REAL-TIME COMPUTING SYSTEMS: THE NEXT GENERATION

John A. Stankovic
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

COINS Technical Report 88-06
January 6, 1988

Real-Time Computing Systems: The Next Generation

Edited By
John A. Stankovic

COINS TECHNICAL REPORT 88-06
Department of Computer and Information Science
University of Massachusetts, Amherst, Ma.

1 The Problem Description

Real-time computing is a wide open research area of intellectually challenging computer science problems with direct payoff to current technology. In the current state of the art, there are few results to enable designers to handle the timing constraints of real-time systems in an effective manner, and there is not enough emphasis being placed on building the proper scientific underpinnings to achieve these needed results. Hence, this paper has six major objectives. They are:

- to convincingly argue that real-time computing is a separate, scientific discipline that is not being comprehensively covered by any other discipline such as operations research or computer science,
- to discuss the fundamental scientific issues of real-time computing,
- to briefly summarize the current state of the art in real-time computing,
- to state and then dispel the most common misconceptions of real-time computing,
- to encourage increased research in real-time systems, and
- to propose a research agenda for avoiding a crisis in the next generation real-time computing systems.

Real-time computing is that type of computing where 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. Examples of current real-time computing systems are command and control systems, nuclear power plants, process control plants, flight control systems, space shuttle and aircraft avionics, and robotics. Current real-time systems are expensive to build and their timing constraints are verified with ad hoc techniques, or with expensive and extensive simulations. Minor changes in the system result in another extensive round of testing. Different components of such systems are extremely difficult to integrate with each other, and consequently add to the cost of such systems. Millions (even billions) of dollars are currently being spent (wasted) by industry and government to build today's real-time systems. The current brute force techniques will not scale to meet the requirements of guaranteeing real-time constraints of the next generation systems.

The next generation real-time systems will be in similar application areas as current systems, but will be more complex in that they will be distributed, contain highly dynamic and adaptive behavior, exhibit intelligent behavior, have long lifetimes, and be characterized as having catastrophic consequences if the logical or timing constraints of the system are not met. Examples of these more sophisticated systems are the autonomous land rover, controllers of robots with elastic joints, systems found in intelligent manufacturing, the space station, and undersea exploration. Two major forces are pushing real-time systems into the next generation. First, is the rapid advance in hardware and, the second, is the need for artificial intelligence in real-time systems. These forces are exacerbating the difficult scientific and engineering problems faced in building real-time systems. These two driving forces are adding complex entities that have to be integrated into current and future applications, but the required design, analysis, and verification techniques for accomplishing this 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. New approaches are needed for real-time scheduling in such systems including on-line guarantees, and incremental algorithms that produce better results as a function of available time.

Section 2 discusses the unique challenge of distributed real-time systems, and provides a brief glimpse of the current state of the art. Section 3 states and attempts to dispel several of the common misconceptions concerning real-time systems. Section 4 presents a research agenda for achieving the goal of a more scientific basis for real-time systems. Finally, section 5 presents the overall conclusions of the report.

2 The Challenge Of Real-Time Computing Systems

2.1 Overview

Currently, a principle idea in the management of large scale systems is to hierarchically decompose the system into modules which 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, there have been no provisions to support the reasoning about time and reliability abstractions, two vital aspects of real-time systems. To develop a scientific underpinning for real-time systems, we are faced with the difficult scientific challenge of creating a set of unified theories and technologies which 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 on correctness, timeliness and reliability for the integrated system.

The goal of building a science of large scale real-time systems will require new research efforts in many distinct and yet related areas such as those discussed in Section 2.2. While each of these areas contains well developed theories and technologies, none currently contains theories and methods which address the central issues in real-time systems: a coher-

ent treatment of correctness, timeliness and fault tolerance in large scale distributed computations.

2.2 Research Issues and the State of the Art

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 satisfied such a specification. 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. It is also not clear how to extend programming notations to allow the programmer to specify computations that are constrained by real-time.

Some researchers advocate formulating real-time specifications by augmenting the system state with an additional *clock* variable [21]. This allows extant specification and verification methods to be used but raises the question of which-if any-actions should cause the *clock* to be incremented. The question is subtle because actions can be executed in parallel. Other researchers (e.g., [14,22]) give special treatment to the time metric, requiring new schemes for verifying properties involving time. The method of [14] using Real Time Logic allows true parallelism to be handled; the method of [22] adds time to traces, permitting a denotational model for a timed version of CSP. By investigating alternative semantic models and using them to model real systems we should learn what is the right balance between expressiveness and abstraction and what is easiest to use.

