CHAPTER 5

Theory of Real-Time Systems – Project Survey*

Hagit Attiya[†]
Nancy A. Lynch
Laboratory for Computer Science
MIT
Cambridge, MA 02139

1 Introduction

An important area of computer applications is real-time process control, in which a computer system interacts with a real-world system in order to guarantee certain desirable real-world behavior. In most interesting cases, the real-world requirements involve timing properties, and so the behavior of the computer system is required to satisfy certain timing constraints. In order to be able to guarantee timing constraints, the computer system must satisfy some assumptions about time—for example, its various components should operate at known speeds.

It is clear that good theoretical work in the area of real-time systems is necessary. In the past few years, several researchers have proposed new frameworks for specifying requirements of such systems, describing implementations, and proving that the implementations satisfy the

*This work was supported by ONR contract N00014-85-K-0168, by NSF grants CCR-8611442 and CCR-8915206, and by DARPA contracts N00014-89-J-1988 and N00014-87-K-0825.

[†]Current address: Computer Science Department, The Technion, Haifa 32000, Israel.

requirements. These frameworks are based on, among others, state machines ([18, 78, 92]), weakest precondition methods ([43]), first-order logic ([47, 48]), temporal logic ([11]), Petri nets ([17, 62, 91]), and process algebra ([9, 40, 46, 53, 88, 100]). Work is still needed in evaluating and comparing the various models for their usefulness in reasoning about important problems in this area and perhaps in developing new models if these prove to be inadequate.

Work is also needed in developing the complexity theory of such systems; very little work has so far been done in this area. An example of the kind of work needed is provided by the theory of asynchronous concurrent systems.[1] That theory contains many combinatorial results that show what can and cannot be accomplished by asynchronous systems; for tasks that can be accomplished, other combinatorial results determine the inherent costs. In addition to their individual importance, these results also provide a testbed for evaluating modeling decisions and a stimulus for the development of algorithm verification techniques. Similar results should be possible for real-time systems. Some examples of complexity results that have already been obtained for real-time systems are the many results on clock synchronization, including [24, 42, 56, 66, 99] (see [94] for a survey).

In this project, we have embarked on a study of complexity results for real-time systems. We have formulated several abstract problems that seem to be characteristic of real-time computing, and have obtained upper and lower bounds for the time complexity of those problems. The problems we have defined are all variations of problems that have previously been studied for asynchronous concurrent systems; the major differences are that we impose rather stringent timing requirements on the solutions and that we assume that our systems satisfy certain assumptions about the timing of events. The assumptions we make about the timing of events are not exact; rather, we assume that the time required for various events is known to be within certain bounds. For example, in real-time systems, there can be uncertainty in the time required for real-world tasks to be completed, for processors to take steps, for clocks to advance and for messages to be delivered.

We have obtained bounds on the time complexity of solving three fun-

[1] Asynchronous systems are those in which processes work at completely independent rates and have no way of estimating time.

damental problems: *mutual exclusion*, *synchronization* and *agreement*, in a model with inexact timing assumptions. These results are described, respectively, in Sections 2, 3 and 4.

In the course of our work on these problems, we felt a need for a systematic method for reasoning about the correctness and performance of timing based systems. To satisfy this need, we have developed a new assertional method of reasoning about timing based systems.

Assertional reasoning is a very useful technique for proving safety properties of sequential and concurrent algorithms. This proof method involves describing the algorithm of interest as a state machine, and defining a predicate known as an *assertion* on the states of the machine. One proves inductively that the assertion is true of all the states that are reachable in a computation of the machine, i.e., that it is an *invariant* of the machine. The assertion is defined so that it implies the safety property to be proved. Assertional reasoning is a rigorous, simple and general proof technique. Furthermore, the assertions usually provide an intuitively appealing explanation of *why* the algorithm satisfies the property. One kind of assertional reasoning uses a mapping to describe a correspondence between the given algorithm and a higher-level algorithm used as a specification of correctness. (See, for example, [57, 67, 71].)

We have developed an assertional technique based on mappings, for proving correctness and timing properties of timing-based systems. This work is described in Section 5.

The formal model we have used to describe our results is the *timed automaton* model, a slight variant of the *time constrained automaton* model of [78]. We have used this model to state the requirements to be satisfied, to define the basic architectural and timing assumptions, to describe the algorithms, and to prove their correctness and timing properties. We have also used it for describing our mapping proof technique.

The last work described in this survey considers the time complexity of wait-free algorithms in shared-memory distributed systems. In such systems, it is possible for processes to operate at very different speeds, e.g., because of implementation issues such as communication and memory latency, priority-based time-sharing of processors, cache misses and page faults. It is also possible for processes to fail entirely. A *wait-free* algorithm guarantees that each nonfaulty process terminates regardless of the speed and failure of other processes ([44, 58]). Because wait-free algorithms guarantee that fast processes terminate without waiting for slow processes, wait-free algorithms seem to be generally thought of as *fast.* However, while it is obvious from the definition that wait-free algorithms are highly resilient to failures, we believe that the assumption that such algorithms are fast requires more careful examination.

We have addressed this general problem by studying the time complexity of wait-free algorithms for the *approximate agreement problem.* Our results are described in Section 6.

A major emphasis in our project has been on the impact of uncertainty in the system on the time complexity of solving problems. This uncertainty might be due to inexact timing assumptions as described above, to unpredictable inputs, or to failures. More specifically, our work on mutual exclusion involves timing uncertainty and unpredictable inputs, our work on synchronization involves only timing uncertainty, and our work on agreement and wait-free algorithms involves timing uncertainty and failures.

The portion of our work that deals with resource allocation problems is related to prior work in scheduling theory (for example, [40, 65, 86]). Our work on resource allocation is distinguished from the scheduling theory work in its emphasis on distributed algorithms and on timing uncertainties and failures. Also, our emphasis has been on upper and lower bounds for solving particular problems, e.g., mutual exclusion, whereas the emphasis in scheduling theory seems to be on general strategies (e.g., first-come-first-serve) for solving large classes of problems. As the subject matter of scheduling theory broadens, however, we believe that the two areas will become more closely related.

Our work has taken advantage of many of the approaches, techniques and results of distributed computing theory. In particular, our selection of problems, our use of automaton-style formal models and assertional reasoning, the design of our algorithms, and the techniques we use to prove lower bounds (e.g., the perturbation of executions and the use of the limitations of local knowledge) have all been heavily influenced by prior work in distributed computing theory.

