Linearizability in the Presence of Drifting Clocks and Under Different Delay Assumptions

Maria Eleftheriou[1] and Marios Mavronicolas[2]

[1] AMER World Research Ltd., Nicosia, CYPRUS
Eleftheriou.Maria@Cyprus.ACNielsen.com
[2] Department of Computer Science and Engineering
University of Connecticut, Storrs, CT 06269–3155, USA
mavronic@engr.uconn.edu

Abstract. The cost of using message-passing to implement *linearizable read/write objects* for shared memory multiprocessors with *drifting clocks* is studied. We take as cost measures the *response times* for performing read and write operations in distributed implementations of virtual shared memory consisting of such objects. A collection of necessary conditions on these response times are presented for a large family of assumptions on the network delays. The assumptions include the common one of lower and upper bounds on delays, and bounds on the difference between delays in opposite directions. In addition, we consider *broadcast* networks, where each message sent from one node arrives at all other nodes at approximately the same time.

The necessary conditions are stated in the form of "gaps" on the values that the response times may attain in any arbitrary execution of the system; the ends of the gap intervals depend solely on the delays in a particular execution, and on certain fixed parameters of the system that express each specific delay assumptions. The proofs of these necessary conditions are comprehensive and modular; they consist of two major components. The first component is independent of any particular type of delay assumptions; it constructs a "counter-example" execution, which respects the delay assumptions only if it is *not* linearizable. The second component must be tailored for each specific delay assumption; it derives necessary conditions for any linearizable implementation by requiring that the "counter-example" execution does *not* respect the specific delay assumptions.

Our results highlight inherent limitations on the *best* possible cost for each specific execution of a linearizable implementation. Moreover, our results imply lower bounds on the *worst* possible such costs as well; interestingly, for the last two assumptions on mesage delays, these worst-case lower bounds are products of the *drifting factor* of the clocks and the *delay uncertainty* inherent for the specific assumption.

1 Introduction

Shared memory has become a convenient paradigm of interprocessor communication in contemporary computer systems. Perhaps this is so due to its combined

P. Jayanti (Ed.): DISC'99, LNCS 1693, pp. 327–341, 1999.

features that, first, it facilitates a natural extension of sequential programming, and, second, it is more high-level than message-passing in terms of semantics. This convenience has favored the evolution of concurrent programming on top of shared memory for the solution of many diverse problems. Thus, supporting shared memory in *distributed memory machines* has become a currently major objective.

Unfortunately, implementing shared memory in a distributed memory machine encounters a lot of complications; these complications are due to the high degree of parallelism and the lack of synchronization between dispersed processors, that are both inherent in a distributed architecture. This necessitates the explicit and precise definition of the guarantees provided by shared memory implemented this way; such definition is called a *consistency condition*. *Linearizability* is a basic consistency condition for concurrent objects of shared memory due to Herlihy and Wing [7]. Informally, linearizability requires that each operation, spanning over an interval of time from its invocation to its response, appears to take effect at some instant in this interval. The use of linearizable data abstractions simplifies both the specification and the proofs of multiple instruction/multiple data shared memory algorithms, and enhances compositionality.

In this work, we continue the study of the impact of timing assumptions on the cost of supporting linearizability in distributed systems; this study has been initiated by Attiya and Welch [2], and continued further by Mavronicolas and Roth [12], Chaudhuri *et al.* [3], Friedman [5], and Kosa [9]. We consider a distributed system that introduces non-negligible timing uncertainty in two significant ways: first, in the synchronization with respect to real time of each individual process, and, second, in the communication among different processes.

Following previous work [2,3,5,9,12], we consider a model consisting of a collection of application programs running concurrently and communicating through virtual shared memory, which consists of a collection of *read/write objects*. These programs are running in a distributed system consisting of a collection of processes located at the nodes of a communication network. The shared memory abstraction is implemented by a *memory consistency system* (MCS), which uses local memory at each process node. Each MCS process executes a protocol, which defines the actions it takes on operation requests by the application programs. Specifically, each application program may submit requests to access shared data to a corresponding MCS process; the MCS process responds to such a request, based, possibly, on information from messages it receives from other MCS processes. In doing so, the MCS must, throughout the network, provide linearizability with respect to the values returned to application programs.

We take as cost measures the *response times* for performing read and write operations on read/write objects in such a distributed system. However, a first major diversion from previous works [2,3,5,9,12] addressing these particular cost measures is that we show bounds on them that hold for each specific execution of the system, while bounds established in previous work on the same cost measures hold only for the worst execution. Recent research work in distributed computing

theory has addressed bounds that hold for each specific execution in the context of the *clock synchronization* [1,13] and *connection management* [8,11] problems. A common argument in support of showing such "per-execution" bounds is that for certain kinds of assumptions on network delays, the costs for the worst-case execution may, in fact, have to be unbounded [1], while one may still want to award algorithms that achieve costs that are the *best possible* for each specific instance [1].

A second major diversion from previous related work [2,3,5,9,12] is with respect to assumptions on message delays; all that work has considered the relatively simple case where there are lower and upper bounds on message delays. Under this assumption, linearizable implementations of shared memory objects have been designed [3,5,12], whose efficiency depends critically on the existence of tight lower and upper bounds on message delays. This assumption, however, may not always apply, since it is often the case that there do not exist tight lower and upper bounds on message delays, while there is some other relevant information about the delays. We draw excellent motivation from the work of Attiya *et al.* [1] on clock synchronization under different delay assumptions to study the problem of implementing linearizable read/write objects in message-passing under the following assumptions on message delays (considered in [1]): (1) There is a lower and an upper bound on delays, $d-u$ and u, respectively. (2) There is a bound ε on the difference between delays in opposite directions; this assumption is supported by experimental results revealing that message delays in opposite directions of a bidirectional link usually come very close (cf. [1]). The clock synchronization problem has been already studied under this assumption [1]. (3) There is a bound β on the difference between the times when different processes receive a broadcast message; this assumption is useful for *broadcast networks* that are used in many local area networks. The clock synchronization problem has been studied under this assumption in [1,6,14].

A third major diversion from previous related work is with respect to the amount of synchronization of processes to real time. While that work [2,3,5,9,12] has assumed "perfect" (non-drifting, but possibly translated) clocks to be available to processes, we allow a small "drift" on the processes' clocks; the impact of this assumption on the time complexity of distributed algorithms has already been studied for the clock synchronization problem (see, e.g., [13]), and the connection management problem [8,11].