In general, the inclusion of a time metric may create subtleties in the semantics of concurrency models (e.g., see [22]) and complicates the verification problem. Whereas the proof of non-interference is the major verification task in extending sequential systems to concurrent systems based

on interleaving, the verification of real-time systems will require the satisfaction of timing constraints where those constraints are derived from the environment and implementation. Consequently, a major challenge is to solve the dilemma that verification techniques abstract away from the implementation, but it is the implementation and environment which provide the true timing constraints. The result is that we need a quantitative (e.g., deadlines, repetition rates) rather than the qualitative analysis (e.g., 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 of concern.

Another problem that will be encountered is dealing with the state explosion problem found in verification techniques. Techniques for abstracting many states into a higher level state are necessary to tackle this problem. This is a difficult problem even when timing constraints are not included.

Real-Time Scheduling Theory

While specification and verification concerns 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 the real-time system scheduling problems are different from the scheduling problems usually considered in areas of operations research. 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 which minimize the response time for a given task set. In many real-time computing systems, there is generally no incentive to minimize the response time other than meeting the 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. 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 it's associated processor utilization level below which the deadlines of all the tasks can be met. There are, however, other measures such as penalty functions which is defined according to the number of jobs that miss their deadlines, or a weighted success ratio which is the percentage of tasks that meet their deadlines weighted by the importance of those tasks.

Relatively little is known about efficient scheduling algorithms for real-time systems. The early results are limited to cases such as: a single processor with a precedence relation over a set of sporadic jobs [2], two processors with a precedence relation over a set of identical sporadic jobs [10], and a set of independent periodic tasks on a unique processor [18] and on multi-processors [5]. This work is applicable only in the simplest of situations. Again, most scheduling problems of interest to practical real-time systems are NP-hard, hence heuristics are required. Recently, there has been some work addressing more complicated yet practical scheduling

constraints. For example, the scheduling of real-time tasks with resource constraints [35,36], distributed real-time scheduling [28], and the problem of stochastic execution time and transient overloads[19]. However, a great deal more needs to be done so that we can handle even more general and realistic situations. For example, the problem of *jointly* scheduling system resources such as CPUs, I/O devices, communication media and secondary storage in order to meet the timing requirements of real-time tasks running on a distributed computer system is a realistic scheduling problem not being addressed to any great extent. We will say more about this problem in the next section on real-time operating systems.

The following is a partial list of some other considerations:

- static vs dynamic resource allocations,
- CPU scheduling vs data I/O management,
- centralized vs distributed systems,
- homogeneous vs heterogeneous systems, and
- on-line vs off-line scheduling.

We also need to consider various task characteristics such as:

- hard vs soft deadlines,
- preemptable vs non-preemptable tasks,
- sporadic vs periodic tasks, and
- independent vs precedence-constrained/communicating tasks.

Although the study of scheduling theory is a challenging and important scientific research area in its own right, we hasten to point out that such study can have significant impact on the design and implementation of practical real-time algorithms. Theoretical results can provide insight for the solution of practical problems, or provide performance estimates. On the other hand, many theoretical results rely on too many assumptions which do not hold in reality. Extensions of the theoretical results to more sophisticated situations would be invaluable.

Real-Time Operating Systems

Real-time operating systems play a key role in most real-time systems. Typically, one requires an operating system to manage the system resources well, so that a user can focus on the application specific problems rather than the underlying system issues. However, in real-time systems [24,29] the operating system and the application are more tightly intertwined than in timesharing systems. This presents the same dilemma mentioned above; how do we provide high level abstractions for real-time programmers, yet meet timing considerations which are fundamentally dependent on the implementation and environment. In addition, almost all resource management techniques used in existing operating systems are not designed to guarantee that a programs' critical timing constraints can be met. Some important operating system research issues include:

- Time-driven resource management. Traditionally, when many tasks are waiting for access to a shared resource, the allocation policy is to provide access in FIFO order. However, this policy totally ignores task's timing constraints. Time-driven allocation policies must be developed that can meet the real-time scheduling requirements[32]. Such management policies should be applied not only for the processor, but also for memory, I/O and communications resources. In fact, the entire operating system paradigm of treating tasks and their resource requests as random is suspect. New paradigms are needed [29].

- Problem-specific OS facilities. A real-time operating system's functions should be able to adapt to a variety of user and system needs. For instance, a real-time operating system should provide a separation between scheduling "policy" and "mechanism". Thus, a user can choose or the system can select the time-driven resource management algorithm most suitable for a particular application or situation. In distributed applications, a transaction mechanism seems to have many advantages. Consequently, the real-time operating system should provide support for transactions with real-time constraints[30].

