Real-Time and Embedded Systems

JOHN A. STANKOVIC

University of Massachusetts, Amherst ⟨*stankovic@cs.umass.edu*⟩

Real-time systems are those systems in which the correctness of the system depends not only on the logical results of computation but also on the time at which the results are produced [Stankovic 1988]. They span a broad spectrum of complexity from very simple microcontrollers in embedded systems (a microprocessor controlling an automobile engine) to highly sophisticated, complex, and distributed systems (air traffic control for the continental United States). Other real-time systems include command and control systems, process control systems, and multimedia and high-speed communication systems. Some real-time systems are beginning to add expert systems and other AI technology, creating additional requirements and complexities. At least three major trends in the real-time and embedded systems field have had a major impact on its technology: the increased growth and sophistication of embedded systems, the development of more scientific and technological results for hard real-time systems, and the advent of distributed multimedia.

Most *embedded systems* consist of a small microcontroller and limited software situated within some product such as a microwave oven or automobile. However, with the increasing sophistication of such systems, powerful microcontrollers and digital signal processing (DSP) chips are commonly used, as are off-the-shelf real-time operating systems and design and debugging tools. Many people working with embedded systems deal on a daily basis with sensors and data acquisition technology and systems; others construct architectures based on single-board computers (many are still 68000-based, but RISC processors are being used more and more) and buses such as the VME bus. Many people are involved with the programming and debugging of embedded systems, largely using the C programming language and cross development and debugging platforms. Embedded systems may or may not have real-time constraints.

Hard real-time systems are those in which missing an important deadline can cause severe consequences, even death. In this area, many fundamental results have been developed. For example, in real-time scheduling, rate monotonic analysis has enabled careful evaluation of many practical systems; the concept and analysis of competitive algorithms has provided important scheduling bounds and limits; and on-line planning has added flexible and dynamic capabilities to real-time systems. Operating systems research has produced predictable primitives, time-constrained synchronization techniques, and reservation and admission-control paradigms. Many other results exist in real-time architecture, fault tolerance, communication protocols, specification and design tools, formal verification, databases and object-oriented systems. Emphasis on all these areas is expected to increase in the foreseeable future. Many hard real-time systems are embedded systems.

Distributed multimedia have produced a new set of soft real-time requirements. Real-time principles lie at the heart of distributed multimedia, but without the concomitant high reliability

requirements found in safety-critical, hard real-time systems.

Underlying Principles. Typically, a real-time system consists of a controlling and a controlled system. For example, in an automated factory, the controlled system is the factory floor with its robots, assembling stations, and the assembled parts, while the controlling system is the computer and human interfaces that manage and coordinate the activities on the factory floor. Thus, the controlled system can be viewed as the environment with which the computer interacts.

The controlling system interacts with its environment using information about the environment available from various sensors. It is imperative that the state of the environment as perceived by the controlling system be consistent with the actual state of the environment. Hence, periodic monitoring of the environment as well as timely processing of the sensed information is necessary.

Timing correctness requirements in a real-time system arise because of the physical impact of the controlling system's activities upon its environment via actuators. The most common timing constraints for tasks are *periodic*, *aperiodic*, and *sporadic*. A periodic task is one that is activated every T units. The deadline for each activated instance may be less than, equal to, or greater than the period T. An aperiodic task is activated at unpredictable times. A sporadic task is an aperiodic task with the additional constraint that there is a minimum interarrival time between task activations.

What happens when timing constraints are not met? The answer depends, for the most part, on the type of application. A real-time system that controls a nuclear power plant or a missile cannot afford to miss timing constraints for critical tasks. Resources needed for critical tasks in such systems must be preallocated so that the tasks can execute without delay. In many situations, however, some leeway does exist. For example, even on an automated factory floor, if it is estimated that the correct command to a robot cannot be generated on time, it may be appropriate to command the robot to stop (provided no other moving objects will collide with it and cause a different type of disaster) or to slow down (thereby dynamically generating more time to produce a correct command).

We now discuss underlying principles in several representative areas of real-time computing, including real-time scheduling, real-time kernels, and distributed multimedia.

Real-time scheduling. Real-time scheduling is the process of creating start and finish times for sets of tasks such that all timing, precedence, and resource constraints are met. Real-time scheduling results in recent years have been extensive. Theoretical results have identified worst-case bounds for dynamic on-line algorithms, and complexity results have been produced for various types of assumed task-set characteristics.

More applied scheduling results have also been produced. For example, an extensive set of improvements has been made to the rate monotonic algorithm, which assigns the highest priority to the most frequent periodic task. Often a system has both periodic and sporadic tasks. To handle this situation, the sporadic server algorithm was invented. This algorithm guarantees that the deadlines of all periodic tasks will be met and provides excellent response time for the sporadic tasks. Techniques were also created to handle the problem of priority inversion, a situation where a low-priority task holds a resource required by a high-priority task. A set of algorithms that perform dynamic on-line planning were also produced. These algorithms have many nice properties, including quality of service guarantees, early warning that a deadline will be missed, and admission control. We have

also seen the development of scheduling results for imprecise computation—a situation where tasks obtain a greater value the longer they execute up to some maximum value.

