g_0 and final state g, there exists an A-run (B-run) R' with the following properties:*

(1) *R' has final state g.*

(2) *R' has initial state $g'_0 = g_0 \bullet \mathsf{upd}(R, \mathcal{U} \cup \mathcal{D})$, and $C(g'_0) = true$.*

(3) *$\mathcal{U}_{R'} = \mathcal{I}$ and the partially ordered set $\langle \mathcal{H}_{R'}, \rightarrow_{R'} \rangle$ is the union of at most $\lfloor M/|Q| \rfloor$ chains of transactions, such that any two chains are disjoint.*

(4) *The read phase of the least element of each chain sees g'_0.*

(5) *Furthermore,*

  (a) *if the algorithm is A, $|\mathcal{U}_{R'}| \leq \delta * M$.*

  (b) *if the algorithm is B, $|\mathcal{U}_{R'}| \leq \delta * \lfloor M/|Q| \rfloor$, and the cardinality of each chain is at most δ.*

PROOF. Consider the state $g'_0 = g_0 \bullet \mathsf{upd}(R, \mathcal{U} \cup \mathcal{D})$. From Lemma A.7, there exists some transaction T in $\mathcal{U}_R$ such that RP_T sees the updates of all transactions in $\mathcal{U}$. Assume that RP_T sees the updates of transactions in $\mathcal{U} \cup \mathcal{D}' \cup \mathcal{I}'$, where $\mathcal{D}' \subseteq \mathcal{D}$, and $\mathcal{I}' \subseteq \mathcal{I}$. Then, since T produces a nonnull update,

$$C(g_0 \bullet \mathsf{upd}(R, \mathcal{U} \cup \mathcal{D}' \cup \mathcal{I}')) = true, \qquad \ldots(\dagger)$$

We now present some intuition that is useful throughout the proof.

Assume that if C is a constraint, s is a state such that $C(s) = true$. Then the state resulting from the removal of an increasing update from the update sequence of s still satisfies C. Likewise the state resulting from the addition of a decreasing update to s satisfies C. The intuition is clear from the definition of increasing and decreasing transactions.

It thus follows from the definition of increasing and decreasing transactions that we can substitute $\mathcal{D}$ for $\mathcal{D}'$ and ϕ for $\mathcal{I}'$ in (†) to get

$$C(g_0 \bullet \mathsf{upd}(R, \mathcal{U} \cup \mathcal{D})) = true,$$

i.e., $C(g'_0) = true$.

```
Procedure Partition
    For all T_I do F_{T_I} = ∅
    For each maximal element I of ⟨I, →_R/I⟩ do
    begin
        F_I = Q_I ; E_I = ∅
        For each T' in I initiated at a site in Q_I do
            F_I = E_I ∪ {T'}
        For each site i ∈ SI do
            If ∃T_I such that i ∈ I_{T_I}
            then begin
                Let T' be the last transaction initiated at i
                If T' →_R I
                then begin
                    F_I = F_T ∪ {i}
                    For all I'' such that T'' is in I
                            and is initiated at i,
                        E_I = E_I ∪ {T''}
                end
            end
    end
end Partition
```

Fig. 6. The Partition procedure.

We now demonstrate how R', with initial state g'_0, can be constructed from an A-run R such that $\mathscr{W}_{R'} = \mathscr{I}$. We then show how R' is constructed if R is a B-run.

A *maximal* element of a partially ordered set $\langle S, \preccurlyeq \rangle$ is defined as an element m in S such that there is no $m'(\neq m)$ in S where $m \preccurlyeq m'$. Note that if the quorum size is $|Q|$, the number of maximal elements of $\langle \mathscr{W}_R, \rightarrow_R \rangle$ is at most $\lfloor M/|Q| \rfloor$.

Consider a set, S, and a relation, Z, defined on a superset of S. Then the restriction of Z to S is denoted by Z/S.

Construction of A-Run R'. First, we construct at most $\lfloor M/|Q| \rfloor$ sets of sites using the Partition procedure in Figure 6.