Research is needed to identify the set of efficient OS primitives required to support the integration of locking protocols, commit protocols, and recovery protocols - all specifically addressing real-time constraints. In particular, recovery by "undoing" operations may not be applicable in many circumstances, rather some form of forward error recovery might be necessary.

- Integrated system-wide scheduling support. Real-time scheduling principles must be applied to system resources, application tasks, and operating system's overall design. In particular, a focussed effort is necessary to integrate real-time communication with real-time cpu scheduling, and with real-time database support for a large, complex real-time computer system. For a sequence of actions to meet a deadline, precedence constraints must be satisfied and resources must be available *in time* for each action of the sequence. Inopportune delays at any stage of the process can cause missed deadlines.

Real-Time Programming Languages and Design Methodology

As the complexity of real-time systems increases, high demand will be placed upon the programming abstractions provided by languages. Currently available abstractions and languages have to evolve with the 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 unpredicatable duration. Consequently, when processes are programmed in the form of Ada tasks, the resulting timing behavior is likely to be unpredictable. The building of the next generation of real-time systems will require a programming methodology that gives due considerations 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 utilizations 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 by measuring the resource utilization of the given set of tasks. 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. The investigation of reusable software modules is an important subject, because it reduces the software development cost and enhances the quality of resulting software. Reusable real-time software modules have the added difficulty of 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, e.g., various recovery mechanisms subject to timing considerations.

Distributed Real-Time Databases

In a real-time database system, often a significant portion of data is highly perishable in the sense that it has value to the mission only if

used quickly. To satisfy the timing requirements, there are two key issues. First, the degree of concurrency in transaction processing must be increased, and second, the integration of concurrency control protocols and real-time scheduling algorithms must occur. We first review the concurrency control issue. The classical theory of concurrency control is known as the serializability theory which accepts concurrent executions (schedules) that are equivalent to some serial schedule. There are three attractive properties of serializable schedules, namely, data consistency, transaction correctness and modularity of concurrency control protocols. Under the assumption that transactions are individually consistent and correct, a serializable concurrent execution of transactions leads to correct results and leaves the database consistent. In addition, serializable schedules can be enforced by *modular* concurrency control protocols, i.e., protocols can be used to schedule a transaction without reference to the semantics of other transactions. For example, the two phase lock protocol [7] is a modular concurrency control protocol. The modularity of concurrency control protocols is important for large scale transaction facilities where transactions are frequently modified. Consistency, correctness and modularity are attractive properties which account for the popularity of serializable concurrency control in commercial database systems.

The characteristics of a distributed real-time database, such as a tracking database, are distinct from commercial database systems. In a real-time database, transactions often have to perform statistical operations, e.g., correlation, over a large amount of data that is continuously updated and must satisfy stringent timing requirements. As observed by Bernstein, Hadzilacos and Goodman [1] serializability is not a good criterion for the concurrency control of such databases because of the limitation of concurrency allowed by serializable concurrent executions. There have been several proposed approaches which are designed to obtain a high degree of concurrency by exploring application specific semantic information [9,20,23]. In addition, a modular decomposition approach for non-serializable concurrency control and failure recovery was proposed [25], which assumes that while global serialization is too strong a condition, some form of distributed but localized serializability is needed. This is a model of the fact that within a cluster of tracking stations, consistency of data should be maintained deterministically.

We hasten to point out that while there is progress in both concurrency control of transactions and hard real-time process scheduling, 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 subject of meeting 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, e.g., processes for signal processing and feedback control, it is customary to assume that tasks are independent, or the amount of time spent in synchronizing their access to shared data is negligible compared with execution time and deadlines. The objective here is to maximize the resource, e.g., CPU utilization subject to meeting timing constraints. The fundamental challenge of real-time databases seems to be the creation of a unified theory which provides 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. Conceptually, we can view concurrency control protocols as a means to generate all the legal schedules. Given the universe of legal schedules, a real-time scheduling protocol is a means to select those schedules that maximize processor utilization subject to the constraints of deadlines. Needless to say, much work is needed in this area.

Artificial Intelligence

The current emphasis in real-time AI research concerns 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 issues currently being addressed in real-time systems research are applicable to such a system but there are additional considerations. 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 system's assumption of a statically determined "decision tree" control sequence. The implication of this is that the sequences of processing cannot be pre-determined. This is by no means a comprehensive list of the issues involved in real–time symbolic systems. We believe that 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, aircraft are not able to fly without digital control computers. In such systems it is meaningless to consider onboard control computers without considering the aircraft itself. The tight interaction between the environment and a real-time computer arises from the time and reliability constraints. Unless the computer can provide "acceptable" services to its environment, the computer's role will be lost and, thus, viewed as failed or non-existent. Failure can occur because of massive loss of components (static failure), or because of not responding fast enough to environmental stimuli (dynamic failure) [26]. 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 [27]. Based on this characterization of a real–time problem, a vast number of design and analysis problems for real-time computers remain to be solved, e.g., optimal error handling, redundancy management, and tuning of architectures.

