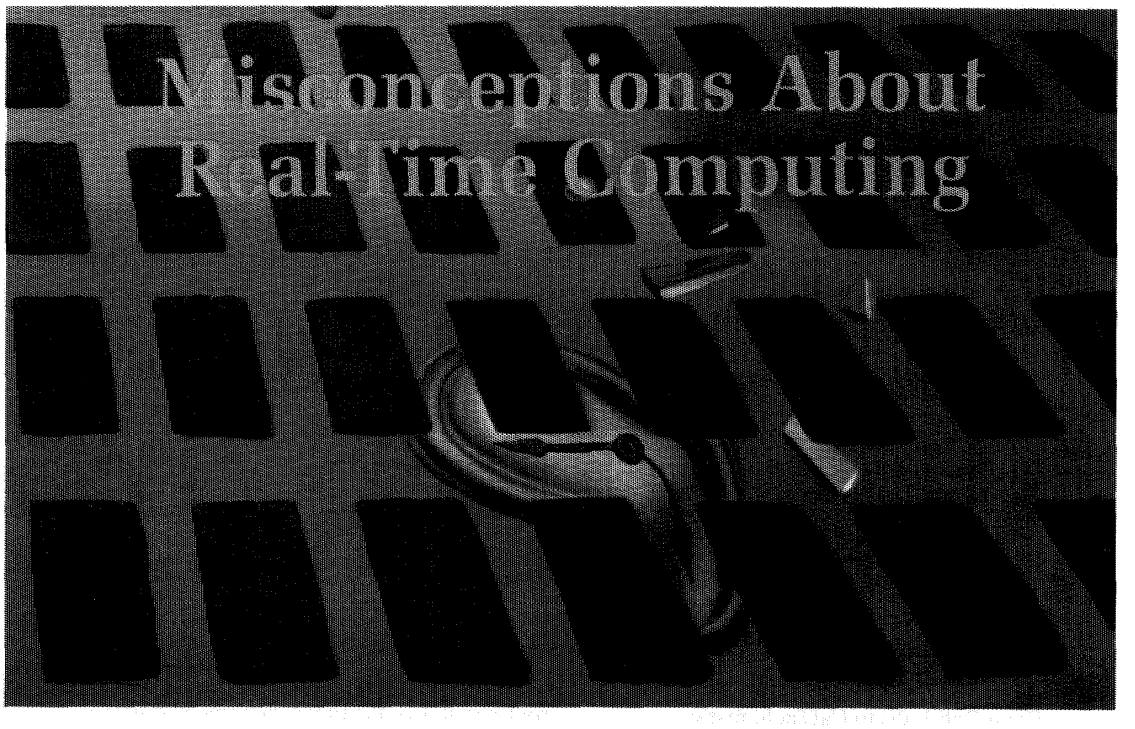

A Serious Problem for Next-Generation Systems

John A. Stankovic*

University of Massachusetts

Real-time computing is a wide-open research area of intellectually challenging computer science problems with direct payoffs to current technology. But results have been few, and designers presently have little that would enable them to handle the timing constraints of real-time systems effectively. Furthermore, not enough emphasis is being placed on building the proper scientific underpinnings to achieve the needed results.[1] Worse yet, many researchers, technical managers, and government contract monitors have serious misconceptions about real-time computing—misconceptions with serious ramifications.

This article has three major objectives:

- to state and then dispel the most common misconceptions about real-time computing,
- to briefly discuss the fundamental scientific issues of real-time computing, and
- to encourage increased research in real-time systems.

Development of next-generation real-time systems must be highly focused and coordinated because their failure to meet timing constraints will result in economic, human, and ecological catastrophes.

*The author uses ideas presented at the Carnegie Mellon University Workshop on Fundamental Issues in Distributed Real-Time Systems, March 1987, and subsequently elaborated on by many individuals (see Acknowledgments).

What is real-time computing?

In real-time computing the correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced. Real-time computing systems play a vital role in our society, and they cover a spectrum from the very simple to the very complex. Examples of current real-time computing systems include the control of laboratory experiments, the control of automobile engines, command-and-control systems, nuclear power plants, process control plants, flight control systems, space shuttle and aircraft avionics, and robotics. The more complicated real-time systems are expensive to build and their timing constraints are verified with ad hoc techniques, or with costly and extensive simulations. Minor changes in the system result in another round of extensive testing. Different system components are

0018-9162/88/1000-0010$01.00 © 1988 IEEE

extremely difficult to integrate and consequently add to overall system cost. Millions (even billions) of dollars are being spent (wasted) by industry and government to build today's real-time systems. Current brute force techniques will not scale to meet the requirements of guaranteeing real-time constraints of next-generation systems.

Next-generation real-time systems will be in application areas similar to those of current systems. However, the systems will be more complex: They will be distributed and capable of exhibiting intelligent, adaptive, and highly dynamic behavior. They will also have long lifetimes. Moreover, catastrophic consequences will result if the logical or timing constraints of the systems are not met. Examples of these more sophisticated systems are the autonomous land rover, controllers of robots with elastic joints, and systems found in intelligent manufacturing, the space station, and undersea exploration.

