CHAPTER 2

Hard versus Soft Real-Time Systems

Now that we have seen several typical real-time applications, we are ready to discuss in depth the characteristics that distinguish them from nonreal-time applications. We begin by discussing exactly what "real time" means.

2.1 JOBS AND PROCESSORS

For the purpose of describing and characterizing different types of real-time systems and methods for scheduling and resource management, it is more convenient for us to speak of all kinds of work done by computing and communication systems in general terms. We call each unit of work that is scheduled and executed by the system a *job* and a set of related jobs which jointly provide some system function a *task*. Hence, the computation of a control law is a job. So is the computation of a FFT (Fast Fourier Transform) of sensor data, or the transmission of a data packet, or the retrieval of a file, and so on. We call them a control-law computation, a FFT computation, a packet transmission, and so on, only when we want to be specific about the kinds of work, that is, the types of jobs.

Similarly, rather than using different verbs (e.g., compute and transmit) for different types of jobs, we say that a job executes or is executed by the (operating) system. Every job executes on some resource. For example, the jobs mentioned above execute on a CPU, a network, and a disk, respectively. These resources are called servers in queuing theory literature and, sometimes, active resources in real-time systems literature. In later chapters, we will use the term server extensively to mean something else. To avoid overloading this term, we call all these resources *processors* except occasionally when we want to be specific about what they are.

2.2 RELEASE TIMES, DEADLINES, AND TIMING CONSTRAINTS

In the next chapter we will discuss in detail how jobs and processors are often characterized in order to schedule, manage, and reason about them. For now, we focus on the release times and deadlines of jobs, two parameters that distinguish jobs in real-time systems from those in nonreal-time systems.

The *release time* of a job is the instant of time at which the job becomes available for execution. The job can be scheduled and executed at any time at or after its release time whenever its data and control dependency conditions are met. As an example, we consider a system which monitors and controls several furnaces. After it is initialized and starts execution (say at time 0), the system samples and reads each temperature sensor every 100 msec and places the sampled readings in memory. It also computes the control law of each furnace every 100 msec in order to process the temperature readings and determine flow rates of fuel, air, and coolant. Suppose that the system begins the first control-law computation at time 20 msec. The fact that the control law is computed periodically can be stated in terms of release times of the control-law computation jobs $J_0, J_1, \ldots, J_k, \ldots$. The release time of the job J_k in this job stream is $20 + k \times 100$ msec, for $k = 0, 1, \ldots$. We say that *jobs have no release time* if all the jobs are released when the system begins execution.

The *deadline* of a job is the instant of time by which its execution is required to be completed. Suppose that in the previous example, each control-law computation job must complete by the release time of the subsequent job. Then, their deadlines are 120 msec, 220 msec, and so on, respectively. Alternatively, if the control-law computation jobs must complete sooner, their deadlines may be 70 msec, 170 msec, and so on. We say that a job has no deadline if its deadline is at infinity.

In this example, as in many others, it is more natural to state the timing requirement of a job in terms of its *response time*, that is, the length of time from the release time of the job to the instant when it completes. We call the maximum allowable response time of a job its *relative deadline*. Hence the relative deadline of every control-law computation job mentioned above is 100 or 50 msec. The deadline of a job, sometimes called its *absolute deadline*, is equal to its release time plus its relative deadline.

In general, we call a constraint imposed on the timing behavior of a job a *timing constraint*. In its simplest form, a timing constraint of a job can be specified in terms of its release time and relative or absolute deadlines, as illustrated by the above example. Some complex timing constraints cannot be specified conveniently in terms of release times and deadlines. We will discuss the parameters needed to specify those constraints when they arise, but in most of this book, we are concerned primarily with this simple form.

2.3 HARD AND SOFT TIMING CONSTRAINTS

It is common to divide timing constraints into two types: hard and soft. There are many definitions of hard and soft real-time constraints. Before stating the definition used in this book, let us first look at three frequently encountered definitions so you will be aware of them. They are based on the functional criticality of jobs, usefulness of late results, and deterministic or probabilistic nature of the constraints.