Note that Partition produces a set of sites, F_T, corresponding to each maximal element T of $\langle \mathscr{I}, \rightarrow_R/\mathscr{I} \rangle$, and by construction, $|F_T| \geq |Q|$. The set E_T is the set of all transactions from $\mathscr{I}$ that are initiated in R at sites in F_T. Furthermore, the sets, E_T, are mutually disjoint, and the sets, F_T, are also mutually disjoint. $\rightarrow_{R'}$ is defined as follows: consider a maximal transaction T of $\langle \mathscr{I}, \rightarrow_R/\mathscr{I} \rangle$ and any linearization, O_T, of $\rightarrow_R/E_T$. Then $\rightarrow_{R'}/E_T = O_T$. $\langle \mathscr{W}_{R'}, \rightarrow_{R'} \rangle$ thus has at most $\lfloor M/|Q| \rfloor$ mutually disjoint chains.

An execution of (the A-run) R' is constructed as follows. We know that $|F_T| \geq |Q|$, and $|E_T| \leq \delta * |F_T|$ since at most the last δ increasing transactions

initiated at each site in F_T can be in E_T. From Lemma A.8, for any T maximal in $\langle \mathcal{S}, \rightarrow_R / \mathcal{S} \rangle$, all transactions in E_T can be executed within sites in F_T according to $\rightarrow_{R'} / E_T$. $<_{R'}$ is the linearization of $\rightarrow_{R'}$ specified by the timestamps in this execution. Furthermore, the timetable check for each transaction is trivially satisfied, since at most δ transactions are initiated at each site.

The Final State of A-Run R' is g. The above result can be shown if RP_T for each transaction T in $\mathcal{H}_{R'}$ produces the update u_T in R' (as in R) and not the null update. Thus, consider a transaction T in $\mathcal{H}_{R'}$. In R, assume that RP_T sees $g_0 \bullet \mathsf{upd}(R, \mathcal{U}' \cup \mathcal{D}' \cup \mathcal{S}')$, where $\mathcal{U}' \subseteq \mathcal{U}$, $\mathcal{D}' \subseteq \mathcal{D}$, and $\mathcal{S}' \subseteq \mathcal{S}$. We know that since T produces a nonnull update, u_T, in R.

$$C(g_0 \bullet [\mathsf{upd}(R, \mathcal{U}' \cup \mathcal{D}' \cup \mathcal{S}') \cup \{u_T\}]) = true.$$

This can be rewritten as

$$C(g_0 \bullet [\mathsf{upd}(R, \mathcal{U}' \cup \mathcal{D}') \cup \mathsf{upd}(R, \mathcal{S}') \cup \{u_T\}]) = true. \qquad (*)$$

By the construction above for R', RP_T sees in R' all updates from $g_0 \bullet (\mathsf{upd}(R, \mathcal{U} \cup \mathcal{D}))$. Assume that it also sees the updates in $\mathsf{upd}(\mathcal{R}', \mathcal{S}'')$, where $\mathcal{S}'' \subseteq \mathcal{S}$. Notice from Partition and the construction of R' that for transactions that are maximal in $\langle \mathcal{S}, \rightarrow_R / \mathcal{S} \rangle$, $\mathcal{S}'' \subseteq \mathcal{S}'$. We now show that RP_T produces u_T in R' whether or not is is maximal n $\langle \mathcal{S}, \rightarrow_R / \mathcal{S} \rangle$.

Case 1. Assume that T is maximal in $\langle \mathcal{S}, \rightarrow_R / \mathcal{S} \rangle$. So $\mathcal{S}'' \subseteq \mathcal{S}'$. From Lemma A.7, it follows that $\mathcal{U}' = \mathcal{U}$. $(*)$ is then

$$C(g_0 \bullet [\mathsf{upd}(R, \mathcal{U} \cup \mathcal{D}') \cup \mathsf{upd}(R, \mathcal{S}') \cup \{u_T\}]) = true.$$

From the definition of increasing and decreasing transactions, we can substitute $\mathcal{D}$ for $\mathcal{D}'$, and $\mathcal{S}''$ for $\mathcal{S}'$, and C will still be true of the resultant state. Thus

$$C(g_0 \bullet [\mathsf{upd}(R, \mathcal{U} \cup \mathcal{D}) \cup \mathsf{upd}(R, \mathcal{S}'') \cup \{u_T\}]) = true.$$

Any transactions in T_1 in $\mathcal{S}''$ produced a nonnull update u_{T_1} in R, and will either produce u_{T_1} or *null* in R'. Then from the definition of an increasing transaction.

$$C(g_0 \bullet [\mathsf{upd}(R, \mathcal{U} \cup \mathcal{D}) \cup \mathsf{upd}(R', \mathcal{S}'') \cup \{u_T\}]) = true.$$

From Assumption 5.2.1, RP_T produces u_T in R'.