Important research issues in making real-time systems fault-tolerant and reliable include:

- The formal specification of the reliability requirement and the impact

of timing constraints on such a requirement is a difficult problem. For example, NASA has set the probability of failure of each flight control computer to be less than 10^{-9} for a 10-hour operation period [13,33]. This requirement of ultra-reliability makes a clear distinction between flight control computers and others. The need and difficulty of developing and analyzing specifications make an innovative research on this subject imperative.

- 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 OS support, together with their effects on performance and reliability are important research subjects.

- The proper tradeoff between use of hardware and software for fault-tolerance must be determined. The hardware approach, e.g., FTMP [13], is fast but requires an excessive amount of hardware to implement triple (or quadruple) modular redundancy needed for the reliability of real-time applications. The software approach (e.g., SIFT [33]), on the other hand, is flexible, inexpensive, but slow. This tradeoff must be optimized such that the system can be made cost-effective, and that timing constraints will be met.

- The effects of real-time workloads on fault-tolerance has not been adequately addressed. It is well-known that the reliability of a computer system depends strongly on the workload it is handling (for example, see [34]). It is essential to characterize the effects of "representative" real-time workloads on fault-tolerance.

Real-Time System Architectures

Many real-time systems can be viewed as a three-stage pipe: data ac-

quisition, data processing, and output to actuators and/or displays [16]. A real-time systems architecture must be designed to support these three components with high fidelity. For the first and last components, the architecture must provide extensive I/O capabilities, while providing fast and reliable operations for the second component.

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

The topology used to interconnect the nodes in a distributed real-time system must possess the following four features.

(1) Homogeneity: Due to homogeneity, tasks can be allocated to any node based solely on deadlines and availability of resources. This is particularly useful when tasks have to be reallocated to different nodes because the nodes to which they were originally allocated failed before their completion.

(2) Scalability: This allows the computational power of the network to be changed at any time without redesigning any of the nodes and causing any problem of fan-in and fan-out.

(3) Survivability: Given a pair of nodes, the topology must have several shortest or optimal paths between the nodes. This enables easy routing of messages in the system. It also increases the survivability of the network in case of node/link failures.

(4) Experimental Flexibility: By disabling some of the links in the chosen topology, various architectures should be easily emulated. As a result, all algorithms that efficiently map onto these architectures can easily be investigated on the selected topology.

The communication delay between tasks residing at different nodes is a major problem in real-time systems. It is particularly important to deliver

messages to meet time constraints in executing real-time tasks. The concept of *virtual cut-through* was proposed [15] to solve this problem: messages will be delivered to destinations without being buffered at intermediate nodes unless the links and/or nodes in their routes are blocked or faulty. Advances in VLSI technology have made it possible to implement the virtual cut-through [4,6]. This VLSI solution appears to be very promising since only 10-15% of messages get buffered in the worst case, i.e., heaviest network traffic.

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

We conclude this section with some of the open research topics in real-time architecture.

- Interconnection Topology for Processors and I/O. Due to the need of extensive I/O and high speed data processing in real-time applications, it is 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 and Reliable Communications. The VLSI implementation of the virtual cut-through may provide fast communications. However, there is still uncertainty on how long it will take to deliver a particular message. It is difficult to guarantee that all hard deadlines will be met under this uncertainty. Moreover, any solution must consider delays that might occur due to failures. Further research is necessary to implement the virtual cut-through by considering both the predictability and reliability in delivering messages.

- Architectural Support for Error Handling. It is essential to have hardware support for speedy error detection, reconfiguration and recovery. This includes self-checking circuitry, maintenance processors, system monitors, voters, etc. Where to place these and when and how to operate them will be important to system performance and reliability.

- Architectural Support for Scheduling Algorithms. In order to support real-time scheduling algorithms, architectures may need to support, among other features, fast preemptability, sufficient priority resolution, efficient support for data-structures like priority queueing, and sophisticated I/O and communication media scheduling.

- Architectural Support for Real-Time Operating Systems. Operating systems for real-time systems would stand to benefit from facilities provided by underlying hardware in terms of support for real-time protocols, multiple contexts, real-time memory management including caching and garbage collection, interrupt handling, clock synchronization, etc.