2.3.1 Common Definitions

According to a commonly used definition, a timing constraint or deadline is *hard* if the failure to meet it is considered to be a fatal fault. A hard deadline is imposed on a job because a late result produced by the job after the deadline may have disastrous consequences. (As examples, a late command to stop a train may cause a collision, and a bomb dropped too late may hit a civilian population instead of the intended military target.) In contrast, the late completion of a job that has a *soft deadline* is undesirable. However, a few misses of soft deadlines do no serious harm; only the system's overall performance becomes poorer and poorer when more and more jobs with soft deadlines complete late. This definition of hard and soft deadlines invariably leads to the question of whether the consequence of a missed deadline is indeed serious enough. The question of whether a timing constraint is hard or soft degenerates to that of how serious is serious.

In real-time systems literature, the distinction between hard and soft timing constraints is sometimes stated quantitatively in terms of the usefulness of results (and therefore the overall system performance) as functions of the tardinesses of jobs. The *tardiness* of a job measures how late it completes respective to its deadline. Its tardiness is zero if the job completes at or before its deadline; otherwise, if the job is late, its tardiness is equal to the difference between its *completion time* (i.e., the time instant at which it completes execution) and its deadline. The usefulness of a result produced by a soft real-time job (i.e, a job with a soft deadline) decreases gradually as the tardiness of the job increases, but the usefulness of a result produced by a hard real-time job (i.e., a job with a hard deadline) falls off abruptly and may even become negative when the tardiness of the job becomes larger than zero. The deadline of a job is softer if the usefulness of its result decreases at a slower rate. By this means, we can define a spectrum of hard/soft timing constraints. This quantitative measure of hardness and softness of deadlines is sometimes useful. It is certainly more appealing to computer scientists and engineers who have been trained not to rely on handwaving, qualitative measures. However, there is often no natural choice of usefulness functions. When choices are made, it is difficult to validate that the choices are sound and that different measures of the overall system performance as functions of tardinesses indeed behave as specified by the usefulness functions. Consequently, this kind of quantitative measure is not as rigorous as it appears to be.

Sometimes, we see this distinction made on the basis of whether the timing constraint is expressed in deterministic or probabilistic terms. If a job must never miss its deadline, then the deadline is hard. On the other hand, if its deadline can be missed occasionally with some acceptably low probability, then its timing constraint is soft. An example is that the system recovery job or a point-of-sales transaction completes within one minute 99.999 percent of the time. In other words, the probability of failure to meet the one-minute relative deadline is 10^{-5}. This definition ignores completely the consequence of a timing failure. In our example, if the failure of an on-time recovery could cause loss of life and property, we would require a rigorous demonstration that the timing failure probability is indeed never more than 10^{-5}. However, we would not require a demonstration of nearly the same rigor for a credit validation.

2.3.2 Hard Timing Constraints and Temporal Quality-of-Service Guarantees

In most of this book, we adopt a simple operational definition: The timing constraint of a job is hard, and the job is a hard real-time job, if the user requires the validation that the system

always meet the timing constraint. By *validation*, we mean a demonstration by a provably correct, efficient procedure or by exhaustive simulation and testing. A large part of this book is devoted to efficient validation algorithms and methods as well as scheduling and resource management strategies that allow the system to be thus validated.

On the other hand, if no validation is required, or only a demonstration that the job meet some *statistical constraint* (i.e., a timing constraint specified in terms of statistical averages) suffices, then the timing constraint of the job is soft. The satisfaction of statistical constraints (e.g., the average number of missed deadlines per minute is two or less) can usually be demonstrated with a performance profile somewhat more thorough than those used to demonstrate the performance of general interactive systems. Most of the techniques for validation discussed in later chapters are not needed.