Two major forces are pushing real-time systems into the next generation: their need for artificial intelligence capabilities and the rapid advance in hardware. These forces are exacerbating the difficult scientific and engineering problems faced in building real-time systems. They add complex entities that must be integrated into current and future applications, but the required design, analysis, and verification techniques for such integration have not kept pace. For example, hardware (and software) technology has made distributed computing and multiprocessing a reality, and soon there will be many networks of multiprocessors. However, almost no fundamental or scientific work has been done in designing and verifying a real-time application's timing requirements when that application is distributed across a network.

As another example, AI systems exhibit a great deal of adaptability and complexity, making it impossible to precalculate all possible combinations of tasks that might occur. This precludes use of static scheduling policies common in today's real-time systems. We need new approaches for real-time scheduling in such systems, including on-line guarantees and incremental algorithms that produce better results as a function of available time.

Common misconceptions

Real-time-system design has not attracted the attention from academic computer scientists and basic-research funding agencies that it deserves. This lack of adequate attention is due, at least in part, to some common misconceptions about real-time systems. Let's look at some of them.

There is no science in real-time-system design.

It is certainly true that real-time-system design is mostly ad hoc. This does not mean, however, that a scientific approach is not possible. Most good science grew out of attempts to solve practical problems, and there is plenty of evidence that engineers of real-time systems need help. For example, the first flight of the space shuttle was delayed, at considerable cost, because of a subtle timing bug that arose from a transient CPU overload during system initialization. Can we then develop a scientific basis for verifying that a design is free of such subtle timing bugs? Indeed, the purpose of this article is to introduce some of the technical problems involved in designing reliable real-time systems and point out where a scientific basis is emerging. We are starting to understand what the important problems are in real-time scheduling of resources.[2] Investigations are beginning into the subtleties of including a time metric in system specification methods and semantic theories for real-time programming languages.[3]

Advances in supercomputer hardware will take care of real-time requirements.

Advances in supercomputer design will likely exploit parallel processors to improve system throughput, but this does not mean that timing constraints will be met automatically. Unless the architecture of the computing system is carefully tailored to match that of the application, the processors and their communication subsystems may not be able to handle all of the task load and time-critical traffic. In fact, real-time task-and-communication scheduling problems will likely get worse as more hardware is used.

Realistically, the history of computing shows that the demand for more computing power has always outstripped the supply. If the past is any guide to the future, the availability of more computing power will only open up real-time applications requiring greater functionality, thus exacerbating the timing problems. There is no substitute for intelligent deployment of finite resources. Other important issues exist in real-time-systems design that cannot be resolved by supercomputer hardware alone, as we will see.

Real-time computing is equivalent to fast computing.

The objective of fast computing is to minimize the average response time of a given set of tasks. However, the objective of real-time computing is to meet the individual timing requirement of each task. Rather than being fast (which is a relative term anyway), the most important property of a real-time system should be predictability; that is, its functional and timing behavior should be as deterministic as necessary to satisfy system specifications. Fast computing is helpful in meeting stringent timing specifications, but fast computing alone does not guarantee predictability.

Other factors besides fast hardware or algorithms determine predictability. Sometimes the implementation language may not be expressive enough to prescribe certain timing behavior. For example, the delay statement of Ada puts only a lower bound on when a task is next scheduled; there is no language support to guarantee that a task cannot be delayed longer than a desired upper bound. The scheduling of (or the lack of programmer control over) nondeterministic constructs such as the select statement in Ada is especially troublesome, since timing properties that involve upper bounds cannot be guaranteed by the usual fairness semantics defining such constructs.

Perhaps the best response to those who claim that real-time computing is equivalent to fast computing is to raise the following question: Given a set of demanding real-time requirements and an implementation using the fastest hardware and software possible, how can one show that the specified timing behavior is indeed being achieved? Testing is not the answer. Indeed, for all the laborious testing and simulation effort on the space shuttle, the timing bug that delayed its first flight was discovered the hard way; there was only a 1 in 67 probability that a transient overload during initialization could put the redundant processors out of sync, but it did nevertheless. Predictability, not speed, is the foremost goal in real-time-system design.

Since testing is not the answer to our problems, do we know the answer? Not completely. We do know that a formal verification procedure coupled with testing would be significantly better than what we have now. However, that is not the

entire answer either. In fact, most of the problems enumerated in the upcoming section on the challenge of real-time computing systems must be solved and then used in an integrated fashion.

Real-time programming is assembly coding, priority interrupt programming, and device driver writing.

To meet tight timing constraints, current practice in real-time programming relies heavily on machine-level optimization techniques. These techniques are labor intensive and sometimes introduce additional timing assumptions (unwisely, but as a last resort) on which the correctness of an implementation depends. Reliance on clever hand-coding and difficult-to-trace timing assumptions is a major source of bugs in real-time programming, especially in modifying large real-time programs. A primary objective in real-time-systems research is in fact to automate, by exploiting optimizing transforms and scheduling theory, the synthesis of highly efficient code and customized resource schedulers from timing-constraint specifications. On the other hand, while assembly language programming, interrupt programming, and device driver writing are aspects of real-time computing, they do not constitute open scientific problems—except in their automation.