- Architectural Support for Language Features: The use of special-purpose architectures designed to support real-time programming languages more explicitly and efficiently can provide immense benefits in the context of real-time systems. For instance, certain architecture features could aid in predicting program execution times, or hardware support for garbage collection can eliminate a bottleneck encountered in today's traditional architectures, or support for concurrency control can improve the performance of real-time languages.

Real–Time Communication

The communication media for next generation distributed real-time systems will be required to form the backbone upon which predictable, stable, 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' inter-process 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 the timing correctness. Timing correctness includes insuring the schedulability of synchronous and sporadic messages as well as insuring that the response time requirements of asynchronous messages are met. Insuring the timing correctness for static real-time communications systems using current technology is difficult. Insuring the timing correctness in the next generations dynamic environment will be a substantial research challenge.

Even though the communications channel is yet another resource, like the processor, there are at least three issues that differentiate the channel scheduling problem from the processor scheduling problem.

- Unlike the processor, which has a single point of access, access to the channel is attempted by a distributed set of nodes. Hence, a distributed protocol is needed.

- While preemptive algorithms are appropriate for scheduling tasks on a processor, preemption during message transmission will mean that the entire message needs to be transmitted.

- In addition to message deadlines arising from the semantics of an application, deadlines can also arise from buffer limitations. For example, when only one buffer is available, the message in the buffer must be transmitted before the next message arrives.

Like the uniprocessor environment where the Time Domain Multiplexing (TDM) techniques (cyclical executives) are being used to insure timing correctness, the current state of the art and predominant practice for insuring network timing correctness is to use TDM techniques. This manifests itself as time-lines on busses and cyclical schedulers on rings. TDM approaches provide a strong systems solution for tasks/message sets with a single period or harmonically related periods like those found in phone systems. Experience with cyclical executives have found that as the task set periods grow in number and become relatively prime, the TDM schedule tends to be *ad hoc* in nature, painful to generate, very inflexible and difficult

to modify [11]. Unfortunately, current local area network trends are marching down the same path with the TDM based approaches embodied by the High Speed Ring Bus (HSRB) [12] and the Fiber Distributed Data Interface (FDDI) [8] token passing rings. The FDDI is a commercially targeted high performance fiber optic token passing ring which should provide a strong solution for the general voice and video applications which have a small number of unique message periods. The HSRB is a proposed military standard for the next generation military systems which forms a subset of the real-time systems that this paper addresses. Its applications environment will be far more varied and dynamic often with large numbers of periods that could be relatively prime. To avoid repeating the problems of cyclical executives in the real-time communications environment, alternative approaches must be developed. To further exacerbate the problem, the next generation distributed real-time systems will become increasingly reconfigurable, driven by reliability and availability requirements. This results in a virtual explosion in the number of unique time-lines to be designed, integrated and tested if TDM based solutions are used. For these highly dynamic, adaptive systems, TDM techniques will be inadequate.

Research is needed not only to address the future challenges of the real-time distributed communications environment but also to develop techniques for applying current scheduling technology in the distributed communications domain. IBM FSD [3] and researchers at Carnegie-Mellon University [17] initially introduced frame level contention resolution with message based (as opposed to station based) priority assignment to the distributed communications domain in IBM's Distributed Systems Data Bus (DSDB). Researchers at Carnegie-Mellon University [31] are currently extending this work by developing techniques for applying recent advances in scheduling theory to the emerging Local Area Network (LAN) standards. New work in window protocols for time constrained message transmission has also appeared [37].

Additional research is needed to develop technologies that support the unique challenges of real-time communications which 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, the strength of these systems will be no stronger than the communications solution that supports it.

3 Common Misconceptions

Real-time system design has not attracted the attention from academic computer scientists that it deserves. This lack of adequate attention has been attributed to some common misconceptions about real-time systems. Let us now discuss the major misconceptions.

There is no science in real-time system design

It is certainly true that the state-of-the-art in real-time system design is mostly ad hoc. That 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, e.g., the first flight of the Space Shuttle was delayed, at considerable cost, because of a subtle timing bug which arose from a transient CPU overload during system initialization. Can we then develop a scientific basis for verifying a design to be 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 some of the scientific basis that is emerging. We are starting to understand what the important problems are in real-time scheduling of resources. The subtleties of including a time metric in system specification methods and semantic

theories for real-time programming languages are beginning to be investigated. A good analogy can be drawn between current research in real-time system design and the status of VLSI design research five years ago. There are some striking similarities and the potential scientific rewards are equally great.