This way to differentiate between hard and soft timing constraints is compatible with the distinction between *guaranteed* and *best-effort* services [Lock86, Clar90]. Stated another way, if the user wants the temporal quality (e.g., response time and jitter) of the service provided by a task guaranteed and the satisfaction of the timing constraints defining the temporal quality validated, then the timing constraints are hard. On the other hand, if the user demands the best quality of service the system can provide but allows the system to deliver qualities below what is defined by the timing constraints, then the timing constraints are soft.

We call an application (task) with hard timing constraints a hard real-time application and a system containing mostly hard real-time applications a hard real-time system. For many traditional hard real-time applications (e.g., digital controllers), all the tasks and jobs executed in every operation mode of the system are known a priori. The traditional approach to building and validating such systems is to avoid hardware and software features that may lead to nondeterminism. Therefore, it is possible to verify the satisfaction of all hard timing constraints by exhaustive simulation and testing. Indeed, until recently, this has been the only approach used to build hard real-time systems.

In recent years, several efficient validation methods for a large class of hard real-time applications have been developed. These methods make on-line validation feasible and, thus, make hard real-time applications that dynamically create and destroy tasks feasible. When an application creates a new task with hard timing constraints, it submits an admission request to the scheduler. Upon the receipt of such a request, the scheduler does an acceptance test to determine whether the system can meet the timing constraints of the new task while meeting all the hard timing constraints of tasks previously admitted into the system. The scheduler accepts and admits the new task to the system only when the task passes the acceptance test. This acceptance test is an on-line validation test. Many of the validation algorithms described in Chapters 6–9 are suitable for this purpose.

2.4 HARD REAL-TIME SYSTEMS

The requirement that all hard timing constraints must be validated invariably places many restrictions on the design and implementation of hard real-time applications as well as on the architectures of hardware and system software used to support them. To justify this requirement, this section examines briefly several examples of hard real-time systems and discuss why hard timing constraints are imposed and why users require their satisfaction be validated and guaranteed.

2.4.1 Some Reasons for Requiring Timing Guarantees

Many embedded systems are hard real-time systems. Deadlines of jobs in an embedded system are typically derived from the required responsiveness of the sensors and actuators monitored and controlled by it. As an example, we consider an automatically controlled train. It cannot stop instantaneously. When the signal is red (stop), its braking action must be activated a certain distance away from the signal post at which the train must stop. This braking distance depends on the speed of the train and the safe value of deceleration. From the speed and safe deceleration of the train, the controller can compute the time for the train to travel the braking distance. This time in turn imposes a constraint on the response time of the jobs which sense and process the stop signal and activate the brake. No one would question that this timing constraint should be hard and that its satisfaction must be guaranteed.

Similarly, each control-law computation job of a flight controller must be completed in time so that its command can be issued in time. Otherwise, the plane controlled by it may become oscillatory (and the ride bumpy) or even unstable and uncontrollable. For this reason, we want the timely completion of all control-law computations guaranteed.

Jobs in some nonembedded systems may also have hard deadlines. An example is a critical information system that must be highly available: The system must never be down for more than a minute. Because of this requirement, reconfiguration and recovery of database servers and network connections in the system must complete within a few seconds or tens of seconds, and this relative deadline is hard.

A frequently asked question is how serious is the consequence of a few missed deadlines. A real-time monitor may nevertheless function satisfactorily when some sensor readings are not processed or lost. A single late recovery job may not cause an information system to crash. We surely have more design options and can make the system better in some other respect and make it less costly if some of the hard timing requirements are relaxed, even to a small degree.