2 Mutual Exclusion

We have studied a variant of the *mutual exclusion problem.* This problem is one of the fundamental problems in distributed computing; it serves as an abstraction of a large class of *hazard avoidance* problems. We note that this particular problem appears in the real-time computing literature (cf. [48]) as the "nuclear reactor problem". There, operators push different buttons to request the motion of different control rods in the same nuclear reactor. It is undesirable to have more than one control rod moving at the same time, presumably since in that case the nuclear reaction might be slowed down too much.

More specifically, we have considered a system consisting of some number, n, of identical moving parts (e.g., control rods), no two of which are supposed to move at the same time. An operator associated with each moving part can request permission for the associated part to move by pushing a button that sends a *REQUEST* signal to the control system. The control system responds with *GRANT* signals; each *GRANT* signal gives permission to the designated moving part to move, but such motion is expected to be finished no more than a fixed time, m, later. The control system is only supposed to issue a *GRANT* signal when it knows that it is safe to move the corresponding moving part, i.e., at least real time m has elapsed since the last *GRANT* signal. We assume, for simplicity, that a *REQUEST* signal is only issued by a particular operator if any preceding *REQUEST* by that operator has already been satisfied (by a corresponding *GRANT* signal). Our goal is to minimize the worst-case time between a *REQUEST* signal and the corresponding *GRANT* signal, i.e., the *worst-case response time.*

The control system might consist of a single process running on a dedicated processor or might be a distributed system running on separate processors communicating over a message system. Solving the problem efficiently requires the control system to make accurate estimates of the elapsed time since the last *GRANT* signal; the difficulty, however, is that the control system only has inaccurate information about time, as given by inaccurate clock components within the system and by estimates of the time required for certain events. Specifically, the only information about time that the control system has is the following:

1. the knowledge that a moving part will stop moving within time m

after a *GRANT* signal,

2. the knowledge that the time between successive ticks of any clock is always in the interval $[c_1, c_2]$, for known constants c_1 and c_2, where $0 < c_1 \leq c_2$,

3. the knowledge that the time between successive steps of any process within the control system is always in the interval $[0, l]$, for a known constant $l, 0 \leq l$, and

4. (if the system is distributed) the knowledge that the time to deliver the oldest message in each channel is no greater than a known constant $d, 0 \leq d$.

In the cases we have in mind, we suppose that $l \ll c_1 < c_2 \ll d \ll m$. We use the notation C to represent the ratio c_2/c_1; C is a useful measure of the timing uncertainty.

We obtain the following results. First, we consider a centralized control system, consisting of just a single process with a local clock. For that case, we show that

$$n \cdot c_2 \left(\lfloor (m + l)/c_1 \rfloor + 1 \right) + l$$

(approximately nmC) is an *exact* bound on the worst-case response time for the timing-based mutual exclusion problem. The upper bound result arises from a careful analysis of a simple FIFO queue algorithm, while the matching lower bound result arises from explicitly constructing and "retiming" executions to obtain a contradiction.

We then consider the distributed case, which is substantially more complicated. For that case, we obtain very close (but not exact) bounds: an upper bound of

$$n \left[c_2 \left(\lfloor (m + l)/c_1 \rfloor + 1 \right) + d + c_2 + 2l \right]$$

(approximately $nmC + nd$) and a lower bound of $nmC + (n - 1)d$. Assuming that the parameters have the relative sizes described earlier, e.g., that d is much larger than l, c_1 and c_2, the gap between these two bounds is just slightly more than a single message delay time. The upper bound arises from a simple token-passing algorithm, while the lower bound proof employs a new technique of shifting some of the events happening at a process while carefully retiming other events.

(The work described in this section appears in [5].)

3 Synchronization

Some problems in real-time computing involve synchronization of several computer system components, in order that they might cooperate in performing a task involving real-world components. For example, multiple robots might cooperate to build a car on an assembly line, with each robot responsible for assembling a small piece of the machinery. Similar synchronization problems arise in distributed computing as well, where the separate components might each be responsible for performing a small part of a computation. We have studied the time complexity of achieving synchronization, in the presence of various assumptions about the timing of basic events.

The particular synchronization problem we have considered is the *session problem*, first defined by Arjomandi, Fischer and Lynch ([2]). Roughly speaking, a *session* is a sequence of events that contains *at least* one step by each process. An algorithm for the *s-session problem* guarantees that each execution of the algorithm includes at least s separate sessions. It was assumed in [2] that processes communicate via *shared variables*, and the time complexity of the session problem was studied in *synchronous* and *asynchronous* models.[2] The results of [2] show that the problem can be solved much faster in the synchronous model than in the asynchronous model.

We have studied the time requirements for the session problem in distributed networks rather than shared memory systems. That is, we have considered a collection of n processes located at the nodes of an undirected communication graph G; communication is assumed to be by messages sent over the edges of G. We have considered both the asynchronous model and a model with inexact timing assumptions (which we call here the *partially synchronous* model).

[2]Synchronous systems are those in which processes operate in lock-step, taking steps simultaneously.

Time is measured under the following assumptions on the system. We assume that the delivery time for every message is in the range $[0, d]$, where d is a known nonnegative constant. Instead of assuming explicit clocks (as in the previous section), we assume that the times between successive local steps are in the range $[0, c_2]$ for the asynchronous model, and in the range $[c_1, c_2]$ for the partially synchronous model, where c_1 and c_2 are constants, $0 < c_1 \leq c_2$. As before, we define $C = c_2/c_1$.

Our upper bound results are as follows. Our first algorithm relies on explicit communication to ensure that the needed steps have occurred, and does not rely on any timing information. In either the asynchronous model or the partially synchronous model, this algorithm has time complexity $(s-1)diam(G)(d+c_2)$, where the *diameter* of an undirected graph G, $diam(G)$, is the maximum distance between any two nodes. Our second algorithm does not use any communication and relies only on timing information; this algorithm works only in the partially synchronous model. Its time complexity is $c_2 + (s-2)(\lfloor C \rfloor + 1)c_2$. These algorithms can be combined to yield a partially synchronous algorithm for the s-session problem whose time complexity is $c_2 + (s-2)\min\{(\lfloor C \rfloor + 1)c_2, diam(G)(d+c_2)\}$.

Our lower bound results are as follows. For the asynchronous model, we prove a lower bound of $(s-1)diam(G)d$ for the time complexity of any algorithm for the s-session problem; this almost matches our upper bound for that model. For the partially synchronous model, we prove two lower bounds. We first show a simple lower bound of $(s-2)\lfloor C \rfloor c_2$ for the case where communication is not used. We then present our main result: a lower bound of $c_2 + (s-2)\min\{(\lfloor \frac{C}{2} \rfloor)c_2, diam(G)d\}$ for the time complexity of any partially synchronous algorithm for the s-session problem. For appropriate values of the various parameters, these results imply a time separation between partially synchronous and asynchronous networks.