Case 2. Assume that T is not maximal in $\langle \mathcal{S}, \rightarrow_R / \mathcal{S} \rangle$. Consider transaction T' in R such that $T \rightarrow_R T'$, T' is maximal in $\langle \mathcal{S}, \rightarrow_R / \mathcal{S} \rangle$, and $T \in E_{T'}$. Assume that $RP_{T'}$ sees, in R', the state $l' = g_0 \bullet \mathsf{upd}(R', \mathcal{U} \cup \mathcal{D} \cup \mathcal{S}_1)$, where $\mathcal{S}_1 \subseteq \mathcal{S}$. Assume that the subset of $\mathsf{upd}(R', \mathcal{S})$ that RP_T sees in R' is $\mathsf{upd}(R', \mathcal{S}_2)$, where $\mathcal{S}_2 \subseteq \mathcal{S}$. Note that T is in $\mathcal{S}_1$ and $\mathcal{S}_2 \subseteq \mathcal{S}_1$, since $T \rightarrow_{R'} T'$ from Partition and the construction of R'. From Assumption 5.2, and the fact that $RP_{T'}$ produces a nonnull update in R' (as proved above), $C(l') = true$.

Then from the definition of an increasing transaction, we can substitute $(\mathcal{I}_2 \cup \{T\})$ for $\mathcal{I}_1$ to get,

$$C(g_0 \bullet \mathsf{upd}(R', \mathcal{U} \cup \mathcal{D} \cup \mathcal{I}_2 \cup \{T\})) = true.$$

From Assumption 5.2.1, RP_T produces u_T in R'.

Since the update of every transaction in $\mathcal{W}_{R'}$ is the same in R' as in R, and any pair of updates commutes, the A-run R' produces the same final state, g, as the A-run R.

Construction of B-Run R′. For Algorithm B, the set $\mathcal{Q}$ was constructed such that it has at most $\lfloor M/|Q| \rfloor$ elements. Furthermore, for each $q \in \mathcal{Q}$, there is a chain determined by $<_R$ over $Incr_q$. The chains can be made mutually disjoint by the following. If a transaction T appears in more than one chain, then choose arbitrarily a chain in which T is retained, and remove T from all the other chains. Thus there are at most $\lfloor M/|Q| \rfloor$ mutually disjoint chains, each of length at most δ. For the B-run R', $\rightarrow_{R'}$ is the union of the ordering indicated by these chains. An execution of R' is constructed as follows. Divide $\mathcal{SI}$ into $\lfloor M/|Q| \rfloor$ mutually disjoint sets of sites, each of size at least $|Q|$. These sets are the quorums in the execution of R'. Thus all transactions in a chain are executed at a single quorum, and each chain of transactions is executed at a different quorum. The update of any transaction executed at one quorum is not known to any transaction executed in any other quorum. $<_{R'}$ is the linearization of $\rightarrow_{R'}$ specified by the timestamps in this execution. Furthermore, the timetable check for each transaction is trivially satisfied since there are at most δ transactions in each chain.

The Final State of R′ is g. Same proof as the one for Algorithm A.

We have thus constructed the A-run (B-run) R' as in the statement of the theorem. □

REFERENCES

AGRAWAL, D. AND MALPANI, A. 1991. Efficient dissemination of information in computer networks. *Comput. J. 34*, (Dec.), 534–541.

ALONSO, R., BARBARA, D., AND GARCIA-MOLINA, H. 1988. Quasi-copies: Efficient data sharing for information retrieval systems. In *Advances in Database Technology—EDBT 88*, J. W. Schmidt, S. Ceri, and M. Missikoff, Eds. Lecture Notes in Computer Science, vol. 303. Springer-Verlag, New York, 443–468.

BARBARA, D. AND GARCIA-MOLINA, H. 1992. The demarcation protocol: A technique for maintaining arithmetic constraints in distributed database systems. In *Proceedings of the International Conference on Extending Data Base Technology*. Springer-Verlag, New York.

BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Mass.

BIRMAN, K., SCHIPER, A., AND STEPHENSON, P. 1991. Light causal and atomic multicast. Tech. Rep. TR-91-1192, Cornell Univ., Ithaca, N. Y.

BIRRELL, A. D., LEVIN, R., NEEDHAM, R. M., AND SCHRODER, M. D. 1982. Grapevine: An exercise in distributed computing. *Commun. ACM 25*, 4 (Apr.), 260–274.