Real-time-systems research is performance engineering.

An important aspect of real-time-systems research is to investigate effective resource allocation strategies so as to satisfy stringent timing-behavior requirements. The synthesis aspects of real-time-system research can indeed be regarded as performance engineering (but see the next misconception below). The proper design of a real-time system, however, requires solutions to many other interesting problems—for example, specification and verification of timing behavior, and programming-language semantics dealing with time. Certain theoretical problems also involve the use of timing constraints, sometimes implicitly, to ensure correctness. For example, the well-known Byzantine generals problem is unsolvable for totally asynchronous systems but is solvable if the generals can vote in rounds. That a good general must deliver a number of messages within a round according to the voting protocol is a form of timing constraint.

Indeed, the correct functioning of many systems often depends on having an implementation that can perform an operation requiring the satisfaction of certain timing constraints, albeit implicitly specified (for example, in the form of testing an atomic predicate such as determining whether a communication channel is empty). An important problem in real-time-systems research is to investigate the role time plays as a synchronization mechanism; for example, what is the logical power of different forms of timing constraints in solving various coordination problems? If a system must depend on the satisfaction of some timing constraints for its correctness, is there a least-restrictive set of timing constraints sufficient for the purpose? Does the imposition of various timing constraints facilitate more efficient solutions to distributed coordination problems? Such questions certainly go beyond traditional performance engineering.

The problems in real-time-system design have all been solved in other areas of computer science or operations research.

While real-time-system researchers should certainly try to exploit the problem solution techniques developed in more established research areas, there are unique problems in real-time systems that have not been solved in any other area. For example, performance engineering in computer science has been concerned mostly with analyzing the average values of performance parameters, whereas an important consideration in real-time-system design is whether or not some stringent deadlines can be met. Queueing models traditionally use convenient stochastic assumptions that are justified by large populations and stable operating conditions. Analytical results based on these assumptions may be quite useless for some real-time applications. For example, the hot-spot contention phenomenon (a highly nonlinear performance degradation due to slight deviations from uniform traffic in multistage interconnection networks) is likely to be catastrophic for time-critical communication packets. Likewise, the combinatorial scheduling problems in operations research deal mostly with one-shot tasks, that is, each task needs to be scheduled only once, whereas in real-time systems the same task may recur infinitely often, either periodically or at irregular intervals, and may have to synchronize or communicate with many other tasks. The general synthesis problem of arbitrary timing behavior will certainly require new techniques not found in existing literature.

It is not meaningful to talk about guaranteeing real-time performance, because we cannot guarantee that the hardware will not fail and the software is bug free or that the actual operating conditions will not violate the specified design limits.

It is a truism that one can only hope to minimize the probability of failure in the systems one builds (assuming a belief in quantum mechanics). The relevant question, of course, is how to build systems in such a way that we can have as much confidence as possible that they will meet specifications at acceptable costs. In real-time-system design, one should attempt to allocate resources judiciously to make certain that any critical timing constraint can be met with the available resources, assuming that the hardware/software functions correctly and the external environment does not stress the system beyond what it is designed to handle. The fact that the hardware/software may not function correctly or that the operating conditions imposed by the external world may exceed the design limits with a nonzero probability does not give the designer license to *increase* the odds of failure by not trying to allocate resources carefully so as to meet critical timing constraints. We certainly cannot guarantee anything outside our control, but what we can guarantee, we should.

Real-time systems function in a static environment.

Depending on the operating mode, a real-time system may have to satisfy different sets of timing constraints at different times. Thus, an important topic in real-time-systems research is the design of hierarchical or selectable schedulers to make resource allocation decisions for different time granularities. A particularly vexing industrial problem is how to reconfigure systems to accommodate changing requirements so as to create minimal disruption to ongoing operation. It is not uncommon for some real-time-system hardware to be in the field 15 or more years; hence, any design methodology for such systems must not assume a static environment. On the other hand, if the real-time application is small, inexpensive, and unlikely to change, then current static solution techniques work well. However, such real-time applications are trivial compared with those addressed in this article.

The challenge of real-time computing systems

An important approach used in managing large-scale systems is to hierarchically decompose the system into modules that are realizations of the abstract data type model. Although this methodology allows us to reason about the correctness of computation at each level of abstraction, it has no provisions to support reasoning about time and reliability abstractions, two vital aspects of real-time systems. To develop a scientific underpinning for real-time systems, we face the difficult scientific challenge of creating a set of unified theories and technologies that will allow us to reason about the correctness, timeliness, and reliability at each level of abstraction and to combine the results of each level into results for the integrated system.

Building a science of large-scale real-time systems will require new research efforts in many distinct and yet related areas. While each of these areas contains well-developed theories and technologies, none currently contains theories and methods addressing the central issue in real-time systems: a coherent treatment of correctness, timeliness, and fault tolerance in large-scale distributed computations.