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

Advance in supercomputer design will likely exploit parallel processors to improve system throughput, but this does not mean that timing constraints will automatically be met. 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, the real-time task and communication scheduling problems are likely to be harder 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. There are also other important issues in real-time systems design that cannot be resolved by supercomputer hardware alone, as discussed in the following points.

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. The objective of real-time computing is to

meet the individual timing requirement of each of the tasks. Rather than being fast (which is a relative term anyway), the most important property that a real-time system should have is predictability, i.e., its functional and timing behavior should be as deterministic as is necessary to satisfy system specifications. Fast computing is helpful in meeting stringent timing specifications, but fast computing alone does not guarantee predictability. There are other factors other than fast hardware or algorithms that determine predictability. Sometimes, the implementation language may not be expressive enough to prescribe certain timing behavior. For example, the delay statement of Ada only puts 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 a only 1 in 67 probability that a transient overload during initialization could put the redundant processors out of sync, and it did. Predictability, not speed is the foremost goal in real-time system design.

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

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 as-

sumptions (unwisely, but as a last resort) on which the correctness of an implementation depends. The 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 writing device drivers 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 component 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 point below). The proper design of a real-time system, however, requires solutions to many other interesting problems, e.g., specification and verification of timing behavior, and programming language semantics dealing with time. There are also theoretical problems that 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 a timing constraint! Indeed, the correct functioning of many systems often depends on the implementation being able to perform an operation that requires the satisfaction of certain timing constraints, albeit implicitly specified (e.g., 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, e.g., 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 that is 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 mostly concerned with analyzing the average values of performance parameters, whereas an important consideration in real-time system design is whether some stringent deadlines can be met or not. Queueing models traditionally make use of 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 non-linear performance degradation due to slight deviations from uniform traffic in multi-stage interconnection networks which has been reported by Pfister et al) is likely to be catastrophic for time-critical communication packets. Likewise, the combinatorial scheduling problems in operations research deal mostly with one-shot tasks, i.e., 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 that are not to be 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 that one believes in quantum mechanics). The relevant question is of course 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 non-zero probability does not give the designer license to INCREASE the odds of failure by not trying to allocate resources carefully so as to meet the critical timing constraints. We certainly cannot guarantee anything outside our control, but for those that we can, 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 schedulers to make resource allocation decisions for different time granularities. A particularly vexing industrial problem is how to reconfigure systems to accomodate 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 for 15 or more years, and hence any design methodology

for such systems must not assume a static environment.

4 Research Agenda

The primary audience for this paper is the active researcher. We hope that the researchers reading this paper have been enticed by the multitude of important problems to be solved. However, we now shift our emphasis to address government and industrial managers who have responsibility for long term research directions. In this section we sketch a research agenda for solving these problems.

All the areas mentioned in this paper are important and increased funding is needed in each of them. However, given limited funds we suggest focussing on two main themes which have the greatest likelihood of having the biggest impact. The themes are:

- the formalization of the time dimension including the specification, verification, and analysis of time constraints, and
- time driven resource allocation which encompasses many of the areas addressed in this paper to various extents.

Fundamental scientific results in these two areas will accelerate progress in the other areas such as architecture, data bases, etc. However, the increased funding and research must be directed to the problems addressed in this report, and not wasted on the misconceived notions of real-time systems. Therefore, we see a more focussed agenda for the first four to five years where the primary activities are in the two areas identified above, and secondary activities are supported in all the other areas mentioned in this report. After this first phase, there needs to be a gradual broadening of support to encompass all the research topics.

A necessary ingredient for achieving the needed results is to expand the research community. A concerted effort must be made to raise people's awareness of the need for funding and research in this area. The number of University groups doing real-time research must increase dramatically (from the current 5-10 range to the 40-50 range). In addition, a Center for real-time systems would be useful and could be responsible for running workshops, increasing community awareness of the problems and needs,

providing a catalyst for merging results from different projects, providing common workloads and other benchmarks, and providing application inputs and real-world constraints to fundamental research. Several other smaller centers could be more technically oriented and be directed to subsets of the overall problem. Initially, one major center, two smaller centers each focussed on one of the two themes outlined above, and the 40-50 University projects would provide a good start to the needed work. Then in 4-5 years, the program needs to further expand as the research agenda broadens to include all of the other topics.

5 Conclusions

In summary, we believe that many real-time systems of tomorrow will be large and complex, will function in distributed and dynamic environments, will include expert system components, will involve complex timing constraints encompassing different granules of time, and will result in economic, human, and ecological catastrophies if these timing constraints are not met. Meeting the challenges imposed by these characteristics very much depends on a focussed 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 the time realted behavior of modules and with unambiguous semantics,
- Scheduling algorithms that can, in an integrated and dynamic fashion, handle complex task structures with resource and precedence constraints, handle resources (such as the communication subnet and I/O devices), and handle 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 efficiently dealing with messages that require timely delivery, and
- Architecture support for fault tolerance, for efficient operating system functionsing, and for time constrained communication.