DURFEE, E. H., LESSER, V. R. AND CORKILL, D. D. 1987. Cooperation through communication in a distributed problem solving network. In *Distributed Artificial Intelligence*, M. N. Huhns, Ed. Research Notes in Artificial Intelligence. Morgan Kaufmann, San Mateo, Calif.

ELMAGARMID, A., ED. 1991. *Data Eng. Bull. 14*, 1 (Mar.).

ESWARAN, K. P., GRAY, J. N., LORIE, R. A., AND TRAIGER, I. L. 1976. The notion of consistency and predicate locks in a database system. *Commun ACM 19*, 11 (Nov.) 624–633.

FARRAG, A. A. AND ÖZSU, M T 1990. Using semantic knowledge of transactions to increase concurrency. *ACM Trans Database Syst. 15*, 2 (Mar.), 484–502.

FISCHER, M. J. AND MICHAEL, A. 1982 Sacrificing serializability to attain high availability of data in an unreliable network In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*. ACM, New York, 70–75.

GARCIA-MOLINA, H. 1983. Using semantic knowledge for transaction processing in a distributed database. *ACM Trans. Database Syst. 8*, 2 (June), 186–213.

GARCIA-MOLINA, H., GAWLICK, D , KLEIN, J., KLEISSNER, K., AND SALEM, K. 1991. Modeling long-running activities as nested sagas. *Data Eng. Bull. 14*, 1 (Mar.), 39–43.

GOLDING, R. A. 1992. The timestamped anti-entropy weak-consistency group communication protocol Tech Rep UCSC-CRL-92-29, Univ. of California, Santa Cruz, Calif.

GRAY, J. N. 1978. Notes on database operating systems. In *Operating Systems: An Advanced Course*. Lecture Notes in Computer Science, vol. 60. Springer-Verlag, Berlin, 393–481

HEDDAYA, A., HSU, M., AND WEIHL, W. E. 1989. Two phase gossip. Managing distributed event histories *Inf Sci. 49*, 1, 35–57

HERLIHY, M. P. 1990. Apologizing versus asking permission· Optimistic concurrency control for abstract data types. *ACM Trans. Database Syst 15*, 1 (Mar), 96–124.

HERLIHY, M P. 1987 Concurrency vs availability: Atomicity mechanisms for replicated data *ACM Trans. Comput Syst 5*, 3 (Aug.), 249–274.

HERLIHY, M P. AND WEIHL, W E. 1988 Hybrid concurrency control for abstract data types In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, New York, 201–210.

HERLIHY, M. P. AND WING, J M. 1987. Specifying graceful degradation in distributed systems In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*. ACM, New York, 167–177.

HERMAN, D. 1983. Towards a systematic approach to implement distributed control of synchronization. In *Distributed Computing Systems*, Y Paker, and J.-P. Verjus, Eds. Academic Press, New York, 3–22

JEFFERSON, D. 1985. Virtual time. *ACM Trans. Program Lang. Syst 7*, 3 (July), 404–425

KIM, J H , PARK, K H , AND KIM, M 1989. A model of distributed control Dependency and uncertainty *Inf Process. Lett 30*, 1 (Jan.), 73–77

KORTH, H. AND SPEEGLE, G. 1988. Formal model of correctness without serializability. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, 379–388

KORTH, H , KIM, W., AND BANCILHON, F. 1988. On long-duration CAD transactions. *Inf Sci. 46*.

KORTH, H F , LEVY, E., AND SIBERSCHATZ, A. 1990. A formal approach to recovery by compensating transactions. In *Proceedings of the 16 International Conference on Very Large Data Base* VLDB Endowment, 95–106.

KRISHNAKUMAR, N. 1992. Increasing concurrency and autonomy in replicated database systems Ph.D. thesis, State Univ. of New York, Stony Brook, N Y.

KRISHNAKUMAR, N. 1991. On computing serial dependency relations. Tech. Rep. SUSB-TR-91-10, State Univ. of New York, Stony Brook, N. Y. To appear in *J Comput. Syst Sci*.

KRISHNAKUMAR, N. AND BERNSTEIN, A. 1992 High throughput escrow algorithms for replicated databases. In *Proceedings of the 18 International Conference on Very Large Data Bases*. VLDB Endowment, 175–186

KRISHNAKUMAR, N. AND BERNSTEIN, A J. 1990 Bounded ignorance in replicated systems Tech. Rep SUSB-TR-90-29, State Univ. of New York, Stony Brook, N. Y.