The following subsections briefly identify the main research areas that need to be better addressed if we are to solve the problems facing developers of next-generation real-time systems. Of course, many of the problems in these areas are difficult to solve even without worrying about real-time constraints. This article emphasizes the special problems that real-time constraints cause. This material is intended for those not familiar with research in real-time computing, so it is necessarily high level. Readers can find a more detailed and technical discussion of each area in a companion report.[1]

Specification and verification. The fundamental challenge in the specification and verification of real-time systems is how to incorporate the time metric. Methods must be devised for including timing constraints in specifications and for establishing that a system satisfies such specifications. The usual approaches for specifying computing system behavior entail enumerating events or actions that the system participates in and describing orders in which these can occur. It is not clear how to extend such approaches for real-time constraints. Neither is it clear how to extend programming notations to allow the programmer to specify computations that are constrained by real time.

A great challenge lies ahead in the modeling and verification of systems that are subject to timing constraints.

In general, inclusion of a time metric can create subtleties in the semantics of concurrency models (see Reed and Roscoe,[4] for example) and complicate the verification problem. Whereas proof of noninterference is the major verification task in extending sequential systems to concurrent systems on the basis of interleaving, verification of real-time systems will require the satisfaction of timing constraints where those constraints are derived from the environment and implementation.[3] Consequently, a major challenge is to solve the dilemma that verification techniques abstract away from the implementation even though it is the implementation and environment which provide the true timing constraints. The result is that we need a quantitative analysis (deadlines, repetition rates) rather than the qualitative analysis (eventual satisfaction) that is typically handled by current verification techniques. On the other hand, the partial synchrony of real-time systems that results from the presence of timing constraints may open up a whole new class of distributed control algorithms and inspire novel verification techniques. Real-time systems lie somewhere between the relatively well studied fully synchronous and fully asynchronous systems, and there is a great challenge ahead in the modeling and verification of systems that are subject to timing constraints.

In addition, to deduce properties of the whole system from properties of its parts and the way these parts are combined, we must characterize a way to compose the real-time constraints and properties of parts to synthesize them for the whole. For real-time properties, the parts interact in ways that depend on resource constraints. Thus, aspects of the system that are usually ignored when real time is not of concern come into play. This again suggests that a new set of abstractions must be devised for real-time systems. For example, *fairness*, which is frequently used when reasoning about concurrent and distributed systems, is no longer an appropriate abstraction of the way processes are scheduled when real time is a concern.

Another problem that will be encountered is dealing with the state explosion found in verification techniques. Hundreds or even thousands of states are usually required to formally express the state of even a relatively simple system in enough detail so that correctness can be proved. Techniques for abstracting many states into a higher level state are necessary to tackle this problem, which is a difficult one even when timing constraints are not included.

Real-time scheduling theory. While specification and verification concern the integrity of the system with respect to the specification, scheduling theory addresses the problem of meeting the specified timing requirements. Satisfying the timing requirements of real-time systems demands the scheduling of system resources according to some well-understood algorithms so that the timing behavior of the system is understandable, predictable, and maintainable.

Scheduling theory is not restricted to the study of real-time systems or even general computer systems. It also arises in the study of manufacturing systems, transportation systems, process control systems, and so on. However, it is important to realize that real-time-system scheduling problems are different from the scheduling problems usually considered in areas of operations research.[5,6] In most operations research scheduling problems, there is a fixed system having completely specified and static service characteristics. The goal is to find optimal static schedules that minimize the response time for a given task set. Many real-time computing systems lack an incentive for minimizing the response time other than for meeting deadlines. The system is often highly dynamic, requiring on-line, adaptive scheduling algorithms. Such algorithms must be based on heuristics, since these scheduling problems are NP-hard.[2] In these cases the goal is to schedule as many jobs as possible, subject to meeting the task timing requirements. Alternative schedules and/or error handlers are required and

must be integrated with the on-line scheduler.

One primary measure of a scheduling algorithm is its associated processor utilization level below which the deadlines of all the tasks can be met.[7] There are other measures too, such as penalty functions defined according to the number of jobs that miss their deadlines, and a weighted success ratio, which is the percentage of tasks that meet their deadlines weighted by the importance of those tasks. The next subsection discusses some additional real-time scheduling issues in the context of operating systems.

Real-time operating systems. One major focal point for developing next-generation real-time systems is the operating system.[8-10] The operating system must provide basic support for guaranteeing real-time constraints, supporting fault tolerance and distribution, and integrating time-constrained resource allocations and scheduling across a spectrum of resource types, including sensor processing, communications, CPU, memory, and other forms of I/O. Given that the system is distributed, we face a complicated end-to-end timing analysis problem (see Figure 1). In other words, time constraints are applied to collections of cooperating tasks, labeled T_i in the figure, and not *only* to individual tasks.