The lower bounds presented in this paper use the same general approach as in [2]. However, since we assume processes communicate by sending messages while [2] assumes processes communicate via shared memory, the precise details differ substantially. The lower bound proof in [2] uses fan-in arguments, while our lower bounds are based on information propagation arguments using long delays of messages, combined with appropriate selection of processes and careful timing arguments.

Awerbuch ([8]) has introduced the concept of a *synchronizer* as a way to translate algorithms designed for synchronous networks to asynchronous networks. Although the results of [8] may suggest that any synchronous network algorithm can be translated into an asynchronous algorithm with constant time overhead, our results imply that this is *not* the case: for some values of the parameters, any translation of a partially synchronous (and in particular, a synchronous) algorithm for the s-session problem to an asynchronous algorithm must incur a non-constant time overhead.

(The work described in this section appears in [7].)

4 Distributed Agreement

We have also considered the time complexity of the problem of *reaching agreement* among nodes in a distributed system, in the case where some of the nodes are faulty. In the version of the agreement problem we have considered, each of n processes starts with an input value. Each process that does not fail must choose a decision value such that (i) no two processes decide differently, and (ii) if any process decides v then v was the input value of some process. We assume that processes fail only by stopping (without warning). This abstract problem can be used to model agreement on the value of a sensor reading obtained with multiple sensors, or agreement on a course of action such as whether an airplane landing should be completed or aborted.

The time complexity of the distributed agreement problem has been well studied in the synchronous model. In this model, computation proceeds in a sequence of numbered rounds of communication. In each round, each non-failed process sends out messages to all processes, receives all messages sent to it at that round, and carries out some local computation. (See, for example, [60, 82, 21, 32, 19, 59, 29, 41, 76, 28, 79, 13, 80, 10] for results involving time complexity in this model.) The most basic time bound results in these papers are matching upper and lower bounds of $f + 1$ on the number of rounds of communication required for reaching agreement in the presence of at most f faults.

We have considered how these bounds are affected by using, instead of the synchronous model, one in which there are inexact timing assumptions. In particular, we have used the partially synchronous model described in Section 3, assuming that d is an upper bound on message delivery time, c_1 and c_2 are lower and upper bounds, respectively, on process step time, and $C = c_2/c_1$.[3] We have assumed that processes fail only by stopping, so that failures can be detected by "timeouts": if an expected message from some process is not received within a sufficiently long time, then that process is known to have failed.

Initially, we had hoped to be able to adapt known results about the synchronous model to obtain good bounds for the version with inexact timing. Indeed, an $(f+1)$-round algorithm can be adapted in a straightforward way to yield an algorithm for the partially synchronous model that requires time at most $(f+1)dC$ if there are f faults. On the other hand, a simple transformation in the reverse direction, of any partially synchronous algorithm to a synchronous algorithm yields a lower bound of $(f+1)d$. There is a significant gap between these two bounds, namely, a multiplicative factor equal to the timing uncertainty, C. We would like to obtain closer bounds on the time complexity of this problem; in particular, we would like to understand how this complexity depends on C.

Our main result is an agreement algorithm in which the uncertainty factor C is only incurred once, i.e., for one "round" of communication, yielding a running time of approximately $2td + dC$ in the worst case. The term of dC arises from the possible need to time out a failed process; if a process fails, then it stops sending messages, and within time approximately dC, every other process can determine that the failure has occurred. It seems surprising that the overhead for such a timeout need only occur *once* in any execution. The algorithm can be viewed as an asynchronous algorithm which uses a fault detection (e.g., timeout) mechanism. That is, the timing bounds c_1, c_2, d are used only in the fault detection mechanism.

Our second result shows that any agreement algorithm must take time at least $(f-1)d + dC$ in the worst case. The lower bound is unusual in that it combines, in a nontrivial way, three different lower bound techniques: a "chain argument" ([32, 19, 29, 76, 14, 28]), used previously to prove that $f+1$ rounds are required in the synchronous

[3]Results of [33, 20] imply that if any one of the bounds c_1, c_2, d does not exist, then there is no agreement algorithm tolerant to even one fault.

rounds mc
that fault-
and a "tir
described
tight, they
just a sing
of the mes

Some
timing ass
determinin
els; only r
were giver
the *latenc*
synchrono
from ours.
within son
stated in t

(The v

5 A r ten

Assertiona
sequential
gorithms.
be used to
sumptions
of properti
dinary" sa
erties (upp
assumptio
of real-tim
proving su

Our m

[4]The wo

rounds model; a "bivalence" argument ([33, 20]) used previously to prove that fault-tolerant agreement is impossible in an asynchronous system; and a "time stretching" argument developed to prove the lower bounds described in Section 2 ([5]). Although these bounds are not completely tight, they do demonstrate that the inherent dependence on C involves just a single term dC; there is no need to multiply C by larger multiples of the message delivery time.

Some prior work on distributed agreement in a model with inexact timing assumptions appears in [27]. The main emphasis in [27] was on determining the maximum fault tolerance possible for various fault models; only rough upper bounds on the time complexity of the algorithms were given, and no lower bounds on time were proved. Related work on the *latency*[4] of reaching agreement when processors are not completely synchronous appears in [12] and [96], although the results are different from ours. These papers assume that process clocks are synchronized to within some fixed additive error. Unlike our results, these results are not stated in terms of absolute real time.

(The work described in this section appears in [4].)

5 A new proof technique for timing-based systems

Assertional reasoning has been used primarily to prove properties of sequential algorithms and synchronous and asynchronous concurrent algorithms. We have developed a way in which assertional reasoning can be used to prove timing properties for algorithms that have timing assumptions of the kind described in the previous sections. Also, the kinds of properties generally proved using assertional reasoning have been "ordinary" safety properties; our method can be used to prove timing properties (upper and lower bounds on time) for algorithms that have timing assumptions. Predictable performance is often a desirable characteristic of real-time systems [95]; assertional techniques could be very helpful in proving such performance properties.

Our method involves constructing a multivalued mapping from an

[4]The worst-case elapsed time measured on the clock of any correct process.

automaton representing the given algorithm to another automaton representing the timing requirements. The key to our method is a way of representing a system with timing constraints as an automaton whose state includes predictive timing information. Timing assumptions and timing requirements for the system are both represented in this way, and the mappings we construct map from the "assumptions automaton" to the "requirements automaton".