In recent years, this observation motivated a variety of approaches to soften hard deadlines. Examples are to allow a few missed deadlines (e.g., [HaRa]) or premature terminations (e.g., [LLSB, LLSC]) as long as they occur in some acceptable way. We will discuss some of these approaches in Chapter 10. Needless to say, these approaches can be applied only when application domain experts know the effects of missed deadlines. Unfortunately, this is sometimes not the case. Even for some simple monitor and control applications, it is difficult to assess the effects of lost sample readings and late commands. In more complex systems, such as the NAVSTAR system,[1] the effect of missed deadlines may be combined with other factors

[1]The NAVSTAR Global Positioning System [DoEl] is a distributed system of space-based and ground-based computers and communication links. The system allows users equipped with navigation receivers to determine accurately their own locations. The space subsystem is a constellation of satellites. Together, the satellites provide 24-hour coverage at all locations. On board each satellite, telemetry and track-control subsystems, as well as other subsystems, communicate with each other via the Mission Data Unit (MDU). MDU contains hardware for timing control, modulation control, and navigation. It also interfaces with the intersatellite link and the downlink. The former supports communication among the satellites. The latter allows the satellite to broadcast to the control system on the ground as well as to its users. Each satellite must periodically estimates its own location. The satellites do this in a cooperative manner by exchanging messages with other satellites that are in range. By measuring the differences in the delays severed by messages from other satellites, each satellite can determine its own location with respect to the locations of the satellites whose messages are used for this purpose. This process is called ranging and is an example of functions that require accurate clock and timing signals and has real-time constraints.

in ways impossible to predict. Consequently, the designer makes sure that the system misses no deadline as long as it is in operation. The hard real-time requirement in fact simplifies the process of validating the overall system.

In general, if safety or property loss is involved, the designer/builder of the system has the burden of proof that bad things will never happen. Whenever it is not possible to prove without doubt that a few timing constraint violations will not jeopardize the safety of users or availability of some critical infrastructure, we take the safe approach and insist on the satisfaction of all timing constraints, even though the requirement may be unnecessarily stringent.

2.4.2 More on Hard Timing Constraints

The above examples also show that there may be no advantage in completing a job with a hard deadline early. As long as the job completes by its deadline, its response time is not important. In fact, it is often advantageous, sometimes even essential, to keep jitters in the response times of a stream of jobs small. (Section 1.4.2 gives an example.) In this case, we do not want to complete the jobs too early or too late. In later chapters, you will see that we often choose to delay the start and completion of hard real-time jobs, in favor of soft real-time or background jobs, and this is the reason.

In principle, our definition of hard and soft timing constraints allows a hard timing constraint to be specified in any terms. Examples are

1. deterministic constraints (e.g., the relative deadline of every control-law computation is 50 msec or the response time of at most one out of five consecutive control-law computations exceeds 50 msec);
2. probabilistic constraints, that is, constraints defined in terms of tails of some probability distributions (e.g., the probability of the response time exceeding 50 milliseconds is less than 0.2); and
3. constraints in terms of some usefulness function (e.g., the usefulness of every control-law computation is 0.8 or more).

In practice, hard timing constraints are rarely specified in the latter two ways. We mostly use deterministic hard timing constraints in this book, as in real-time systems literature. A good question is why. The answer is that it is much easier to validate deterministic timing constraints than probabilistic constraints and those expressed in terms of usefulness functions. We will discuss what some of the difficulties are in Chapter 6.

2.5 SOFT REAL-TIME SYSTEMS

A system in which jobs have soft deadlines is a *soft real-time system*. The developer of a soft real-time system is rarely required to prove rigorously that the system surely meet its real-time performance objective. Examples of such systems include on-line transaction systems and telephone switches, as well as electronic games. The less rigorous validation required of the system and, often, more relaxed timing constraints allow the developer to consider other performance metrics equally seriously. Meeting all deadlines is not the only consideration,

sometimes, not even the primary consideration. An occasional missed deadline or aborted execution is usually considered tolerable; it may be more important for such a system to have a small average response time and high throughput.

A system may have critical timing requirements but is nevertheless considered to be a soft real-time system. An example is a stock price quotation system. It should update the price of each stock each time the price changes. Here, a late update may be highly undesirable, because the usefulness of a late price quotation decreases rapidly with time. However, in a volatile market when prices fluctuate at unusually high rates, we expect that the system cannot keep up with every price change but does its best according to some criteria. Occasional late or missed updates are tolerated as a trade-off for other factors, such as cost and availability of the system and the number of users the system can serve.