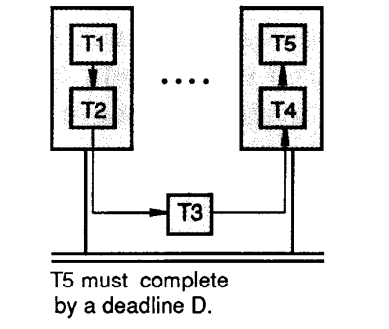

Figure 1. The end-to-end timing problem.

For example, assume that a given node processes sensor hits and determines that a vehicle has entered its area of concern. This node may have to communicate that information to one or more remote nodes and receive replies indicating the appropriate action. The time by which this node must act depends on a time constraint imposed by the velocity of the incoming vehicle. Such a system is distributed, highly dynamic, and operates under strict time constraints. The entire collection of tasks dealing with sensor processing, communication between tasks, and application logic must be accomplished under a single end-to-end timing constraint.

To develop next-generation real-time distributed operating systems suitable for complicated applications such as the space station, teams of robots working in a hazardous environment, or command-and-control applications, at least three major scientific innovations are required.

(1) The *time dimension* must be elevated to a central principle of the system. Time requirements and properties cannot be an afterthought. An especially perplexing aspect of this problem is that most system design and verification techniques are

based on abstraction, which ignores implementation details. This is obviously a good idea; however, in real-time systems, timing constraints are derived from the environment and the implementation. Solving this dilemma is a key scientific issue.

(2) The basic paradigms found in today's general-purpose distributed operating systems must change. The current basis, with the notion of application tasks requesting resources as if they were random processes and the operating system being designed to expect such random inputs, is unacceptable. A new, more deterministic paradigm is needed. In such a paradigm, each operating-system primitive, along with application tasks and their interactions, would be well understood, bounded, and predictable. The interaction between tasks includes invocation interactions, communication interactions, and resource conflict interactions. In the new paradigm the system must be flexible enough to react to a highly dynamic and adaptive environment, but at the same time able to predict and possibly avoid resource conflicts so that timing constraints can be (predictably) met. In other words, the environment may cause an unpredictable combination of events to occur, but the system must be carefully constructed to enable it to react in such a way that at any time during execution the system can predict its capabilities in meeting its deadline. The new paradigm must be based on the delicately balanced notions of flexibility and predictability. One such paradigm is being investigated in the Spring kernel.[9]

(3) A highly integrated and time-constrained resource allocation approach[2] is necessary to adequately address timing constraints, predictability, adaptability, and fault tolerance. Most current scheduling algorithms typically account for one resource at a time and ignore fault tolerance. This is especially true in the present state of real-time scheduling. Independent scheduling of unique resources is not sufficient when attempting to perform time-constrained scheduling. For a task to meet its deadline, we must ensure that resources are available to it *in time* and that the sequencing of events meets precedence constraints.

One can identify many other issues (at all levels of detail) that are critical to real-time operating systems. The need for a global time reference and the ability to scale to larger and larger systems are just two examples.

Real-time programming languages and design methodology. As the complexity of real-time systems increases, high demand will be placed on the programming abstractions provided by languages. Currently available abstractions and languages must evolve with future advances in the specification techniques, logic, and theory underlying real-time systems. Unfortunately, this goal has not been fulfilled in the past. For example, Ada is designed for embedded hard real-time applications and is intended to support static priority scheduling of tasks. However, the definition of Ada tasking allows a high-priority task to wait for a low-priority task for an unpredictable duration. Consequently, when processes are programmed in the form of Ada tasks, the resulting timing behavior is likely to be unpredictable. Building the next generation of real-time systems will require a programming meth-

odology that gives due consideration to the needs of meeting real-time requirements. The important research issues include

- Support for the management of time. First, language constructs should support the expression of timing constraints. For example, Ada tasking should have supported the raising of an exception when a task's deadline is missed. Second, the programming environment should provide the programmer with the primitives to control and to keep track of the resource utilization of software modules. This includes being able to develop programs with predictable performance in terms of absolute time. Finally, language constructs should support the use of sound scheduling algorithms.
- Schedulability check. Given a set of well-understood scheduling algorithms, schedulability analysis allows us to determine if the timing requirements can be met. With proper support for the management of time, it may be possible to perform schedulability checks at compile time. This idea is similar to the type-checking concept.
- Reusable real-time software modules. An added difficulty with reusable real-time software modules is meeting different timing requirements for different applications.
- Support for distributed programs and fault tolerance. The problem of predicting the timing behavior of real-time programs is further exacerbated in the context of distributed systems. Special fault-tolerance features might be added to the semantics of the language—for example, various recovery mechanisms subject to timing considerations.

Distributed real-time databases. In a real-time database system, a significant portion of data is often highly perishable in the sense that it has value to the mission only if used quickly. Satisfying the timing requirements involves two key issues: First, the degree of concurrency in transaction processing must be increased; second, concurrency control protocols and real-time scheduling algorithms must be integrated.