The main contribution of our work is a systematic methodology for proving necessary conditions on the response times of read and write operations, that hold for each specific execution of any linearizable implementation, under a variety of message delay assumptions, and allowing a small "drift" on the processes' clocks. This methodology yields a collection of corresponding necssary conditions. Our proof methodology is modular, and consists of two major components. The first component is independent of the specific type of delay assumptions, while the second one addresses each such type in a special way.

In more detail, the first component starts with a linearizable execution that is chosen in a different way for the write operation, the read operation, and

their combination, respectively. In each of the three cases, we use the technique of "retiming," originally introduced by Lundelius and Lynch for showing lower bounds for the clock synchronization problem [10], to transform this execution into another *possible* execution of the system that is *not* linearizable. The transformation maintains the view held by each process in the original execution to the result of the transformation; moreover, the clocks in the latter are still drifting. Roughly speaking, retiming is used to change the timing and the ordering of events in an execution of the system, while precluding any particular process from "realizing" the change.

The second component is tailored for each specific assumption on message delays. More specifically, the starting point of the second component is the result of transforming the original execution, and the corresponding message delays in this result. For each specific assumption on message delays, we insist that the resulting delays confirm to the assumption. This yields corresponding upper and lower bounds on the response time of the read and write operations, as a function of the message delays in the original linearizable execution.

Our lower and upper bounds highlight inherent limitations on the *best* possible cost for each specific execution of a linearizable implementation, as a function of the message delays in the execution, and the parameters associated with each specific assumption on message delays. Moreover, our results imply also $\Omega(\rho^2\varepsilon)$ and $\Omega(\rho^2\beta)$ *worst-case* lower bounds on response times for both write and read operations for the bias model and the model of broadcast networks, respectively. These lower bounds indicate that the timing uncertainty ρ^2 in the drifting clocks model must multiply the delay uncertainty (ε and β, respectively) for each of these models. We have not been able to deduce a corresponding fact for the model with lower and upper bounds on delays. (However, for the special case where $\rho = 1$, our general results imply worst-case results that are identical to those in [2,12].) This model appears to be stronger than the previous two since it does not allow unbounded delays; we conjecture that linearizable implementations allowing for response times $o(\rho^2 u)$ for both write and read operations are possible for this model.

2 Framework

For the system model, we follow [2,12]. We consider a collection of *application programs* running concurrently and communicating through virtual shared memory, consisting of a collection $\mathcal{X}$ of *read/write objects,* or *objects* for short. Each object $X \in \mathcal{X}$ attains values from a *domain*, a set $\mathcal{V}$ of *values*. We assume a system consisting of a collection of *nodes,* connected via a *communication network*. The shared memory abstraction is implemented by a *memory consistency system* (MCS), consisting of a collection of MCS processes, one at each node, that use local memory, execute some local protocol, and communicate through sending *messages* along the network. Each MCS process p_i, located at node i, is associated with an application program P_i; p_i and P_i interact by using *call* and *response* events.

Call events at p_i represent initiation of operations by the application program P_i; they are $\mathsf{Read}_i(X)$ and $\mathsf{Write}_i(X, v)$, for all objects $X \in \mathcal{X}$ and values $v \in \mathcal{V}$. *Response events* represent responses by p_i to operations initiated by the application program P_i; they are $\mathsf{Return}_i(X, v)$ and $\mathsf{Ack}_i(X)$, for all objects $X \in \mathcal{X}$ and values $v \in \mathcal{V}$. *Message-delivery events* represent delivery of a message from any other MCS process to p_i. *Message-send events* represent sending of a message by p_i to any other MCS process.

For each i, there is a physical, real-time clock at node i, readable by MCS process p_i but not under its control, that may drift away from the rate of real time. Formally, a *clock* is a strictly increasing (hence, unbounded), piece-wise continuous function of real time $\gamma_i : \Re \to \Re$. Denote $\widetilde{\gamma_i}$ the *inverse* of γ. Fix any constant $\rho > 1$, called *drift*. A *ρ-drifting clock,* or *drifting clock* for short, is a clock $\gamma_i : \Re \to \Re$ such that for all real times $t_1, t_2 \in \Re$ with $t_1 < t_2$, $1/\rho \leq (\gamma_i(t_2) - \gamma_i(t_1))/(t_2 - t_1) \leq \rho$. Define ρ^2 to be the *drifting factor* of a ρ-drifting clock. The clocks cannot be modified by the processes. Processes do not have access to real time; instead, each process obtains information about time from its clock. The call, message-delivery and timer-expire events are called *interrupt events*. The response, message-send and timer-set events are called *react* events.

Each MCS process p_i is modeled as an automaton with a (possibly infinite) set of states, including an initial state, and a transition function. Each interrupt event at MCS process p_i causes an application of its transition function, resulting in a *computation step*. The transition function is a function from tuples of a state, a clock time and an interrupt event to tuples of a state and sets of react events. Thus, the transition function takes as input the current state, the local clock time, and an interrupt event, and returns a new state, a set of response events for the corresponding application program, a set of messages to be sent to other MCS processes, and a set of timer-set events. A *history for an MCS process p_i with clock γ_i* is a mapping h_i from $\Re$ (real time) to finite sequences of computation steps by p_i such that: (1) For each real time t, there is only a finite number of times $t' < t$ such that the corresponding sequence of steps $h_i(t')$ is non-empty; thus, the concatenation of all such sequences in real-time order is also a sequence, called the *history sequence*. (2) The old state in the first computation step in the history sequence is p_i's initial state. (3) The old state of each subsequent computation step is the new state of the previous computation step in the history sequence. (4) For each real time t, the clock time component of every computation step in the sequence $h_i(t)$ is equal to $\gamma_i(t)$. (5) For each real time t, there is at most one computation step whose interrupt event is a timer-set event; this step is ordered last in the sequence $h_i(t)$. (6) At most one call event is "pending" at a time; this outlaws pipelining or prefetching at the interface between p_i and P_i. (8) For each call event, there exists a matching response event in some subsequent computation step of the history sequence.

Each pair of matching call and response events forms an *operation*. The call event marks the start of the operation, while the response event marks its end. An operation *op* is *invoked* when the application program issues the appropriate

call event for *op*; *op terminates* when the MCS process issues the appropriate response for *op*. For a given MCS, an *execution* σ is a set of histories, one for each MCS process, such that for any pair of MCS processes p_i and p_j, $i \neq j$, there is a one-to-one correspondence between the messages sent by p_i to p_j and those delivered at p_j that were sent by p_i. Use this message correspondence to define the *delay* of any message in an execution to be the real time of delivery minus the real time of sending. By definition of execution, a zero lower bound and an infinite upper bound hold on delay. Define $\Delta_{ij}^{(e)}$ to be the set of delays of messages from MCS process p_i to MCS process p_j in execution e. Two executions are *equivalent* [10] if each process has the same history sequence and associated local clock times in both. Intuitively, equivalent executions are indistinguishable to the processes, and only an "outside observer" with access to real time can tell them apart.