This paper shows that real-time systems have brought about challenges in a wide range of computer science disciplines that are not being met. Each of these challenges must be overcome before a science of large scale real-time systems can become a reality. The task is very formidable and success will require a concerted effort on the part of many different participants in the computer science community and it will require expanded efforts and the enticement of new researchers into this field, especially in academia where relatively little such work is going on. We must coordinate the interactions among the research efforts in universities and the development efforts in industries so that academic researchers will be familiar with the key problems faced by system developers, and system developers will be aware of the relevant new theories and technologies. The solution to development of 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 among universities, industry and government laboratories directed toward the distinct problems that this topic brings to the fore. This latter point is similar to one of the major conclusions of [38] where extensive cooperation among all these agencies is recommended to sovle problems in High Performance Computing, Software Technology and Algorithms, Networking, and Basic Research and Human Resources. The conclusions of this paper are in concert with the general conclusions of that report, but this paper is much more detailed specifically expounding upon the basic research needs of one important research topic: real-time computing.

6 Acknowledgement

In the preparation of this document, I have taken the liberty of using the ideas presented in the CMU Workshop on Fundamental Issues in Distributed Real-Time Systems, March 1987. I want to thank all the workshop participants for their contribution. In particular, a number of people actually wrote portions of this article. I would especially like to thank Lui Sha (CMU) and John Lehoczky (CMU) for taking major responsibility to develop and write parts of Section 2. I also want to thank Al Mok (Texas) for his contributions on the subject of specification and verification and for taking major responsibility for the section on Common Misconceptions. I also thank David C. L. Liu (Illinois) for his contribution on real-time scheduling, Ellen Waldrum (Texas Instruments) and Karen Johnson (Texas Instruments) for contributions on real-time AI, Kang Shin (Michigan) for contributions on fault tolerance and architecture, Hide Tokuda (CMU) for his contribution on operating systems, Ragunathan Rajkumar (CMU) and Hide Tokuda on the subject of real-time programming, Zhao Wei (Massachusetts) and Jay Strosnider (CMU) for the section on real-time communication, and Krithi Ramamritham (Massachusetts) and Pat Watson (IBM) for general comments and help with the overview and summary. I would also like to thank Andre van Tilborg (Office of Naval Research) for his identification of the need for a report of this type and his overall support, comments and ideas that appear in various places throughout the report. A number of other people read and commented on various versions of this report. A special thanks to all of them.

References

[1] Bernstein, P. A., V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publication Co., 1987.

[2] Blazewicz, J., Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines, *Proceedings of the International Workshop on Modelling and Performance Evaluation of Computer Systems*, pp.57-65, 1976.

[3] Dailey, G.E., Distributed Systems Data Bus, External Audit - PDR, 03J/04273/03, IBM FSD, Manassas, Virginia, May, 1983.

[4] Dally, W. J and C. L. Seitz, The Torus Routing Chip, *Distributed Computing*, Vol. 1, No. 3, 1986.

[5] Dhall, S. K., and C. L. Liu, On a Real-Time Scheduling Problem, *Operations Research*, Vol. 26, No. 1, pp. 127-140, February 1978.

[6] Dolter, J. W., P. Ramanathan, and K. G. Shin, A VLSI Architecture for Dynamic Routing in HARTS, submitted for publication.

[7] Eswaran, K. P., J. N. Gray, R. A. Lorie and I. L. Traiger, The Notion of Consistency and Predicate Lock in a Database System, *CACM*, Vol. 19, No. 11 , November 1976.

[8] FDDI Token Ring - Media Access Control, draft proposed, American National Standards Institute, No. ANSI X3T9.5/83-16, March, 1985.

[9] Garcia-Molina, H., Using Semantic Knowledge For Transaction Processing In A Distributed Database, *ACM Transaction on Database Systems*, Vol. 8, No. 2, June, 1983.

[10] Garey, M. R. and D. S. Johnson, Two-processors Scheduling with Start-times and Deadlines, *SIAM Journal on Computing*, Vol. 6, pp. 416-426, 1977.

[11] Hood, P. and Grover, V., Designing Real Time Systems in Ada, SofTech, Inc. Waltham, MA, Tech. Report 1123-1, January, 1986.

[12] HSRB, SAE AE9-B High Speed Data Bus Standard, Society of Automotive Engineers, Subcommittee 9-B, Issue 1, Draft 2, January, 1986.