The formal model used in our work is the *timed automaton* model, adapted from the *time-constrained automaton* model of [78]. A timed automaton is a pair (A, b), consisting of an *I/O automaton* A ([71, 72]), together with a *boundmap* b, which is a formal description of the timing assumptions for the components of the system. We have introduced the notion of a *timing condition* to state upper and lower bounds on the difference between the times at which certain events or states appear in an execution; the conditions imposed by a boundmap are timing conditions of a particular kind. An automaton and a set of timing conditions (in particular, a timed automaton) generate a set of *timed executions* and a corresponding set of *timed behaviors.*

While convenient for specifying timing assumptions and requirements, timed automata are not directly suited for carrying out assertional proofs about timing properties, because timing constraints are described by specially-defined timing conditions and are not part of the automaton itself. We have therefore introduced a way of incorporating timing conditions into an automaton definition. For a given timed automaton A, and a set $\mathcal{U}$ of timing conditions, the automaton *time*$(A,\mathcal{U})$ is defined to be an ordinary I/O automaton (not a timed automaton) whose state includes predictive information describing the first and last times at which various events can next occur; this information is designed to enforce the timing conditions in $\mathcal{U}$.

The timing requirements to be proved for an algorithm described as a timed automaton, (A, b), are described as a set of timing conditions, $\mathcal{U}$, for A. The *requirements automaton* is defined to be *time*$(A,\mathcal{U})$. Thus, predictive information about the first and last times at which certain events of interest can next occur are built into the state of the requirements automaton.

The problem of showing that a given algorithm (A, b) satisfies the timing requirements is then reduced to that of showing that any behavior of the automaton $time(A, b)$ is also a behavior of $time(A, \mathcal{U})$. This is done using invariant assertion techniques; in particular, one demonstrates a multivalued mapping from $time(A, b)$ to $time(A, \mathcal{U})$.

We have applied our technique to three examples. The first example is a simplified version of the timing-dependent resource granting system described in Section 2. We have given careful proofs of upper and lower bounds on the amount of time prior to the first *GRANT* event and in between each successive pair of *GRANT* events. The second example is a system consisting of a "line" of processes, in which each process relays a signal received from the process at its left to the process at its right. We have given careful proofs of upper and lower bounds on the time to propagate a signal from the left end to the right end of the line. The third, more complicated example involves one process incrementing a counter until another process modifies a flag, and then decrementing this counter. When the counter reaches 0, a *DONE* action occurs. We have given careful proofs of upper and lower bounds on the time until a *DONE* occurs.

The mappings we provide for all three of these examples have a particularly interesting and simple form—a set of inequalities relating the time bounds to be proved to those that can be computed from the state. These inequalities contain information about how the bounds are to be satisfied.

Technically, mapping techniques of the sort used in this paper are only capable of proving safety properties, but not liveness properties. Timing properties have aspects of both safety and liveness. A timing lower bound asserts that an event cannot occur before a certain amount of time has elapsed; a violation of this property is detectable after a finite prefix of a timed execution, and so a timing lower bound can be regarded as a safety property. A timing upper bound asserts that an event must occur before a certain amount of time has elapsed. This can be regarded as making two separate claims: that the designated amount of time does in fact elapse (a liveness property), and that that amount of time cannot elapse without the event having occurred (a safety property). In our work, we have *assumed* the liveness property that time increases without bound, so that all the remaining properties that need to be proved in order to prove either upper or lower time bounds are safety properties. Thus, our mapping technique provides complete proofs for timing properties without requiring any special techniques (e.g., variant functions or temporal logic methods) for arguing liveness.

We have shown that this method is *complete*: if every behavior of (A, b) is also a behavior of $time(A, \mathcal{U})$ then is there necessarily a strong possibilities mapping (in the form of inequalities) from $time(A, b)$ to $time(A, \mathcal{U})$. Related completeness results for the usage of refinement mappings to prove properties of non timing-based algorithms were proved in [1] and [77].

There has been some prior work on using assertional reasoning to prove timing properties. In particular, Hasse [43], Shankar and Lam [92], Tel [97], Schneider [90], Lewis [63] and Shaw [93] have all developed models for timing-based systems that incorporate time information into the state, and have used invariant assertions to prove timing properties. In [97] and [63], in fact, the information that is included is similar to ours in that it is also predictive timing information (but not exactly the same information as ours). None of this work has been based on mappings, however. Several other, quite different formal approaches to proving timing properties have also been developed. Some representative papers describing these other methods are [11], [52], [48], [45], [100], [49], and [35].

(The work described in this section appears in [68].)

6 Time complexity of asynchronous resilient algorithms

In shared-memory distributed systems, some number n of independent asynchronous processes communicate by reading and writing to shared memory. *Wait-free* algorithms have been proposed as a mechanism for computing in the face of variable speeds and failures: a wait-free algorithm guarantees that each nonfaulty process terminates regardless of the speed and failure of other processes ([44, 58]). The design of wait-free shared-memory algorithms has recently been a very active area of research (see, e.g., [3, 22, 44, 58, 64, 83, 84, 89, 98]).

We have studied the *time complexity* of wait-free and non-wait-free algorithms in "normal" executions, where no failures occur and processes operate at approximately the same speed. We have selected this particular subset of the executions for making the comparison, because it is only reasonable to compare the behavior of the algorithms in cases where both are required to terminate. Since wait-free algorithms terminate even when some processes fail, while non-wait-free algorithms may fail to terminate in this case, the comparison should only be made in executions in which no process fails, i.e., in *failure-free* executions. The time measure we have used is the one introduced in [54, 55], and used to evaluate the time complexity of asynchronous algorithms, in, e.g., [2, 15, 69, 70, 85]. To summarize, we are interested in measuring the time cost imposed by the wait-free property, as measured in terms of extra computation time in the most normal (failure-free) case.

We have addressed the general question by considering a specific problem—the *approximate agreement* problem studied, for example, in [23, 30, 31, 73]; we have studied this problem in the context of a particular shared-memory primitive—single-writer multi-reader atomic registers. In this problem, each process starts with a real-valued input, and (provided it does not fail) must eventually produce a real-valued output. The outputs must all be within a given distance ε of each other, and must be included within the range of the inputs. This problem, a weaker variant of the well-studied problem of distributed consensus (e.g., [33, 60]), is closely related to the important problem of synchronizing local clocks in a distributed system.

Approximate agreement can be achieved very easily if waiting is allowed, by having a designated process write its input to the shared memory; all other processes wait for this value to be written and adopt it as their outputs. In terms of the time measure described above, it is easy to see that the time complexity of this algorithm is constant—independent of n, the range of inputs and ε. On the other hand, there is a relatively simple wait-free algorithm for this problem which is based on successive averaging of intermediate values. The time complexity of this algorithm depends linearly on n, and logarithmically on the size of the range of input values and on $1/\varepsilon$. A natural question to ask is whether the time complexity of this algorithm is optimal for wait-free approximate agreement algorithms.