We continue with specific assumptions on the delays, borrowing from [1,13]. Each assumption gives rise to a particular delay model with an associated set of admissible executions. The assumption of lower and upper bounds on the delays [2,5,12] places a lower and an upper bound on the delay for any message exchanged between any pair of processes. Fix some known parameters u and d, $0 \leq u \leq d \leq \infty$; u is the *delay uncertainty,* while d is the *maximum delay.* Execution σ is *admissible* if for each pair of MCS processes p_i and p_j, for every message $\mathtt{m}$ in σ from p_i to p_j, $d_\sigma(\mathtt{m}) \in [d-u, d]$.

The assumption of bounds on the round trip delay bias [1, Section 5.2] requires that the difference between the delays of any pair of messages in opposite direction be bounded. Fix any constant $\varepsilon > 0$, called the *delay uncertainty.* Formally, an execution σ is *admissible* if for any pair processes p_i and p_j, and for any pair of messages $\mathtt{m}$ and $\mathtt{m}'$ received by p_i from p_j and received by p_j from p_i, respectively, $|d_\sigma(\mathtt{m}) - d_\sigma(\mathtt{m}')| \leq \varepsilon$.

The assumption of multicast networks has been studied in [1,4,6,14] in the context of the clock synchronization problem; our presentation follows [1, Section 5.3]. To define this assumption, we replace message-send events by events of the form $\mathsf{Broadcast}_i(\mathtt{m})$ at the MCS process p_i, for all messages $\mathtt{m}$; such events represent a broadcast of $\mathtt{m}$ to all MCS processes. The definition of an execution is modified so that for any pair of processes p_i and p_j, $i \neq j$, there is a one-to-one correspondence between the messages broadcast by p_i, and those delivered at p_j and broadcast by p_i. Use this message correspondence to define the *delay of message* $\mathtt{m}$ *to process* p_j *in execution* σ*,* denoted $d_\sigma(\mathtt{m}, p_j)$, to be the real time of delivery at p_j in σ minus the real time of broadcast by p_i in σ. Fix any constant $\beta > 0$, called the *broadcast accuracy.* Execution σ is *admissible* if for any process p_i, for any message $\mathtt{m}$ broadcast by p_i, $|d_\sigma(\mathtt{m}, p_j) - d_\sigma(\mathtt{m}, p_k)| \leq \beta$; that is, $\mathtt{m}$ reaches p_j at most β time units later it reaches p_k, and vice versa.

Each object X has a *serial specification* [7], which describes its behavior in the absence of concurrency and failures. Formally, it defines: (1) A set $Op(X)$ of *operations on* X*,* which are ordered pairs of *call* and *response* events. Each operation $op \in Op(X)$ has a value $val(op)$ associated with it. (2) A set of *legal operation sequences for* X*,* which are the allowable sequences of operations on X.

For each process p_i, $Op(X)$ contains a *read* operation $[\mathsf{Read}_i(X), \mathsf{Return}_i(X, v)]$ on X and a *write* operation $[Write_i(X, v), Ack_i(X)]$ on X, for all values $v \in \mathcal{V}$; v is the value associated with each of these operations. The set of legal operation sequences for X contains all sequences of operations on X for which, for any read operation *rop* in the sequence, either $val(rop) = \perp$ and there is no preceding write operation in the sequence, or $val(wop) = val(rop)$ for the latest preceding write operation *wop*. A sequence of operations τ for a collection of processes and objects is *legal* if, for every object $X \in \mathcal{X}$, the restriction of τ to operations on X, denoted $\tau \mid X$, is in the set of legal operation sequences for X.

Given an execution σ, let $ops(\sigma)$ be the sequence of call and response events appearing in σ in real-time order, breaking ties for each real time t as follows: First, order all response events whose matching call events occur before time t, using process identification numbers (*ids*) to break any remaining ties. Then, order all operations whose call and response events both occur at time t. Preserve the relative ordering of operations for each process, and break any remaining ties using again process ids. Finally, order all call events whose matching response events occur after time t, using process ids to break any remaining ties. An execution σ specifies a partial order $\xrightarrow{\sigma}$ on the operations appearing in σ: for any operations op_1 and op_2 appearing in σ, $op_1 \xrightarrow{\sigma} op_2$ if the response for op_1 precedes the call for op_2 in $ops(\sigma)$; that is, $op_1 \xrightarrow{\sigma} op_2$ if op_1 completely precedes op_2 in $ops(\sigma)$. Given an execution σ, an operation sequence τ is a *serialization* of σ if it is a permutation of $ops(\sigma)$. A serialization τ of σ is a *linearization* of σ if it extends $\xrightarrow{\sigma}$; that is, if $op_1 \xrightarrow{\sigma} op_2$, then $op_1 \xrightarrow{\tau} op_2$. Let τ be a sequence of operations. Denote by $\tau \mid i$ the restriction of τ to operations at process p_i; similarly, denote by $\tau \mid X$ the restriction of τ to operations on the object X. For an execution σ, these definitions can be extended in the natural way to yield $ops(\sigma) \mid i$ and $ops(\sigma) \mid X$. An execution σ is *linearizable* [7] if there exists a legal linearization τ of σ such that for each MCS process p_i, $ops(\sigma) \mid i = \tau \mid i$. An MCS is a *linearizable implementation* of $\mathcal{X}$ if every admissible execution of the MCS is linearizable.

The efficiency of an implementation $\mathcal{A}$ of $\mathcal{X}$ is measured by the *response time* for any operation on an object $X \in \mathcal{X}$. Given a particular MCS $\mathcal{A}$ and a read/write object X implemented by it, the time $|op_{\mathcal{A}}(X, \sigma)|$ taken by an operation *op* on X in an admissible execution σ of $\mathcal{A}$ is the maximum difference between the times at which the response and call events of *op* occur in σ, where the maximum is taken over all occurrences of *op* in σ. In particular, we denote by $|\mathbf{R}_{\mathcal{A}}(X, \sigma)|$ and $|\mathbf{W}_{\mathcal{A}}(X, \sigma)|$ the maximum time taken by a read and a write operation, respectively, on X in σ, where the maximum is taken over all occurrences of the corresponding operations in σ. Define $|\mathbf{R}_{\mathcal{A}}(X)|$ (resp., $|\mathbf{W}_{\mathcal{A}}(X)|$) to be the maximum of $|\mathbf{R}_{\mathcal{A}}(X, \sigma)|$ (resp., $|\mathbf{W}_{\mathcal{A}}(X, \sigma)|$) over all executions σ of $\mathcal{A}$.