The characteristics of a real-time database, such as a tracking database, are distinct from commercial database systems. In a real-time database, transactions often must perform statistical operations—correlation, for example—over a large amount of data that is continuously updated, and they must satisfy stringent timing requirements. As observed by Bernstein et al.,[11] serializability is not a good criterion for concurrency control of such databases because of the limitation of concurrency allowed by serializable concurrent executions. Several approaches designed to obtain a high degree of concurrency by exploring application-specific semantic information have been proposed. In addition, a modular decomposition approach for nonserializable concurrency control and failure recovery has been proposed.[12] This approach assumes that global serialization is too strong a condition. Instead it uses a setwise serializability notion, which is shown to apply to the database associated with a cluster of tracking stations.

While concurrency control of transactions and hard real-time process scheduling have both progressed, very little is known about the integration between concurrency control protocols and hard real-time process-scheduling algorithms. In concurrency control we do not typically address the meeting of hard deadlines. The objective is to provide a high degree of concurrency and thus faster average response time. On the other hand, in the schedulability analysis of real-time process scheduling—for example, processes for signal processing and feedback control—it is customary to assume that tasks are independent, or that the time spent synchronizing their access to shared data is negligible compared with execution time and deadlines. The objective here is to maximize resources, such as CPU utilization, subject to meeting timing constraints. The fundamental challenge of real-time databases seems to be the creation of a unified theory that will provide us with a real-time concurrency-control protocol that maximizes both concurrency and resource utilization subject to three constraints at the same time: data consistency, transaction correctness, and transaction deadlines.

The dynamic nature of symbolic systems requires support for automated memory management.

Artificial intelligence. Real-time AI research currently emphasizes reasoning about time-constrained processes and using heuristic knowledge to control or schedule these processes. A key consideration in robust problem solving is to provide the best available solution within a dynamically determined time constraint. Many of the current issues in real-time-systems research are applicable to such a system, but additional considerations exist. For example, the dynamic nature of symbolic systems requires support for automated memory management. Current garbage collection techniques make it difficult, if not impossible, to guarantee a maximum system latency.

Other features of symbolic systems that exacerbate the predictability problem include the ability to create and execute a function at runtime and the ability to execute a runtime-selected function passed as an argument. In addition, opportunistic control strategies provide the ability to dynamically direct program execution in response to real-time events. This contrasts sharply with the current real-time-systems assumption of a statically determined "decision tree" control sequence. The implication here is that the sequences of processing cannot be predetermined.

This is by no means a comprehensive list of the issues involved in real-time symbolic systems. No doubt additional research issues will arise as real-time AI applications evolve and attempts are made to solve problems in severely time-constrained domains.

Fault tolerance. A real-time computer system and its environment form a synergistic pair. For example, most commercial and military aircraft cannot fly without digital-control computers. In such systems it is meaningless to consider on-board control computers without considering the aircraft itself. The tight interaction between the environment and a real-time computer arises from time and reliability constraints. Unless the computer can provide "acceptable" services to its environment, its role will be lost and thus viewed as failed or nonexistent. Failure can result from massive loss of components (static failure) or from failure to respond fast enough to environmental stimuli (dynamic failure).[13] This interplay must be carefully characterized for various real-time applications, in which even the determination of deadlines is by itself a relatively unexplored problem. On the basis of this characterization of a real-time problem, a

vast number of design and analysis problems for real-time computers remain to be solved—for example, optimal error handling, redundancy management, and tuning of architectures.

Certain research issues are important in making real-time systems fault tolerant and reliable. Specifically,

- The formal specification of the reliability requirement and the impact of timing constraints on such a requirement is a difficult problem.
- Error handling is usually composed of an ordered sequence of steps: error detection, fault location, system reconfiguration, and recovery. All these steps must be designed and analyzed in the context of combined performance (including timing constraints) and reliability. Interplay between these steps must be carefully studied. Hardware and operating-system support, together with their effects on performance and reliability, are important research subjects.
- The effects of real-time work loads on fault tolerance has not been adequately addressed. It is well known that the reliability of a computer system depends heavily on its work load. Characterizing the effects of "representative" real-time work loads on fault tolerance is essential.

Real-time-system architectures. Many real-time systems can be viewed as a three-stage pipeline: data acquisition from sensors, data processing, and output to actuators and/or displays.[14] Next-generation systems will often be distributed such that each node may be a multiprocessor (see Figure 2). A real-time system's architecture must be designed to support these components with high fidelity. For data acquisition and for actuators, the architecture must provide extensive I/O capabilities while providing fast, timely, and reliable operations for the data-processing stage.

Conventional real-time architectures are based on dedicated hardware and software: The architecture usually must change with a change in applications. Such architectures are neither cost effective nor well utilized. Because of advances in VLSI technology, it is becoming possible to develop a new distributed architecture suitable for broader classes of real-time applications. Important issues in this new architecture include interconnection topology, interprocess communications, and support of operating-system and fault-tolerance functions.