LADIN, R , LISKOV, B., AND SHRIRA, L. 1990. Lazy replication: Exploiting the semantics of distributed services. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing* ACM, New York

LAMPORT, L. 1978 Time, clocks and ordering of events in a distributed system. *Commun ACM 21*, 7 (July), 558–565.

LEVY, E , KORTH, H , AND SILBERSCHATZ, A. 1991. A theory of relaxed atomicity In *Proceedings of the 10 Annual ACM Symposium on Principles of Distributed Computing*. ACM, New York.

LISKOV, B., LADIN, R., AND SHRIRA, L. 1988. A technique for constructing highly available distributed services. *Algorithmica 3*, 393–420.

LYNCH, N. A. 1983. Multilevel atomicity—a new correctness criterion for database concurrency control. *ACM Trans. Database Syst. 8*, 4 (Dec.), 484–502.

LYNCH, N. A., BLAUSTEIN, B. T., AND SIEGEL, M. 1986. Correctness conditions for highly available replicated databases. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*. ACM, New York, 11–28.

O'NEIL, P. E. 1986. The escrow transactional model. *ACM Trans. Database Syst. 11*, 4 (Dec.), 405–430.

PAIGE, R. 1990. Symbolic finite differencing—part i. In the *European Symposium on Programming*. Springer-Verlag, New York, 36–56.

PU, C. AND LEFF, A. 1991. Execution autonomy in distributed transaction processing. Tech. Rep. CUCS-024-91 Columbia Univ., New York.

QIAN, X. AND WIEDERHOLD, G. 1986. Knowledge-based integrity constraint validation. In *Proceedings of the 12 International Conference on Very Large Data Bases*. VLDB Endowment, 3–12.

REUTER, A. AND WACHTER, H. 1991. The contract model. *Data Eng. Bull. 14*, 1 (Mar.), 39–43.

RUSINKIEWICZ, M. AND SHETH, A. 1991. Polytransactions for managing interdependent data *Data Eng. Bull. 14*, 1 (Mar.), 39–43.

RUSINKIEWICZ, M., SHETH, A., AND KARABATIS, G. 1991. Specifying interdatabase dependencies in multidatabase environments. Tech. Rep. TM-STS-018609/1, Bellcore, Morristown, N J.

SARIN, S. K. 1986. Robust application design in highly available distributed databases. In *Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*. IEEE, New York, 87–94.

SARIN, S. K., DEWITT, M., AND ROSENBURG, R. 1988. Overview of SHARD: A system for highly available replicated data. Tech. Rep. CCA-88-01, Computer Corp. of America, Boston, Mass.

SARIN, S. K., KAUFMAN, C. W., AND SOMERS, J. E. 1986. Using history information to process delayed database updates. In *Proceedings of the 12 International Conference on Very Large Data Bases*. VLDB Endowment, 71–78.

SHA, L., LEHOCZKY, J. P., AND JENSEN, E. D. 1988. Modular concurrency control and failure recovery. *IEEE Trans. Comput. 37*, 2 (Feb.), 146–159

SHETH, A., LEU, Y., AND ELMAGARMID, A. 1991. Maintaining consistency of interdependent data in multidatabase systems. Tech. Rep. TM-STS-019409/1, Bellcore, Morristown, N. J.

SKEEN, M. D. 1982. Crash recovery in a distributed database system. Ph.D. thesis, Univ. of California, Berkeley, Calif.

VERJUS, J.-P. 1983. Synchronization in distributed systems. In *Distributed Computing Systems*, Y. Paker and J.-P Verjus, Eds. Academic Press, New York, 3–22.

WEIHL, W. E. 1989. Local atomicity properties: Modular concurrency control for abstract data types. *ACM Trans. Program. Lang. Syst. 11*, 2 (Apr.), 249–282.

WEIHL, W. E. 1988. Commutativity-based concurrency control for abstract data types. *IEEE Trans. Comput 37*, 12 (Dec.), 1488–1505.

WEIKUM, G. AND SCHEK, H.-J. 1991. Multi-level transactions and open-nested transactions. *Data Eng. Bull. 14*, 1 (Mar.), 39–43.

WONG, M. H. AND AGRAWAL, D. 1992. Tolerating bounded inconsistency for increasing concurrency in database systems. In *Proceedings of the 11th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*. ACM, New York, 236–245.

WUU, G. T. J. AND BERNSTEIN, A. 1984. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*. ACM, New York, 233–244.

Accepted June 1994