[13] Hopkins, A. L., et. al, FTMP – A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft, *Proc. IEEE*, Vol. 66, No. 10, pp. 1221 1239, Oct. 1978.

[14] Jahanian F., and A. K. Mok, Safety Analysis of Timing Properties in Real-Time Systems, *IEEE Transactions on Software Engineering*, SE-12(9), pp. 890-904, September 1986.

[15] Kermani, P. and L. Kleinrock, Virtual Cut-Through: A New Computer Communication Switching Technique, *Computer Networks*, Vol. 3, 1979, pp. 267-286.

[16] Krishna, C. M., K. G. Shin, and I. S. Bhandari, Processor Tradeoffs in Distributed Real-Time Systems *IEEE Trans. Comput.*, Vol. C-36, No. 9, pp. 1030–1040, Sep. 1987.

[17] Lehoczky, J.P. and Sha, L., Performance of Real-Time Bus Scheduling Algorithms, ACM Performance Evaluation Review, Special Issue, Vol. 14, No. 1, May, 1986.

[18] Liu, C. L., and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real–Time Environment, *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, pp. 46-61, January 1973.

[19] Locke, C. D., Best-Effort Decision Making for Real–Time Scheduling, PhD Thesis, CMU, May 1985.

[20] Lynch, N. A., Multi-level Atomicity - A New Correctness Criterion for Database Concurrency Control, *ACM Transactions on Database Systems*, Vol. 8, No. 4, December, 1983.

[21] Ostoff, J. S., and W. M. Wonham, "Modelling, Specifying, and Verifying Real–Time Embedded Computer Systems," *Proc. 8th RTSS*, Dec. 1987.

[22] Reed, G. M., and A. W. Roscoe, A Timed Model for Communicating Sequential Processes, *Proceedings of ICALP'86*, Springer LNCS 226, pp. 314-323, 1986.

[23] Schwarz, Peter and Alfred Spector, Synchronizing Shared Abstract Types *ACM Transaction on Computer Systems*, August 1984.

[24] Schwan, K., et. al., High Performance Operating System Primitives for Robotics and Real-Time Control Systems, *ACM Transactions on Computer Systems*, Vol. 5, No. 3, August 1987.

[25] Sha, L., J. Lehoczky, and E. D. Jensen, Modular Concurrency Control and Failure Recovery, to appear in *IEEE Transaction on Computers*, 1988.

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

[27] Shin, K. G., Introduction to the Special Issue on Real-Time Systems, *IEEE Trans. Comput.*, Vol. C-36, No. 8, pp. 901-902, Aug. 1987.

[28] Stankovic, J., K. Ramamritham, and S. Cheng, Evaluation of a Bidding Algorithm for Hard Real-Time Distributed Systems, *IEEE Transactions on Computers*, Vol. C-34, No. 12, pp. 1130-1143, December 1985.

[29] Stankovic, J. and K. Ramamritham, The Design of the Spring Kernel, *Proc. of the Real Time Systems Symposium*, Dec. 1987.

[30] Stankovic, J. and W. Zhao, On Real–Time Transactions, submitted to *ACM SIGMOD Record*, Special Issue on Real–Time Database Systems, Nov. 1987.

[31] Strosnider, J.K., Lehoczky, J.P., and Sha, L., A Novel Scheduling Approach for Token Passing Media, submitted for publication.

[32] Tokuda, H., J. Wendorf, and H. Wang, Implementation of a Time Driven Scheduler for Real–Time Operating Systems, *Proc. Real–Time Systems Symposium*, Dec. 1987.

[33] Wenseley, J. H., et. al, SIFT – Design and Analysis of a Fault-Tolerant Computer for Aircraft Control, *Proc. IEEE*, Vol. 66, No. 10, pp. 1240–1255, Oct. 1978.

[34] Woodbury M. H., and K. G. Shin, Workload Effects on Fault Latency for Real-Time Computing Systems, *Proc. 1987 Real-Time Systems Symp.*, Dec. 1987.

[35] Zhao, W., K. Ramamritham, and J. Stankovic, Scheduling Tasks with Resource Requirements in Hard Real-Time Systems, *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 5, pp. 564-577, May 1987.

[36] Zhao, W., K. Ramamritham, and J. Stankovic, Preemptive Scheduling Under Time and Resource Constraints, *IEEE Transactions on Computers*, Vol. C-36, No. 8, pp. 949-960, August 1987.

[37] Zhao, W., J. Stankovic, K. Ramamritham, A Window Protocol for Transmission of Time Constrained Messages, submitted for publication.

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