Our first major result is an algorithm for the special case where $n = 2$, whose time complexity is constant, i.e., it does *not* depend on the range of inputs or on ε. The algorithm uses a novel method of overcoming the uncertainty that is inherent in an asynchronous environment, without resorting to synchronization points (cf. [39]) or other waiting mechanisms (cf. [15]): this method involves ensuring that the two processes base their decisions on information that is approximately, but not exactly, the same.

Next, using a powerful technique of integrating wait-free (but slow) and non-wait-free (but fast) algorithms, together with an $O(\log n)$ wait-free input collection function, we generalize the key ideas of the 2-process algorithm to obtain our second major result: a wait-free algorithm for approximate agreement whose time complexity is $O(\log n)$. Thus, the time complexity of this algorithm does not depend on either the size of the range of input values or on ε, but it still depends on n, the number of processes.

At this point, it was natural to ask whether the logarithmic dependence on n is inherent for wait-free approximate agreement algorithms, or whether, on the other hand, there is a constant-time wait-free algorithm (independent of n). Our third major result shows that the $\log n$ dependency is inherent: any wait-free algorithm for approximate agreement has time complexity at least $\log n$.[5] This implies an $\Omega(\log n)$ time separation between the non-wait-free and wait-free computation models.

We note that the constant time 2-process algorithm behaves rather badly if one of the processes fails. The *work* performed in an execution of an algorithm is the total number of atomic operations performed in that execution by all processes before they decide. We have proved a tradeoff between the time complexity of and the work performed by any wait-free approximate agreement algorithm. We have shown that for *any* wait-free approximate agreement algorithm for 2 processes, there exists an execution in which the work exhibits a nontrivial dependency on ε and the range of inputs.

In practice, the design of distributed systems is often geared towards optimizing the time complexity in "normal executions," i.e., executions where no failures occur and processes run at approximately the same pace, while building in safety provisions to protect against failures (cf. [61]). Our results indicate that, in the asynchronous shared-memory

[5]The lower bound is attained in an execution where processes run synchronously and no process fails.

setting, t
must resu
situation
tems that
are *early-*
still termi
imposed b
ple, in [14
systems,
in failure-

Recen
chronous
issues, by
tolerance
combinati
wait-free
has not p

(The

7 Fu

Our futur
for proble
we are cu

- Mor
excl
- A p
- Prol
- Exte
case
cur.
- The

ırcoming the
ənt, without
mechanisms
es base their
ly, the same.

e (but slow)
(log n) wait-
.he 2-process
.lgorithm for
. Thus, the
ər the size of
the number

hmic depen-
; algorithms,
ait-free algo-
ıat the log n
imate agree-
ł(log n) time
.tion models.

haves rather
an execution
ɔerformed in
ave proved a
ırmed by any
ı that for *any*
, there exists
ndency on ε

ared towards
., executions
ely the same
; failures (cf.
ıred-memory

synchronously

setting, there are problems for which building in such safety provisions *must* result in performance degradation in the normal executions. This situation contrasts with that occurring, for example, in synchronous systems that solve the distributed consensus problem. In that setting, there are *early-stopping* algorithms (e.g., [25, 28, 79]) that tolerate failures, yet still terminate in *constant* time when no failures occur. The exact cost imposed by fault-tolerance on normal executions, was studied, for example, in [14, 28, 79]. It has been shown, for synchronous message passing systems, that non-blocking commit protocols take twice as much time, in failure-free executions, as blocking protocols ([29]).

Recent work has addressed the issue of adapting the usual synchronous shared-memory PRAM model to better reflect implementation issues, by reducing synchrony ([15, 16, 39, 81, 74]) or by requiring fault-tolerance ([50, 51]). To the best of our knowledge, the impact of the *combination* of asynchrony and fault-tolerance (as exemplified by the wait-free model) on the time complexity of shared-memory algorithms has not previously been studied.

(The work described in this section appears in [6].)

7 Further research

Our future research plans include more work on upper and lower bounds for problems of interest in real-time computing. Some specific problems we are currently working on are as follows.

- More complex resource allocation problems than simple mutual exclusion.
- A problem of probabilistic processor fault diagnosis.
- Probabilistic versions of agreement problems.
- Extensions of the consensus problem described in Section 4 to the case where more severe faults than simple stopping faults can occur.
- The problem of determining the power of a bounded capacity link.

We also plan to continue our work on modeling and verification. Specifically, this includes the following.

- An attempt to relate recent work on process algebraic approaches to reasoning about real-time systems to state-machine models such as [78].
- An attempt to develop timing analysis techniques for randomized algorithms.
- An effort to apply the mapping method to verify substantial algorithms of interest in the fields of communication and/or real-time processing. A particular example we are working on is a new link-state packet distribution algorithm proposed as a standard by DEC.
- An effort to simplify our mapping method described in Section 5.

7.0.1 Acknowledgements:

We owe great thanks to the other researchers who worked with us on these papers: Cynthia Dwork, Marios Mavronicolas, Nir Shavit and Larry Stockmeyer. We also thank Nancy Leveson, Michael Merritt, Stephen Ponzio, Mark Tuttle and Jennifer Welch for many helpful discussions and for their reading of drafts of our papers.

References

[1] M. Abadi and L. Lamport, "The existence of refinement mappings," DEC SRC Research Report 29, August 1988.

[2] E. Arjomandi, M. J. Fischer and N. Lynch, "Efficiency of synchronous versus asynchronous distributed systems," *Journal of the ACM,* Vol. 30, No. 3 (July 1983), pp. 449–456.

[3] J. Aspnes and M. Herlihy, "Fast randomized consensus using shared memory," *Journal of Algorithms*, September 1990, to appear.

[4] H.
on
tain

[5] H.
con
Rea
cal
MIT

[6] H.
fast
tion

[7] H.
sem
Cor
199

[8] B.
nal

[9] J.
Tec

[10] P.
trib
Cor

[11] A.
prog
tem
ber

[12] F.
Fro
Int.
Res

[13] B.
tole
ples

ion. Specif-

approaches
nodels such

randomized

ıstantial al-
ınd/or real-
on is a new
standard by

ı Section 5.

with us on
Shavit and
ael Merritt,
helpful dis-

ement map-

ency of syn-
ournal of the

ensus using
1990, to ap-

[4] H. Attiya, C. Dwork, N. .A. Lynch and L. J. Stockmeyer, "Bounds on the time to reach agreement in the presence of timing uncertainty," in preparation.