Fix e to be any execution, and let $op = [\mathsf{Call}(op), \mathsf{Response}(op)]$ be any operation in e. We denote by $t_c^{(e)}(op)$ and $t_r^{(e)}(op)$ the (real) times at which $\mathsf{Call}(op)$ and $\mathsf{Response}(op)$, respectively, occur in e. We use $val^{(e)}(op)$ to denote the value associated with the "execution" of operation *op* in e.

3 Writes

A construction of a non-linearizable, if admissible, execution is presented in Section 3.1; this execution is used in Section 3.2 for deriving necessary conditions for the write operation under specific assumptions on the delays. We refer to any linearizable implementation $\mathcal{A}$ of read/write objects, including an object X with at least two writers p_i and p_j, and a distinct reader p_k.

3.1 A Non-Linearizable, if Admissible, Execution

This construction is based on one in [2, Section 4] and [12, Section 5]. We start with an admissible execution e, in which p_i writes x_i to X, then p_j writes x_j to X, $x_j \neq x_i$, and finally p_k reads x_j from X; moreover, we assume that all clocks in e run at a rate of σ for some constant σ such that $1/\rho \leq \sigma \leq \rho$. If p_i's history is shifted later, while p_j's history is shifted earlier, each by an appropriate amount, while both are either "stretched" or "shrinked" by a factor of σ, depending on whether $1 \leq \sigma \leq \rho$ or $1/\rho \leq \sigma \leq 1$, the result is an execution e', not necessarily admissible, in which the write operation by p_j precedes the write operation by p_i, which, in turn, precedes the read operation by p_k. If, in addition, all clocks are correspondingly "stretched" or "shrinked" by the same factor of σ, all three processes still "see" the same events occurring at the same local time and cannot, therefore, distinguish between e and e'; thus, in particular, p_k still reads x_j from X, which implies that e', if admissible, is not linearizable. We now present some details of the construction.

By the serial specification of X, there exists an admissible execution e of $\mathcal{A}$ consisting of the following operations at processes p_i, p_j, and p_k: p_i performs a write operation wop_i on X with $t_c^{(e)}(wop_i) = 0$ and $val^{(e)}(wop_i) = x_i$; p_j performs a write operation wop_j on X with $t_c^{(e)}(wop_j) = |wop_i|$ and $val^{(e)}(wop_j) = x_j$; p_k performs a read operation rop_k on X with $t_c^{(e)}(rop_k) = |wop_i| + \max\{|wop_i|, |wop_j|\}$; apparently, $\max\{|wop_i|, |wop_j|\} = |\mathbf{W}_{\mathcal{A}}(X, e)|$, so that $t_c^{(e)}(rop_k) = |wop_i| + |\mathbf{W}_{\mathcal{A}}(X, e)|$. Moreover, assume that $\gamma_i^{(e)}(t) = \gamma_j^{(e)}(t) = \gamma_k^{(e)}(t) = \sigma t$ for some positive constant σ such that $1/\rho \leq \sigma \leq \rho$, so that all clocks are ρ-drifting. (We omit reference to clocks of other processes in this extended abstract.)

Since $\mathcal{A}$ is a linearizable implementation and e is an admissible execution, e is a linearizable execution. Thus, there exists a legal linearization τ of e such that for each MCS process p, $ops(e) \mid p = \tau \mid p$. We use the construction of e to show simple properties of the sequence τ, namely that $wop_i \xrightarrow{\tau} wop_j$, and that $wop_j \xrightarrow{\tau} rop_k$. Since τ is a legal operation sequence, these properties imply that $val^{(e)}(rop_k) = val^{(e)}(wop_j) = x_j$.

We now "perturb" the (admissible) execution e in order to obtain another execution e', which is not necessarily admissible; however, we shall show that if e' is admissible, then it is not linearizable. We construct e' as follows. (1) Set $\gamma_i^{(e')}(t) = t/\sigma - \sigma\,|\mathbf{W}_{\mathcal{A}}(X, e)|$, $\gamma_j^{(e')}(t) = t/\sigma + \sigma\,|\mathbf{W}_{\mathcal{A}}(X, e)|$, and

$\gamma_k^{(e')}(t) = t/\sigma$. (2) For any process p_l with clock γ'_l, define a mapping h'_l from $\Re$ to finite sequences of computation steps by p_l as follows. Each step at p_l associated with real time t in h_l is associated with real time $\widetilde{\gamma}_l^{(e')}(\gamma_l^{(e)}(t))$ in e'; in addition, h'_l preserves the ordering of steps in h_l. (3) e' preserves the correspondence between message-delivery and message-send events in e.

Since e is an execution of $\mathcal{A}$, for each MCS process p_l, h_l is a history for p_l with clock $\gamma_l^{(e)}$. By rule (2), this implies that h'_l is a history for p_l with clock $\gamma_l^{(e')}$; moreover, for any real times $t_1, t_2 \in \Re$ with $t_1 < t_2$, $\gamma_l^{(e')}(t_2) - \gamma_l^{(e')}(t_1) = (t_2 - t_1)/\sigma$. Since $1/\rho \leq \sigma \leq \rho, 1/\rho \leq 1/\sigma \leq \rho$, $\gamma_l^{(e')}$ is ρ-drifting; thus, by rule (3), it follows that e' is an execution of $\mathcal{A}$. In addition, rule (2) immediately implies that executions e and e' are equivalent. We continue to establish a fundamental property of the execution e'.

Lemma 1. *Assume that e' is an admissible execution. Then, e' is not linearizable.*

Proof. We give a sketch of the proof. Since $\mathcal{A}$ is a linearizable implementation and e' is an admissible execution of $\mathcal{A}$, e' is a linearizable execution. Thus, there exists a legal linearization τ' of e' such that for each MCS process p, $ops(e') \mid p = \tau' \mid p$. We show simple properties of the sequence τ', namely that $wop_j \xrightarrow{\tau'} wop_i$ and $wop_i \xrightarrow{\tau'} rop_k$. Since τ is a legal operation sequence, these properties imply that $val^{(e')}(rop_k) = val^{(e')}(wop_i) = x_i$. Since $x_i \neq x_j$, it follows that $val^{(e)}(rop_k) \neq val^{(e')}(rop_k)$. However, the equivalence of e and e' implies that $val^{(e)}(rop_k) = val^{(e')}(rop_k)$. A contradiction.