Real-time kernels. Operating systems must provide basic support for predictably satisfying real-time constraints, for fault tolerance and distribution, and for 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. At least three issues need to be addressed:

—The time dimension must be elevated to a central principle of the system, not simply left as an afterthought.

—The basic paradigms found in today's general-purpose distributed operating systems must change. Currently, they are based on the notion that application tasks request resources as if they were random processes; operating systems are designed to expect random inputs and to display good average-case behavior. The new paradigm must be based on the delicate balance of flexibility and predictability: the system must remain flexible enough to allow a highly dynamic and adaptive environment, but at the same time be able to predict and possibly avoid resource conflicts so that timing constraints can be met.

—A highly integrated and time-constrained resource allocation approach is necessary to address timing constraints, predictability, adaptability, correctness, safety, and fault tolerance.

Real-time kernels are also being extended to operate in highly cooperative multiprocessor and distributed system environments. In such systems, sets of communicating tasks (possibly located across a network) must be scheduled to complete before a deadline, accounting for all system overheads, such as message copying and message transmission over the network. This is known as end-to-end scheduling.

The Mars project [Kopetz 1989], the Spring project [Stankovic 1991], and a project at the University of Michigan [Shin 1991] are all attempting to solve distributed end-to-end scheduling. The Mars project uses an a priori analysis and then statically schedules and reserves resources so that distributed execution can be guaranteed to make its deadline. The Spring approach supports dynamic requests for real-time virtual circuits (guaranteed delivery time) and real-time datagrams (best-effort delivery) integrated with CPU scheduling so as to guarantee application-level end-to-end timing requirements. Spring uses a distributed reflective memory based on a fiber optic ring to achieve the lower-level predictable communication properties. The Michigan work also supports dynamic real-time virtual circuits and datagrams, but is based on a general multi-hop communication subnet.

Regarding the portability of applications, many real-time UNIX operating systems are appearing [Furht 1991], and a standard for real-time operating systems, RT POSIX, is being developed [Gallmeister 1995]. While such standards facilitate porting the code, how to assess the timing properties of the ported application is still an open issue.

Distributed multimedia. Many real-time control applications, such as agile manufacturing and process control, operate in highly nondeterministic environments under timing constraints of many types. Significant improvements can be made by embedding continuous and multimedia support in these applications. For example, in agile manufacturing, remote factories, each consisting of many automated workcells, must coordinate to handle new strategies for incoming product orders, to design new products, to schedule just-in-time deliveries of manufactured components, to monitor the plant operations, and to solve difficult manufacturing floor problems collaboratively.

To implement solutions cost-effectively and give multimedia applications direct access to plant operational data, it is envisioned that the same computers would control the time-constrained operations in and across the workcells and support multimedia [Guha 1995]. The backbone network would likely be ATM. Distributed multimedia over ATM networks has enormous potential to provide these applications with teleconferencing for real-time coordination and collaborative design, as well as visual access via cameras to remote and local plant operations. These applications would also benefit from a distributed real-time database that contains sensor information and the values of control variables, both constrained by temporal validity intervals, plant operational data, information on availability of raw materials, inventory of products, customer orders, and so on. The confluence of real-time databases, multimedia, and real-time control has great potential for moving application areas such as manufacturing and process control into the next generation.

REFERENCES

FURHT, B., GROSTICK, D., GLUCH, D., RABBAT, G, PARKER, J., AND MCROBERTS, M. 1991. *Real-Time Unix Systems, Design and Application Guide*. Kluwer Academic, Boston, MA.

GALLMEISTER, B. 1995. *POSIX.4: Programming for the Real World*. O'Reilly and Associates, Sebastopol, CA.

GUHA, A., PAVAN, A., LIU, J., RASTOGI, A., AND STEEVES, T. 1995. Supporting real-time and multimedia applications on the Mercury testbed. *IEEE J. Select. Areas Commun. 13*, 4, 749–763.

KOPETZ, H., DAMM, A., KOZA, C., AND MULOZZANI, D. 1989. Distributed fault tolerant real-time systems: The Mars approach. *IEEE Micro 9*, 1, 25–40.

SHIN, K. 1991. HARTS: A distributed real-time architecture. *IEEE Computer 24*, 5, 25–35.

STANKOVIC, J. 1988. Misconceptions about real-time computing: A serious problem for next generation systems. *IEEE Computer 21*, 10, 10–19.

STANKOVIC, J. AND RAMAMRITHAM, K. 1991. The Spring kernel: A new paradigm for real-time systems. *IEEE Software 8*, 3, 62–72.