[5] H. Attiya and N. A. Lynch, "Time bounds for real-time process control in the presence of timing uncertainty," *Proc. 10th IEEE Real-Time Systems Symposium,* 1989, pp. 268–284. Also, Technical Memo MIT/LCS/TM-403, Laboratory for Computer Science, MIT, July 1989.

[6] H. Attiya, N. Lynch and N. Shavit, "Are wait-free algorithms fast?" to appear in *the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS),* October 1990.

[7] H. Attiya and M. Mavronicolas, "Efficiency of asynchronous vs. semi-synchronous networks," to appear in *the 28th annual Allerton Conference on Communication, Control and Computing,* October 1990.

[8] B. Awerbuch, "The complexity of network synchronization," *Journal of the ACM,* Vol. 32, No. 4 (1985), pp. 804–823.

[9] J. C. M. Baeten and J. A. Bergstra, *Real time process algebra,* Technical Report P8916b, University of Amsterdam, March 1990.

[10] P. Berman, J. A. Garay and K. J. Perry, "Towards optimal distributed consensus," *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 410–415.

[11] A. Bernstein and P. Harter, Jr. "Proving real-time properties of programs with temporal logic," *Proc. 8th Symp. on Operating System Principles,* Operating Systems Review, Vol. 15, No. 5 (December 1981), pp. 1–11.

[12] F. Cristian, H. Aghili, R. Strong and D. Dolev, "Atomic broadcast: From simple message diffusion to Byzantine agreement," *Proc. 15th Int. Conf. on Fault Tolerant Computing,* 1985, pp. 1–7. Also, IBM Research Report RJ5244, revised October 1989.

[13] B. A. Coan, "A communication-efficient canonical form for fault-tolerant distributed protocols," *Proc. 5th ACM Symp. on Principles of Distributed Computing*, 1986, pp. 63–72.

[14] B. A. Coan and C. Dwork, "Simultaneity is harder than agreement," *Proc. 5th IEEE Symp. on Reliability in Distributed Software and Database Systems*, 1986, pp. 141–150.

[15] R. Cole and O. Zajicek, "The APRAM: Incorporating asynchrony into the PRAM model," *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures,* 1989, pp. 169–178.

[16] R. Cole and O. Zajicek, "The expected advantage of asynchrony," *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures,* 1990, to appear.

[17] J. E. Coolahan and N. Roussopoulus, "Timing requirements for time-driven systems using augmented Petri nets," *IEEE Transactions on Software Engineering,* Vol. SE-9, No. 5 (September 1983), pp. 603–616.

[18] B. Dasarathy, "Timing constraints of real-time systems: Constructs for expressing them, methods for validating them," *IEEE Transactions on Software Engineering,* Vol. SE-11, No. 1 (January 1985), pp. 80–86.

[19] R. DeMillo, N. A. Lynch and M. Merritt, "Cryptographic protocols," *Proc. 14th Annual ACM Symp. on Theory of Computing,* May 1982, pp. 383–400.

[20] D. Dolev, C. Dwork and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *Journal of the ACM,* Vol. 34, No. 1 (January 1987), pp. 77–97.

[21] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch and H. R. Strong, "Efficient byzantine agreement without authentication," *Information and Control*, Vol. 52 (1982), pp. 257–274.

[22] D. Dolev, E. Gafni and N. Shavit, "Toward a non-atomic era: ℓ-exclusion as a test case," *Proc. 20th ACM Symp. on the Theory of Computing*, 1988, pp. 78–92.

[23] D. Dolev, N. Lynch, S. Pinter, E. Stark and W. Weihl, "Reaching approximate agreement in the presence of faults," *Journal of the ACM,* Vol. 33, No. 3, 1986, pp. 499–516.

[24] D. Do
impos
puter

[25] D. Do
imme
Scien

[26] D. Do
tine a
(Nove

[27] C. Dw
ence o
pp. 28

[28] C. Dw
in by
on Th
Kaufm
Comp

[29] C. Dw
ment,
puting

[30] A. Fe
agreem
Comp

[31] A. Fe
ACM
64–76.

[32] M. Fis
interac
No. 4

[33] M. Fi
tribute
Vol. 3

[24] D. Dolev, J. Halpern and H. R. Strong, "On the possibility and impossibility of achieving clock synchronization." *Journal of Computer and Systems Sciences,* Vol. 32, No. 2 (1986) pp. 230–250.

[25] D. Dolev, R. Reischuk, and H. R. Strong, "Eventual is earlier than immediate," *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, 1982, pp. 196–203.

[26] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal on Computing,* Vol. 12, No. 3 (November 1983), pp. 656–666.

[27] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, Vol. 35 (1988), pp. 288–323.

[28] C. Dwork and Y. Moses, "Knowledge and common knowledge in byzantine environments I: Crash failures," *Proc. 1st Conf. on Theoretical Aspects of Reasoning About Knowledge*, Morgan-Kaufmann, Los Altos, CA, 1986, pp. 149–170; *Information and Computation*, to appear.

[29] C. Dwork and D. Skeen, "The inherent cost of nonblocking commitment," *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, 1983, pp. 1–11.

[30] A. Fekete, "Asymptotically optimal algorithms for approximate agreement," *Proc. 5th ACM Symp. on Principles of Distributed Computing*, 1986, pp. 73–87.

[31] A. Fekete, "Asynchronous approximate agreement," *Proc. 6th ACM Symp. on Principles of Distributed Computing*, 1987, pp. 64–76.

[32] M. Fischer and N. Lynch, "A lower bound for the time to assure interactive consistency," *Information Processing Letters,* Vol. 14, No. 4 (June 1982), pp. 183–186.

[33] M. Fischer, N. Lynch and M. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM,* Vol. 32, No. 2 (1985), pp. 374–382.

[34] M. W. Franklin and A. Gabrielian, "A transformational method for verifying safety properties in real-time systems," in *Proc. 10th IEEE Real-Time Systems Symp.*, pp. 112–123, December 1989.

[35] A. Gabrielian and M. W. Franklin, "State-based specification of complex real-time systems," in *Proc. IEEE Real-Time Systems Symp.*, 1988, pp. 2–11.

[36] A. Gabrielian and M. W. Franklin, "Multi-level specification and verification of real-time software," Technical Report 89-14, Tomson-CSF, Inc., July 1989.

[37] R. Gerth and A. Boucher, "A timed failure semantics for extended communicating processes," In *Proc. ICALP '87*, Springer-Verlag Lecture Notes in Computer Science #267, 1987.

[38] R. Gerber and I. Lee, "The formal treatment of priorities in real-time computation." In *Proc. 6th IEEE Workshop on Real-Time Software and Operating Systems*, 1989.