3.2 Results for Specific Models of Delays

Our methodology is as follows. We first calculate message delays in execution e' (independent of specific delay assumptions). Next, we consider separately each specific assumption on delays; requiring that message delays in the execution e' constructed in Section 3.1 satisfy the assumption yields the admissibility of e', which, by Lemma 1, implies the non-linearizability of e'. For the model with lower and upper bounds on delays, we show:

Theorem 1. *Consider the model with lower and upper bounds on the delays. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two writers p_i and p_j, and a distinct reader p_k. Fix any parameters δ_{ij}, δ_{ji}, δ_{ik}, δ_{ki}, δ_{jk}, $\delta_{kj} > 0$. Then, for any parameter $\sigma \in [1/\rho, \rho]$, there exists an admissible execution e of $\mathcal{A}$ with $\delta_{ij} \in \Delta_{ij}^{(e)}, \delta_{ji} \in \Delta_{ji}^{(e)}, \delta_{ik} \in \Delta_{ik}^{(e)}, \delta_{ki} \in \Delta_{ki}^{(e)}, \delta_{jk} \in \Delta_{jk}^{(e)}, \delta_{kj} \in \Delta_{kj}^{(e)}$, such that either*

$$|\mathbf{W}_{\mathcal{A}}(X, e)| < \max\{\frac{\delta_{ij}}{2} - \frac{d}{2\sigma^2}, \frac{d-u}{2\sigma^2} - \frac{\delta_{ji}}{2}, \delta_{ik} - \frac{d}{\sigma^2}, \frac{d-u}{\sigma^2} - \delta_{ki}, \frac{d-u}{\sigma^2} - \delta_{jk}, \delta_{kj} - \frac{d}{\sigma^2}\},$$

or

$$|\mathbf{W}_{\mathcal{A}}(X,e)| > \min\{\frac{\delta_{ij}}{2} - \frac{d-u}{2\sigma^2}, \frac{d}{2\sigma^2} - \frac{\delta_{ji}}{2}, \delta_{ik} - \frac{d-u}{\sigma^2}, \frac{d}{\sigma^2} - \delta_{ki}, \frac{d}{\sigma^2} - \delta_{jk}, \delta_{kj} - \frac{d-u}{\sigma^2}\}.$$

Proof. We give a sketch of the proof. Assume, by way of contradiction, that there exists a linearizable implementation $\mathcal{A}$ of read/write objects, including the object X, such that for any parameter $\sigma \in [1/\rho, \rho]$, for every admissible execution e of $\mathcal{A}$ with $\delta_{ij} \in \Delta_{ij}^{(e)}, \delta_{ji} \in \Delta_{ji}^{(e)}, \delta_{ik} \in \Delta_{ik}^{(e)}, \delta_{ki} \in \Delta_{ki}^{(e)}, \delta_{jk} \in \Delta_{jk}^{(e)}, \delta_{kj} \in \Delta_{kj}^{(e)}$, neither inequality holds. We establish that the execution e' constructed in Section 3.1 is an admissible execution of $\mathcal{A}$; appealing to Lemma 1, this implies that e' is non-linearizable, which contradicts the fact that $\mathcal{A}$ is a linearizable implementation. (To prove that e' is an admissible execution, we show by case analysis that for any pair of processes p_l and p_m, and for any message $\mathtt{m}$ received by p_m from p_l, $d^{(e')}(\mathtt{m}) \in [d-u, d]$.

Theorem 1 establishes the existence of executions with "gaps" for the response times of write operations. For the model with a bound on the round-trip delay bias, we show:

Theorem 2. *Consider the model with a bound on the round-trip delay bias. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two writers p_i and p_j, and a distinct reader p_k. Fix any parameters δ_{ij}, δ_{ji}, δ_{ik}, δ_{ki}, δ_{jk}, $\delta_{kj} > 0$. Then, for any parameter $\sigma \in [1/\rho, \rho]$, there exists an admissible execution e of $\mathcal{A}$ with $\delta_{ij} \in \Delta_{ij}^{(e)}, \delta_{ji} \in \Delta_{ji}^{(e)}, \delta_{ik} \in \Delta_{ik}^{(e)}, \delta_{ki} \in \Delta_{ki}^{(e)}, \delta_{jk} \in \Delta_{jk}^{(e)}, \delta_{kj} \in \Delta_{kj}^{(e)}$, such that either*

$$|\mathbf{W}_{\mathcal{A}}(X,e)| < -\frac{\varepsilon}{4\sigma^2} + \max\{\frac{\delta_{ik}-\delta_{ki}}{2} - \frac{\varepsilon}{4\sigma^2}, \frac{\delta_{kj}-\delta_{jk}}{2} - \frac{\varepsilon}{4\sigma^2}, \frac{\delta_{ij}-\delta_{ji}}{4}\},$$

or

$$|\mathbf{W}_{\mathcal{A}}(X,e)| > \frac{\varepsilon}{4\sigma^2} + \min\{\frac{\delta_{ik}-\delta_{ki}}{2} + \frac{\varepsilon}{4\sigma^2}, \frac{\delta_{kj}-\delta_{jk}}{2} + \frac{\varepsilon}{4\sigma^2}, \frac{\delta_{ij}-\delta_{ji}}{4}\}.$$

The proof of Theorem 2 is similar to the proof of Theorem 1, and it is omitted. Theorem 2 demonstrates the existence of executions with "gaps" on the response times of write operations. In order to derive a *worst-case* lower bound on the response time for write operations from Theorem 2, we set $\sigma = 1/\rho$, $\delta_{ij} - \delta_{ji} = \varepsilon$, $\delta_{ik} - \delta_{ki} = \varepsilon$, and $\delta_{kj} - \delta_{jk} = \varepsilon$. With these choices, the upper limit on $|\mathbf{W}_{\mathcal{A}}(X,e)|$ becomes negative, and, therefore, it cannot be met, which implies that the lower limit on $|\mathbf{W}_{\mathcal{A}}(X,e)|$ must be met, which is positive for these choices. We obtain:

Corollary 1. *Consider the model with a bound on the round-trip delay bias. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two writers p_i and p_j, and a distinct reader p_k. Then, $|\mathbf{W}_{\mathcal{A}}(X)| > \rho^2\varepsilon/4 + \varepsilon/4$.*

For the model of broadcast networks, we show:

Theorem 3. *Consider the model of broadcast networks. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two writers p_i and p_j, and a distinct reader p_k. Fix any parameters δ_{ij}, δ_{ji}, δ_{ik}, δ_{ki}, δ_{jk}, $\delta_{kj} > 0$. Then, there exists an admissible execution e of $\mathcal{A}$ with $\delta_{ij} \in \Delta_{ij}^{(e)}, \delta_{ji} \in \Delta_{ji}^{(e)}, \delta_{ik} \in \Delta_{ik}^{(e)}, \delta_{ki} \in \Delta_{ki}^{(e)}, \delta_{jk} \in \Delta_{jk}^{(e)}, \delta_{kj} \in \Delta_{kj}^{(e)}$, such that either*

$$|\mathbf{W}_{\mathcal{A}}(X,e)| < -\frac{\beta}{2\sigma^2} + \max\{\delta_{ij} - \delta_{ik} - \frac{\beta}{2\sigma^2}, \delta_{jk} - \delta_{ji} - \frac{\beta}{2\sigma^2}, \frac{\delta_{kj} - \delta_{ki}}{2}\},$$

or

$$|\mathbf{W}_{\mathcal{A}}(X,e)| > \frac{\beta}{2\sigma^2} + \min\{\delta_{ij} - \delta_{ik} + \frac{\beta}{2\sigma^2}, \delta_{jk} - \delta_{ji} + \frac{\beta}{2\sigma^2}, \frac{\delta_{kj} - \delta_{ki}}{2}\}.$$

The proof of Theorem 3 is similar to the proof of Theorem 1, and it is omitted. Theorem 3 demonstrates the existence of executions with "gaps" on the response times of writes operations. In order to derive a *worst-case* lower bound on the response time for write operations from Theorem 3, we set $\sigma = 1/rho$, $\delta_{ij} - \delta_{ik} = \beta$, $\delta_{jk} - \delta_{ji} = \beta$, and $\delta_{kj} - \delta_{ki} = \beta$. With these choices, the upper limit on $|\mathbf{W}_{\mathcal{A}}(X,e)|$ becomes negative, and, therefore, it cannot be met, which implies that the lower limit on $|\mathbf{W}_{\mathcal{A}}(X,e)|$ must be met, which is positive for these choices. We obtain:

Corollary 2. *Consider the model of broadcast networks. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two writers p_i and p_j, and a distinct reader p_k. Then, $|\mathbf{W}_{\mathcal{A}}(X)| > \rho^2\beta/2 + \beta/2$.*

4 Reads

A construction of a non-linearizable, if admissible, execution is presented in Section 4.1; this execution is used in Section 4.2 for deriving necessary conditions for the read operation under specific assumptions on the delays. We refer to any linearizable implementation $\mathcal{A}$ of read/write objects including an object X with at least two readers p_i and p_j, and a distinct writer p_k.

4.1 A Non-Linearizable, if Admissible, Execution

This construction is based on one in [2, Section 4] and [12, Section 5]. We start with an admissible execution e, in which p_i reads $\perp$ from X, then p_j and p_i alternate reading from X while p_k is writing x to X, and finally p_j reads x from X; moreover, we assume that all clocks in e run at a rate of σ, for some constant σ such that $1/\rho \leq \sigma \leq \rho$. Thus, there exists a read operation rop_0, say by p_i, that returns $\perp$ and is immediately followed by a read operation rop_1 by p_j that returns x. If p_i's history is shifted later by $|\mathbf{R}_{\mathcal{A}}(X,e)|$, while p_j's history is shifted earlier by $|\mathbf{R}_{\mathcal{A}}(X,e)|$, while both are either "swelled" or "shrinked"

by a factor of σ, the result is an execution e' in which rop_1 precedes rop_0. If, in addition, all clocks are correspondingly "swelled" or "shrinked" by the same factor σ, all three processes still "see" the same events occurring at the same local time and cannot, therefore, distinguish between e and e'; thus, in particular, p_j and p_i still read x and $\perp$ in their read operations rop_1 and rop_0, respectively, in this order. This implies that e', if admissible, is non-linearizable. We now present some details of the construction.

Let $b = \lceil |\mathbf{W}_{\mathcal{A}}(X)|/2|\mathbf{R}_{\mathcal{A}}(X)| \rceil$. By the serial specification of X, there exists an admissible execution e of $\mathcal{A}$ consisting of the following operations at processes p_i, p_j, and p_k. For each integer l, $0 \leq l \leq b$, p_i performs a read operation $rop_i^{(2l)}$ on X; for each integer l, $0 \leq l \leq b$, p_j performs a read operation $rop_j^{(2l+1)}$ on X; p_k performs a write operation wop_k on X with $val^{(e)}(wop_k) = x$. For each l, $0 \leq l \leq 2b+1$, let $rop^{(l)} = rop_i^{(l)}$ if l is even, or $rop_j^{(l)}$ if l is odd. The definition of the call times of read operations in e is inductive. For the basis case, $t_c^{(e)}(rop^{(0)}) = 0$. Assume inductively that we have defined $t_c^{(e)}(rop^{(l)})$ where $0 \leq l < 2b+1$. Then, $t_c^{(e)}(rop^{(l+1)}) = t_c^{(e)}(rop^{(l)}) + |rop^{(l)}|$. Set also $t_c(wop_k) = |rop^{(0)}|$. Moreover, assume that $\gamma_i^{(e)}(t) = \gamma_j^{(e)}(t) = \gamma_k^{(e)}(t) = \sigma\, t$ for some constant σ such that $1/\rho \leq \sigma \leq \rho$, so that all clocks $\gamma_i^{(e)}, \gamma_j^{(e)}$, and $\gamma_k^{(e)}$ are ρ-drifting. (We omit reference to clocks of other processes in this extended abstract.)