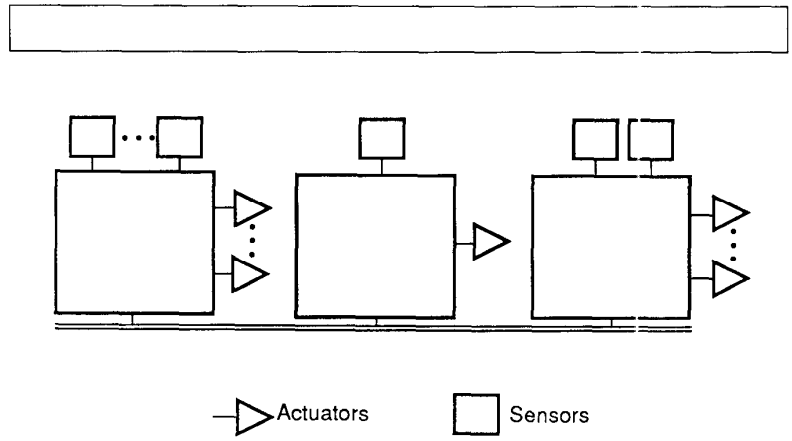

Figure 2. A distributed real-time architecture.

Some open research topics in real-time architecture are

- Interconnection topology for processors and I/O. The need for extensive I/O and high-speed data processing in real-time applications makes it important to develop an integrated interconnection topology for both processors and I/O. Although the processing topology has been studied extensively, little attention has been paid to the distribution of I/O data.
- Fast, reliable, and time-constrained communications.
- Architectural support for error handling.
- Architectural support for scheduling algorithms.
- Architectural support for real-time operating systems.
- Architectural support for real-time language features.

Ideally, any architectural design should adopt a synergistic approach wherein theory, operating system, and hardware are all developed with the single goal of achieving the real-time constraints in a cost-effective and integrated fashion.

Real-time communication. The communication media for next-generation distributed real-time systems will be required to form the backbone upon which predictable, stable, and extensible system solutions will be built. To be successful, the real-time communication subsystem must be able to predictably satisfy individual message-level timing requirements. The timing requirements are driven not only by applications' interprocess communication, but also by time-constrained operating-system functions invoked on behalf of application processes. Networking solutions for this context are distinguished from the standard nonreal-time solutions with the introduction of *time*. In a nonreal-time setting, it is sufficient to verify the logical correctness of a communications solution; however, in a real-time setting it is also necessary to verify timing correctness. Software engineering practices have helped in determining the logical correctness of systems solutions but have not addressed timing correctness. Timing correctness includes ensuring the schedulability of synchronous and sporadic messages as well as ensuring that the response time requirements of asynchronous messages are met. Ensuring timing correctness for static real-time communications systems using current technology is difficult; ensuring timing correctness in the next generation's dynamic environment will be a substantial research challenge.

Additional research is needed to develop technologies that support the unique challenges of real-time communications. These include

- Dynamic routing solutions with guaranteed timing correctness.
- Network buffer management that

supports scheduling solutions.
- Fault-tolerant and time-constrained communications.
- Network scheduling that can be combined with processor scheduling to provide system-level scheduling solutions.

Meeting these challenges will require substantial research effort with associated breakthroughs in network control theory. With the network forming the backbone of many next-generation distributed real-time systems, a system will be no stronger than the communications solution that supports it.

Conclusions

Many real-time systems of tomorrow will be large and complex and will function in distributed and dynamic environments. They will include expert system components and will involve complex timing constraints encompassing different granules of time. Moreover, economic, human, and ecological catastrophes will result if these timing constraints are not met. Meeting the challenges imposed by these characteristics very much depends on a focused and coordinated effort in all aspects of system development, such as

- Specification and verification techniques that can handle the needs of real-time systems with a large number of interacting components.
- Design methodologies that can be used to synthesize systems with the specified timing properties where these timing properties are considered from the beginning of the design process.
- Programming languages with explicit constructs to express—with unambiguous semantics—the time-related behavior of modules.
- Scheduling algorithms that can, in an integrated and dynamic fashion, handle (1) complex task structures with resource and precedence constraints, (2) resources (such as the communication subnet and I/O devices), and (3) timing constraints of varying granularity.
- Operating-system functions designed to deal with highly integrated and cooperative time-constrained resource management in a fast and predictable manner.
- Communication architectures and protocols for dealing efficiently with messages that require timely delivery.
- Architecture support for fault tolerance, efficient operating-system functioning, and time-constrained communication.

Real-time systems have brought about unmet challenges in a wide range of computer science disciplines. Each of these challenges must be overcome before a science of large-scale real-time systems can become a reality. The task is formidable, and success will require a concerted effort by many different participants in the computer science community. It will require the enticement of new researchers into the field, especially in academia, where relatively little work of this nature is being done. We must coordinate interaction between research efforts in universities and development efforts in industry so that academic researchers will be familiar with key problems faced by system developers, and system developers will be aware of relevant new theories and technologies.