[39] P. Gibbons, "Towards better shared memory programming models," *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, 1989, pp. 169–178.

[40] D. W. Gillies and J. W.-S. Liu, "Greed in resource scheduling," *Proc. 10th IEEE Real-Time Systems Symposium*, 1989, pp. 285–294.

[41] V. Hadzilacos, *Issues of fault tolerance in concurrent computations*, Ph.D. Thesis, Harvard University, June 1984. Technical Report TR–11–84, Department of Computer Science, Harvard University.

[42] J. Halpern, N. Megiddo and A. A. Munshi, "Optimal precision in the presence of uncertainty." *Journal of Complexity*, Vol. 1 (1985), pp. 170–196.

[43] V. H. Hasse, "Real-time behavior of programs," *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 5 (September 1981), pp. 494–501.

[44] M. Herlihy, "Wait-free implementations of concurrent objects," *Proc. 7th ACM Symp. on Principles of Distributed Computing*, 1988, pp. 276–290.

[45] J. Hoo
messag
Compu
1987.

[46] C. Hui
real-tir
Proc. 1
1987, p

[47] F. Jah
real-tir
Vol. SI

[48] F. Jah
analysi
ers, Vo

[49] F. Jah
of Mod
Symp.,

[50] P. Kar
can be
tribute

[51] Z. Kee
compu
1990,

[52] R. Ko
gramn
Symp.

[53] R. Ko
and S
tribut
3 (Dec

[54] L. Lar
Inforr

[45] J. Hooman, *A compositional proof theory for real-time distributed message passing,* TR. 4-1-1(1), Department of Mathematics and Computer Science, Eindhoven University of technology, March 1987.

[46] C. Huizing, R. Gerth, and W. P. deRoever, "Full abstraction of a real-time denotational semantics for an OCCAM-like language," in *Proc. 14th ACM Symp. on Principles of Programming Languages,* 1987, pp. 223–237.

[47] F. Jahanian and A. Mok, "Safety analysis of timing properties in real-time systems," *IEEE Transactions on Software Engineering,* Vol. SE-12, No. 9 (September 1986), pp. 890–904.

[48] F. Jahanian and A. Mok, "A graph-theoretic approach for timing analysis and its implementation," *IEEE Transactions on Computers,* Vol. C-36, No. 8 (August 1987), pp. 961–975.

[49] F. Jahanian and D. A. Stuart, "A method for verifying properties of Modechart specifications," in *Proc. IEEE Real-Time Systems Symp.,* 1988, pp. 12–21.

[50] P. Kanellakis and A. Shvartsman, "Efficient parallel algorithms can be made robust," *Proc. 8th ACM Symp. on Principles of Distributed Computing,* 1989, pp. 211–221.

[51] Z. Kedem, K. Palem and P. Spirakis, "Efficient robust parallel computations," *Proc. 22nd ACM Symp. on Theory of Computing,* 1990, pp. 138–148.

[52] R. Koymans, J. Vytopil and W. P. deRoever, "Real-time programming and asynchronous message passing," in *Proc. 2nd ACM Symp. on Principles of Distributed Computing,* 1983, pp. 187–197.

[53] R. Koymans, R. K. Shyamasundar, W. P. deRoever, R. Gerth, and S. Arun-Kumar, "Compositional semantics for real-time distributed computing," *Information and Computation,* Vol. 79, No. 3 (December 1988), pp. 210–256.

[54] L. Lamport, "The synchronization of independent processes," *Acta Informatica,* Vol. 7, No, 1 (1976), pp. 15–34.

[55] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Transactions on Software Engineering,* Vol. SE-3, No. 2 (March 1977) pp. 125–143.

[56] L. Lamport, "Time, clocks and the ordering of events in distributed systems." *Communications of the ACM,* Vol. 21, No. 7 (July 1978), pp. 558–565.

[57] L. Lamport, "Specifying concurrent program modules," *ACM Trans. on Programming Languages and Systems,* Vol. 5, No. 2 (April 1983), pp. 190–222.

[58] L. Lamport, "On interprocess communication. parts I and II" *Distributed Computing 1, 2* 1986, 77–101.

[59] L. Lamport and M. J. Fischer, "Byzantine generals and transaction commit protocols," Tech. Report Op. 62, SRI International, Menlo Park, CA, 1982.

[60] L. Lamport, R. Shostak and M. Pease, "The byzantine generals problem," *ACM Transaction on Prog. Lang. and Sys.,* Vol. 4, No. 3 (July 1982), pp. 382–401.

[61] B. Lampson, "Hints for computer system design", in *Proc. 9th ACM Symposium on Operating Systems Principles,* 1983, pp. 33–48.

[62] N. Leveson and J. Stolzy, "Safety analysis using Petri Nets," *IEEE Transactions on Software Engineering,* Vol. SE-13, No. 3 (March 1987), pp. 386–397.

[63] H. R. Lewis, "Finite-state analysis of asynchronous circuits with bounded temporal uncertainty," Techincal Report TR-15-89, Aiken Computation Laboratory, Harvard University.

[64] M. Li, J. Tromp and P. M.B. Vitanyi, "How to share concurrent wait-free variables," *ICALP 1989.* Expanded version: Report CS-R8916, CWI, Amsterdam, April 1989.

[65] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *Journal of the ACM,* Vol. 20, No. 1 (1973), pp. 46-61.

[66] J. Lu
synch
gust/

[67] N. Ly
Adva

[68] N. A.
erties
ples c
Techn
Scien

[69] N. Ly
ment
Vol.

[70] N.
man,
Com

[71] N. L
tribu
Dist
able
pute

[72] N. L
toma
cal M
Mass

[73] S. M
cisio
ciple

[74] C. M
algor
port
forni

s programs,"
SE-3, No. 2

n distributed
' (July 1978),

ules," *ACM*
'ol. 5, No. 2

and II" *Dis-*

d transaction
ional, Menlo

tine generals
., Vol. 4, No.

in *Proc. 9th*
983, pp. 33–

Nets," *IEEE*
Io. 3 (March

ous circuits
rt TR-15-89,

e concurrent
Report CS-

or multipro-
of the ACM,

[66] J. Lundelius and N. Lynch, "An upper and lower bound for clock synchronization," *Information and Control,* Vol. 62, Nos. 2/3 (August/September 1984), pp. 190–204.

[67] N. Lynch, "Concurrency control for resilient nested transactions," *Advances in Computing Research,* Vol. 3, 1986, pp. 335–373.