Since $\mathcal{A}$ is a linearizable implementation and e is an admissible execution of $\mathcal{A}$, e is a linearizable execution. Thus, there exists a legal linearization τ of e such that for each MCS process p_l, $ops(e) \mid l = \tau \mid l$. We use the construction of e to show simple properties of the operation sequence τ, namely that $rop_i^{(0)} \xrightarrow{\tau} wop_k$ and that $wop_k \xrightarrow{\tau} rop_j^{(2b+1)}$. We show that for each l, $0 \leq l \leq 2b$, $rop^{(l)} \xrightarrow{\tau} rop^{(l+1)}$. These properties imply that there exists an index l_0, $0 \leq l_0 \leq 2b$, such that $rop^{(l_0)} \xrightarrow{\tau} wop_k \xrightarrow{\tau} rop^{(l_0+1)}$. Since τ is a legal operation sequence, this implies that $val^{(e)}(rop^{(l_0)}) = \perp$ and $val^{(e)}(rop^{(l_0+1)}) = x$. Assume, without loss of generality, that l_0 is even, so that $rop^{(l_0)}$ is a read operation by process p_i.

We now "perturb" the (admissible) execution e in order to obtain another execution e' which is not necessarily admissible; however, we shall show that if e' is admissible, then it is not linearizable. We construct e' as follows. (1) Set $\gamma_i^{(e')}(t) = t/\sigma - \sigma\,|\mathbf{R}_{\mathcal{A}}(X,e)|$, $\gamma_j^{(e')}(t) = t/\sigma + \sigma\,|\mathbf{R}_{\mathcal{A}}(X,e)|$, and $\gamma_k^{(e')}(t) = t/\sigma$. (2) e' preserves the correspondence between message-delivery and message-send events in e. (3) For any process p_l, each step at p_l occurring at real time t in e is scheduled to occur at real time $\widetilde{\gamma}_l^{(e')}(\gamma_l^{(e)}(t))$ in e'; in addition, e' preserves the ordering of steps in e. Since e is an execution of $\mathcal{A}$, for each MCS process p_l, h_l is a history for p_l with clock $\gamma_l^{(e)}$. By rule (2), this implies that h'_l is a history for p_l with clock $\gamma_l^{(e')}$; moreover, for any real times $t_1, t_2 \in \Re$ with $t_1 < t_2$, $\gamma_l^{(e')}(t_2) - \gamma_l^{(e')}(t_1) = (t_2 - t_1)/\sigma$. Since $1/\rho \leq \sigma \leq \rho$, $1/\rho \leq 1/\sigma \leq \rho$, so that $\gamma_l^{(e')}$ is ρ-drifting; thus, by rule (3), it follows that e' is an execution of $\mathcal{A}$. In addition, rule (3) immediately implies that executions e and e' are equivalent. We continue to show a fundamental property of the execution e'.

Lemma 2. *Assume that e' is an admissible execution. Then, e' is not linearizable.*

4.2 Results for Specific Models of Delays

We consider separately each specific assumption on message delays; requiring that message delays in the execution e' constructed in Section 4.1 satisfy the assumption yields the admissibility of e', which, by Lemma 2, implies that e' is not linearizable. For the model with lower and upper bounds on the delays, we show:

Theorem 4. *Consider the model with lower and upper bounds on the delays. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two readers p_i and p_j, and a distinct writer p_k. Fix any parameters δ_{ij}, δ_{ji}, δ_{ik}, δ_{ki}, δ_{jk}, $\delta_{kj} > 0$. Then, there exists an admissible execution e of $\mathcal{A}$ with $\delta_{ij} \in \Delta_{ij}^{(e)}, \delta_{ji} \in \Delta_{ji}^{(e)}, \delta_{ik} \in \Delta_{ik}^{(e)}, \delta_{ki} \in \Delta_{ki}^{(e)}, \delta_{jk} \in \Delta_{jk}^{(e)}, \delta_{kj} \in \Delta_{kj}^{(e)}$, such that either*

$$|\mathbf{R}_{\mathcal{A}}(X,e)| < \max\{\frac{\delta_{ij}}{2} - \frac{d}{2\rho^2}, \frac{d-u}{2\rho^2} - \frac{\delta_{ji}}{2}, \delta_{ik} - \frac{d}{\rho^2}, \frac{d-u}{\rho^2} - \delta_{ki}, \frac{d-u}{\rho^2} - \delta_{jk}, \delta_{kj} - \frac{d}{\rho^2}\},$$

or

$$|\mathbf{R}_{\mathcal{A}}(X,e)| > \min\{\frac{\delta_{ij}}{2} - \frac{d-u}{2\rho^2}, \frac{d}{2\rho^2} - \frac{\delta_{ji}}{2}, \delta_{ik} - \frac{d-u}{\rho^2}, \frac{d}{\rho^2} - \delta_{ki}, \frac{d}{\rho^2} - \delta_{jk}, \delta_{kj} - \frac{d-u}{\rho^2}\}.$$

For the model with a bound on the round-trip delay bias, we show:

Theorem 5. *Consider the model with a bound on the round-trip delay bias. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two readers p_i and p_j, and a distinct writer p_k. Fix any parameters δ_{ij}, δ_{ji}, δ_{ik}, δ_{ki}, δ_{jk}, $\delta_{kj} > 0$. Then, there exists an admissible execution e of $\mathcal{A}$ with $\delta_{ij} \in \Delta_{ij}^{(e)}, \delta_{ji} \in \Delta_{ji}^{(e)}, \delta_{ik} \in \Delta_{ik}^{(e)}, \delta_{ki} \in \Delta_{ki}^{(e)}, \delta_{jk} \in \Delta_{jk}^{(e)}, \delta_{kj} \in \Delta_{kj}^{(e)}$, such that either*

$$|\mathbf{R}_{\mathcal{A}}(X,e)| < -\frac{\varepsilon}{4\rho^2} + \max\{\frac{\delta_{ik} - \delta_{ki}}{2} - \frac{\varepsilon}{4\rho^2}, \frac{\delta_{kj} - \delta_{jk}}{2} - \frac{\varepsilon}{4\rho^2}, \frac{\delta_{ij} - \delta_{ji}}{4}\},$$

or

$$|\mathbf{R}_{\mathcal{A}}(X,e)| < \frac{\varepsilon}{4\rho^2} + \min\{\frac{\delta_{ik} - \delta_{ki}}{2} + \frac{\varepsilon}{4\rho^2}, \frac{\delta_{kj} - \delta_{jk}}{2} + \frac{\varepsilon}{4\rho^2}, \frac{\delta_{ij} - \delta_{ji}}{4}\}.$$