The solution to developing a theory of large-scale real-time systems does not lie in the current methodologies of operations research, database theory, scheduling theory, or operating-systems theory. Rather, it lies in well-coordinated and expanded efforts of universities, industry, and government laboratories directed toward the distinct problems that this topic introduces. The need for such cooperation was also emphasized in a report to the Executive Office of the President, Office of Science and Technology Policy.[15] While that report dealt with problems in high-performance computing, software technology and algorithms, networking, basic research, and human resources, our general conclusions are the same. However, this article is much more detailed in its emphasis on the basic research needs of a single important research topic: real-time computing. □

Acknowledgments

This article incorporates ideas presented at the Carnegie Mellon University Workshop on Fundamental Issues in Distributed Real-Time Systems and represents the work of many individuals. Those who contributed significantly to the ideas presented here include Lui Sha, John Lehoczky, Hide Tokuda, Jay Strosnider, and Ragunathan Rajkumar, Carnegie Mellon University; Al Mok, University of Texas at Austin; Andre van Tilborg, Office of Naval Research; Krithi Ramamritham and Zhao Wei, University of Massachusetts; Kang Shin, University of Michigan; David C.L. Liu, University of Illinois; Pat Watson, IBM; and Karen Johnson and Ellen Waldrum, Texas Instruments.

A longer and more technical version of this article,[1] produced in concert with the individuals listed above, is available from J. Stankovic. This work has been partially supported by the Office of Naval Research under contract 048-716/3-22-85.

References

1. J. Stankovic, "Real-Time Computing Systems: The Next Generation," Tech. Report TR-88-06, COINS Dept., Univ. of Massachusetts, Jan. 1988.
2. W. Zhao, K. Ramamritham, and J. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," *IEEE Trans. Software Eng.*, Vol. SE-13, No. 5, May 1987, pp. 564-577.
3. F. Jahanian and A.K. Mok, "Safety Analysis of Timing Properties in Real-Time Systems," *IEEE Trans. Software Eng.*, Vol. SE-12, No. 9, Sept. 1986, pp. 890-904.
4. G.M. Reed and A.W. Roscoe, "A Timed Model for Communicating Sequential Processes," *Proc. ICALP 86*, Springer LNCS 226, 1986, pp. 314-323.
5. S.K. Dhall and C.L. Liu, "On a Real-Time Scheduling Problem," *Operations Research*, Vol. 26, No. 1, Feb. 1978, pp. 127-140.
6. M.R. Garey and D.S. Johnson, "Two-Processor Scheduling with Start-Times and Deadlines," *SIAM J. Computing*, Vol. 6, 1977, pp. 416-426.
7. C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, Vol. 20, No. 1, Jan. 1973, pp. 46-61.
8. K. Schwan et al., "High Performance Operating System Primitives for Robotics and Real-Time Control Systems," *ACM Trans. Computer Systems*, Vol. 5, No. 3, Aug. 1987, pp. 189-231.
9. J. Stankovic and K. Ramamritham, "The Design of the Spring Kernel," *Proc. Real-Time Systems Symp.*, CS Press, Los Alamitos, Calif., Dec. 1987, pp. 146-155.
10. H. Tokuda, J. Wendorf, and H. Wang, "Implementation of a Time Driven Scheduler for Real-Time Operating Systems," *Proc. Real-Time Systems Symp.*, CS Press, Los Alamitos, Calif., Dec. 1987, pp. 271-280.
11. P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, Mass., 1987.
12. L. Sha, J. Lehoczky, and E.D. Jensen, "Modular Concurrency Control and Failure Recovery," *IEEE Trans. Computers*, Vol. 37, No. 2, Feb. 1988, pp. 146-159.

13. K.G. Shin, C.M. Krishna, and Y.-H. Lee, "A Unified Method for Evaluating Real-Time Computer Controllers and Its Application," *IEEE Trans. Automatic Control*, Vol. AC-30, No. 4, Apr. 1985, pp. 357-366.

14. C.M. Krishna, K.G. Shin, and I.S. Bhandari, "Processor Trade-offs in Distributed Real-Time Systems," *IEEE Trans. Computers*, Vol. C-36, No. 9, Sept. 1987, pp. 1,030-1,040.

15. *A Research and Development Strategy for High Performance Computing*, Report, Executive Office of the President, Office of Science and Technology Policy, Nov. 20, 1987.

John A. Stankovic is an associate professor in the Computer and Information Science Department at the University of Massachusetts at Amherst. His research interests include investigating various approaches to scheduling on local area networks and multiprocessors, and developing flexible, distributed, hard real-time systems. He is currently building a hard real-time kernel called Spring, based on a new scheduling paradigm and on ensuring predictability. He is also doing research on a distributed database testbed called CARAT, which has been operational for several years.

Stankovic has held visiting positions in the Computer Science Department at Carnegie Mellon University and at INRIA in France. He is an editor for *IEEE Transactions on Computers* and a member of ACM and Sigma Xi. Stankovic received a BS in electrical engineering and MS and PhD degrees in computer science, all from Brown University, in 1970, 1976, and 1979, respectively.

Readers may contact the author at COINS Dept., Lederle Graduate Research Center, Univ. of Massachusetts at Amherst, Amherst, MA 01003.