[68] N. A. Lynch and H. Attiya, "Using mappings to prove timing properties," proceedings of *the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC),* 1990, pp. 265–280. Also, Technical Memo MIT/LCS/TM-412.b, Laboratory for Computer Science, MIT, December 1989.

[69] N. Lynch and M. Fischer, "On describing the behavior and implementation of distributed systems," *Theoretical Computer Science,* Vol. 13, No. 1 (January 1981), pp. 17-43.

[70] N. Lynch and K. Goldman, *Lecture notes for 6.852.* MIT/LCS/RSS-5, Laboratory for Computer Science, MIT, 1989.

[71] N. Lynch and M. Tuttle, "Hierarchical correctness proofs for distributed algorithms," in *Proc. 7th ACM symp. on Principles of Distributed Computing,* 1987, pp. 137–151. Expanded version available as Technical Report MIT/LCS/TR-387, Laboratory for Computer Science, MIT, April 1987.

[72] N. Lynch and M. Tuttle, "An introduction to input/output automata," *CWI-Quarterly,* Vol. 2, No. 3, 1989. Also, Technical Memo, MIT/LCS/TM-373, Laboratory for Computer Science Massachusetts Institute of Technology, November 1988.

[73] S. Mahaney and F. Schneider, "Inexact agreement: Accuracy, precision, and graceful degradation," *Proc. 4th ACM Symp. on Principles of Distributed Computing*, 1985, pp. 237–249.

[74] C. Martel, A. Park and R. Subramonian, "Optimal asynchronous algorithms for shared memory parallel computers," Technical Report CSE-89-8, Division of Computer Science, University of California, Davis, July 1989.

[75] C. Martel, R. Subramonian and A. Park, "Asynchronous PRAMs are (almost) as good as synchronous PRAMs," to appear in *the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS),* October 1990.

[76] M. Merritt, "Notes on the Dolev-Strong Lower Bound for Byzantine Agreement," unpublished manuscript, 1985.

[77] M. Merritt, "Completeness theorems for automata," REX Workshop, May 1989.

[78] M. Merritt, F. Modugno and M. Tuttle, "Time constrained automata," unpublished manuscript, November 1988. Revised: Aguest 1990.

[79] Y. Moses and M. R. Tuttle, "Programming simultaneous actions using common knowledge," *Algorithmica,* Vol. 3 (1988), pp. 121–169.

[80] Y. Moses and O. Waarts, "Coordinated traversal: $(t+1)$-round byzantine agreement in polynomial time," *Proc. 29th IEEE Symp. on Foundations of Computer Science*, 1988, pp. 246–255.

[81] N. Nishimura, "Asynchronous shared memory parallel computation," *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures,* 1990, to appear.

[82] M. Pease, R. Shostak and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM,* Vol. 27, No. 2 (1980), pp. 228–234.

[83] G. Peterson, "Concurrent reading while writing," *ACM Transactions on Programming Languages and Systems,* Vol. 5, No. 1 (January 1983), pp. 46–55.

[84] G. Peterson, and J. Burns, "Concurrent reading while writing II: The multi-writer case," *Proc. 28th IEEE Symp. on Foundations of Computer Science,* 1987, pp. 383–392.

[85] G. Peterson and M. Fischer, "Economical solutions for the critical section problem in a distributed system," *Proc. 9th ACM Symp. on Theory of Computing,* 1977, pp. 91–97.

[86] R. R
thesis

[87] K. R
sched
IEEE
pp. 1

[88] G. M
seque

[89] R. Sc
MIT,

[90] F. B.
tions
Work

[91] J. Si
mode
IFIP
terda

[92] A. U
tems
tribu

[93] A. C
ware
No.

[94] B. Si
chro

[95] J. St
para
tems

[96] R. S
atom
Sym

[86] R. Rajkumar, *Task synchronization in real-time systems,* Ph.D. thesis, Carnegie-Mellon University, Augurst 1989.

[87] K. Ramamritham, J. Stankovic and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers,* Vol. C-38, No. 8 (August 1989), pp. 1110–1123.

[88] G. M. Reed and A. W. Roscoe, "A timed model for communicating sequential processes," in *ICALP '86.*

[89] R. Schaffer, "On the correctness of atomic multi-writer registers," MIT/LCS/TM-364, June 1988.

[90] F. B. Schneider, "Real-time reliable systems project," in *Foundations of Real-Time Computing Research Initiative,* ONR Kickoff Workshop, November 1988, pp. 28–32.

[91] J. Sifakis, "Petri nets for performance evaluation, in measuring, modeling and evaluating computer systems," in *Proc. 3rd Symp. IFIP Working Group 7.3,* H. Beilner and E. Gelenbe (eds.), Amsterdam, The Netherlands, North-Holland, 1977, pp. 75–93.

[92] A. U. Shankar and S. L. Lam, "Time-dependent distributed systems: Proving safety, liveness and real-time properties," *Distributed Computing,* Vol. 2, pp. 61–79, 1987.

[93] A. C. Shaw, "Reasoning about time in higher-level language software," *IEEE Transactions on Software Engineering,* Vol. SE-15, No. 7 (July 1989), pp. 875–889.

[94] B. Simons, J. L. Welch and N. Lynch, "An overview of clock synchronization," IBM Technical Report RJ 6505, October 1988.

[95] J. Stankovic and K. Ramamritham, "The SPRING kernel: A new paradigm for real-time operating systems," *ACM Opertating Systems Reviews,* Vol 23, No. 3 (July 1987), pp. 54–71.

[96] R. Strong, D. Dolev and F. Cristian, "New latency bounds for atomic broadcast," to appear in *11th IEEE Real-Time Systems Symposium,* 1990.

[97] G. Tel, "Assertional verification of a timer based protocol," in *ICALP '88,* Lecture Notes in Computer Science 317, Springer-Verlag, pp. 600–614.

[98] P. Vitanyi and B. Awerbuch, "Atomic shared register access by asynchronous hardware," *Proc. 27th IEEE Symp. on Foundations of Computer Science*, pp. 233–243, 1986.

[99] J. L. Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Information and Computation,* Vol. 77, No. 1 (April 1988), pp. 1–36.

[100] A. Zwarico, *Timed acceptence: an algebra of time dependent computing,* Ph.D. thesis, Dept. of Computer and Information Science, University of Pennsylvania, 1988.

[101] A. Zwarico, I. Lee and R. Gerber, "A complete axiomatization of real-time processes," submitted for publication.

HMS M
Verifi

An ove
present
that are
ing aut
pared t
fining t
discret
framew
ments
ing suc
formed
and a n
are pre
is prese
ificatio
time d
verifica
ing pro
ness–p
ample
schedu
tempo

INTRO

Various te
have been
studies ha