Theorem 5 demonstrates the existence of executions with "gaps" on the response times of read operations. In order to derive a *worst-case* lower bound on the response time for read operations from Theorem 5, we set $\sigma = 1/\rho$, $\delta_{ij} - \delta_{ji} = \varepsilon$, $\delta_{ik} - \delta_{ki} = \varepsilon$, and $\delta_{kj} - \delta_{jk} = \varepsilon$. With these choices, the upper limit on $|\mathbf{R}_{\mathcal{A}}(X,e)|$ becomes negative, and, therefore, it cannot be met, which implies that the lower limit on $|\mathbf{R}_{\mathcal{A}}(X,e)|$, which is positive for these choices, must be met. We obtain:

Corollary 3. *Consider the model with a bound on the round-trip delay bias. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two readers p_i and p_j, and a distinct writer p_k. Then, $|\mathbf{R}_{\mathcal{A}}(X)| > \rho^2\varepsilon/4 + \varepsilon/4$.*

For the model of broadcast networks, we show:

Theorem 6. *Consider the model of broadcast networks. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two readers p_i and p_j, and a distinct writer p_k. Fix any parameters δ_{ij}, δ_{ji}, δ_{ik}, δ_{ki}, δ_{jk}, $\delta_{kj} > 0$. Then, there exists an admissible execution e of $\mathcal{A}$ with $\delta_{ij} \in \Delta_{ij}^{(e)}, \delta_{ji} \in \Delta_{ji}^{(e)}, \delta_{ik} \in \Delta_{ik}^{(e)}, \delta_{ki} \in \Delta_{ki}^{(e)}, \delta_{jk} \in \Delta_{jk}^{(e)}, \delta_{kj} \in \Delta_{kj}^{(e)}$, such that either*

$$|\mathbf{R}_{\mathcal{A}}(X,e)| < -\frac{\beta}{2\rho^2} + \max\{\delta_{ij} - \delta_{ik} + \frac{\beta}{2\rho^2}, \delta_{jk} - \delta_{ji} + \frac{\beta}{2\rho^2}, \frac{\delta_{kj} - \delta_{ki}}{2}\},$$

or

$$|\mathbf{R}_{\mathcal{A}}(X,e)| > \frac{\beta}{2\rho^2} + \max\{\delta_{ij} - \delta_{ik} + \frac{\beta}{2\rho^2}, \delta_{jk} - \delta_{ji} + \frac{\beta}{2\rho^2}, \frac{\delta_{kj} - \delta_{ki}}{2}\}.$$

Theorem 6 demonstrates the existence of executions with "gaps" on the response times of read operations. In order to derive a *worst-case* lower bound on the response time for read operations from Theorem 6, we set $\sigma = 1/rho$, $\delta_{ij} - \delta_{ik} = \beta$, $\delta_{jk} - \delta_{ji} = \beta$, and $\delta_{kj} - \delta_{ki} = \beta$. With these choices, the upper limit on $|\mathbf{R}_{\mathcal{A}}(X,e)|$ becomes negative, and, therefore, it cannot be met, which implies that the lower limit on $|\mathbf{R}_{\mathcal{A}}(X,e)|$ which is positive for these choices, must be met. We obtain:

Corollary 4. *Consider the model of broadcast networks. Let $\mathcal{A}$ be any linearizable implementation of read/write objects, including an object X with at least two readers p_i and p_j, and a distinct writer p_k. Then, $|\mathbf{R}_{\mathcal{A}}(X)| > \rho^2\beta/2 + \beta/2$.*

References

1. H. Attiya, A. Herzberg, and S. Rajsbaum, "Optimal Clock Synchronization under Different Delay Assumptions," *SIAM Journal on Computing,* Vol. 25, No. 2, pp. 369–389, April 1996. 329, 329, 329, 329, 329, 329, 329, 329, 332, 332, 332, 332
2. H. Attiya and J. L. Welch, "Sequential Consistency versus Linearizability," *ACM Transactions on Computer Systems,* Vol. 12, No. 2, pp. 91–122, May 1994. 328, 328, 328, 329, 329, 330, 330, 332, 334, 337
3. S. Chaudhuri, R. Gawlick, and N. Lynch, "Designing Algorithms for Distributed Systems Using Partially Synchronized Clocks," *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing,* pp. 121–132, August 1993. 328, 328, 328, 329, 329, 329
4. D. Dolev, R. Reischuk, and H. R. Strong, "Observable Clock Synchronization," *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing,* pp. 284–293, August 1994. 332

5. R. Friedman, "Implementing High-Level Synchronization Operations in Hybrid Consistency," *Distributed Computing,* Vol. 9, No. 3, pp. 119–129, December 1995. 328, 328, 328, 329, 329, 329, 332
6. J. Halpern and I. Suzuki, "Clock Synchronization and the Power of Broadcasting," *Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing,* pp. 588–597, October 1990. 329, 332
7. M. Herlihy and J. Wing, "Linearizability: A Correctness Condition for Concurrent Objects," *ACM Transactions on Programming Languages and Systems,* Vol. 12, No. 3, pp. 463–492, July 1990. 328, 332, 333
8. J. Kleinberg, H. Attiya, and N. Lynch, "Trade-offs Between Message Delivery and Quiesce Times in Connection Management Protocols," *Proceedings of the 3rd Israel Symposium on the Theory of Computing and Systems,* pp. 258–267, January 1995. 329, 329
9. M. J. Kosa, "Making Operations of Concurrent Data Types Fast," *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing,* pp. 32–41, August 1994. 328, 328, 328, 329, 329
10. J. Lundelius and N. Lynch, "An Upper and Lower Bound for Clock Synchronization," *Information and Control,* Vol. 62, pp. 190–204, August/September 1984. 330, 332
11. M. Mavronicolas and N. Papadakis, "Trade-off Results for Connection Management," *Proceedings of the 11th International Symposium on Fundamentals of Computation Theory,* pp. 340–351, Lecture Notes in Computer Science, Vol. 1279, Springer-Verlag, Krakow, Poland, September 1997. 329, 329
12. M. Mavronicolas and D. Roth, "Efficient, Strongly Consistent Implementations of Shared Memory," *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG'92),* pp. 346–361, Lecture Notes in Computer Science, Vol. # 647, Springer-Verlag, November 1992. 328, 328, 328, 329, 329, 329, 330, 330, 332, 334, 337
13. B. Patt-Shamir and S. Rajsbaum, "A Theory of Clock Synchronization," *Proceedings of the 26th Annual ACM Symposium on Theory of Computing,* pp. 810–819, May 1994. 329, 329, 332
14. K. Sugihara and I. Suzuki, "Nearly Optimal Clock Synchronization under Unbounded Message Transmission Time," *Proceedings of the 3rd International Conference on Parallel Processing,* pp. 14–17, 1988. 329, 332