The timing requirements of soft real-time systems are often specified in probabilistic terms. Take a telephone network for example. In response to our dialing a telephone number, a sequence of jobs executes in turn, each routes the control signal from one switch to another in order to set up a connection through the network on our behalf. We expect that our call will be put through in a short time. To ensure that this expectation is met most of the time by the network, a timing constraint may be imposed on this sequence of jobs as a design objective (e.g., the sequence must complete in no more than 10 seconds for 95 percent of the time and in no more than 20 seconds for 99.95 percent of the time). The users are usually satisfied if after extensive simulation and trial use, the system indeed appears to meet this requirement.

As a final example, let us consider multimedia systems that provide the user with services of "guaranteed" quality. For example, a frame of a movie must be delivered every thirtieth of a second, and the difference in the times when each video frame is displayed and when the accompanied speech is presented should be no more than 80 msec. In fact, it is common to subject each new video stream to be transmitted by a network to an acceptance test. If the network cannot guarantee the satisfaction of timing constraints of the stream without violating the constraints of existing streams, the new stream is rejected, and its admission is requested again at some later time. However, the users are often willing to tolerate a few glitches, as long as the glitches occur rarely and for short lengths of time. At the same time, they are not willing to pay the cost of eliminating the glitches completely. For this reason, we often see timing constraints of multimedia systems guaranteed on a statistical basis, (e.g., the average number of late/lost frames per minute is less than 2). Moreover, users of such systems rarely demand any proof that the system indeed honor its guarantees. The quality-of-service guarantee is soft, the validation requirement is soft, and the timing constraints defining the quality are soft.

2.6 SUMMARY

This chapter defines several terms that will be used frequently in subsequent chapters. They are jobs, tasks, and timing constraints. Most of this book focuses on timing constraints that can be expressed in terms of release times and deadlines of jobs. In particular, we say that the scheduler works correctly if it never schedules any job before the release time of the job. A correctly scheduled job meets its timing constraint if it completes by its deadline, or it completes by its deadline with at least a certain probability, and so on.

The timing constraint of a task can be hard or soft, depending on whether a rigorous validation of the timing constraint is required (hard) or not (soft). In practice, a hard real-time system invariably has many soft real-time jobs and vice versa. The division is not always as obvious as we made it out to be here and, moreover, is not always necessary. In subsequent chapters, we will use the simpler terms *real-time system* or *system* whenever we mean either a hard real-time system or a soft real-time system or when there is no ambiguity about which type of system is meant by it.

We will focus on how to design a system so it is possible to validate its timing constraints and how to do validation if validation is required. In general, the process of validating that a system indeed meets its real-time performance objectives involves three major steps. The first step ensures that the timing constraints of each application are consistent with its high-level real-time requirements and that the timing constraints of its individual components are mutually consistent. The second step ensures that every component can meet its timing constraints if it executes alone and has all the required resources. The third and last step ensures that when scheduled together according to the algorithms used by the underlying operating system and networks, the timing constraints of all applications competing for all the available system resources are always met. In other words, the first step verifies that the timing constraints are specified correctly. The second step verifies the feasibility of each component with the underlying hardware and software resources. The last step verifies that the system as a whole behaves as specified by its timing constraints.

You may have noticed that in our discussion thus far the term validation has been used to mean specifically the last step of the validation process. Indeed, the book focuses on the last step of the validation process. We assume that the correct specification of the timing constraints and the feasibility of every individual component have already been verified. Therefore, the timing constraints of the system are given. For methods on specifying timing constraints and verifying their correctness, you need to read books and articles on formal methods (e.g., [Heit, VaKo]). Similarly, you can find algorithms for finding processor time demands of jobs in [AMWH, HeWh, KiMH, LBJR].