# Toward a Taxonomy of Time-Constrained Applications

Scott Banachowski and Scott A. Brandt
*University of California, Santa Cruz*
E-mail: {sbanacho,sbrandt}@cs.ucsc.edu

## Abstract

*We are developing a taxonomy for classifying applications based on their time constraints. The taxonomy is based on three components that capture the key characteristics between different classes of applications: the process's execution behavior, the timing constraints it requests, and the level of guarantee it is willing to accept from the system. The taxonomy is useful for two reasons: it provides a survey of the types of existing real-time scheduling applications, and it is a tool for understanding how to develop integrated schedulers that handle multiple classes of applications simultaneously.*

## 1 Introduction

Computer systems continue to grow in both capability and complexity. Embedded, special-purpose and general-purpose systems are expected to simultaneously handle a variety of application workloads. Our research focuses on integrated CPU scheduling of different classes of time-sensitive applications. A *class* is a set of applications sharing similar timeliness constraints. To address the large number of existing classes (culled from real-time research literature), it is useful to be able to describe and compare different sets of timeliness constraints. The paper describes a taxonomy for classifying different types of applications.

There are three components that describe a class: (1) behavior, which captures how an observer sees the process execute; (2) constraint, which describes how the application expresses its processing needs; and (3) guarantee, which is how an operating system handles the constraints of the application. A class is defined by a tuple consisting of types for each of these three components (types are described further in Section 2.2).

## 2 Classification of Applications

Applications belong to one of three *families*: hard real-time (RT), soft real-time (SRT), and best-effort (BE). Generally, these terms describe the criticality of time constraints or the nature of guarantee required from an operating system. To create a taxonomy that is flexible in describing different classes of applications, it is necessary to identify more specific descriptions of the types of processing models, constraints, and guarantees.

### 2.1 Processing Families

Hard real-time processes have timeliness constraints that must be met—not meeting a constraint is considered an error, in some cases leading to catastrophic failure of the system. A *guarantee* is a contract between a process and the system that a constraint will be met. A hard real-time system is one designed specifically to make and meet guarantees of hard real-time processes. When a hard real-time system runs an RT task, a guarantee is implied. Admission control or off-line feasibility checks are the traditional methods to negotiate guarantees in a hard real-time system, and are based on the worst-case execution requirements.

Soft real-time processes have timeliness constraints, but failing to meet constraints leads to performance degradation instead of error. Although not required for SRT processes, guarantees are often desired. For example, a video-conferencing application tolerates dropped frames without severely impeding a conversation, but many missed frames or delays is distracting enough to make execution not worthwhile without a minimum guarantee. Often, the requirements of a soft real-time task are characterized with average-case rather than worst-case estimates [17]; a guarantee may ensure that a task performs well on average, assuming it may accept degradation in the worst case. A soft real-time system is one that specifically supports one or more classes of soft real-time processes, usually attempting to maximize an aspect of system performance, such as maximizing the total utility of values attached to meeting deadlines [8].

Best effort processes have no timeliness constraints, and execute with whatever resources are available to them. A best-effort system is one that makes no guarantees to processes. The goal of a most BE systems, including general-purpose time-sharing systems [13, 2], is fairly allocating resources and maintaining good response time.

### 2.2 Classes of Processes

Each processing family has different timing constraints. More importantly, the way that applications wish to express these constraints to an operating system may differ, as may the guarantees they are willing to accept from the system. Even applications of the same family may differ in the information the system must know in order to properly schedule them. For example, real time task's temporal constraints may be strictly deadline-based when processing a frame of data, or rate-based when keeping a buffer from under-flowing. To classify tasks, we have developed a taxonomy, where each class is described by a three-tuple of components:

1. The processing behavior of the application. The behavior describes how and when the process uses the CPU resource, without regard to any specific timeliness requirements. Table 1 lists the types of behaviors.

2. The parameters used to specify requirements. The requirements are the specific constraints that applications request from operating systems. The way constraints are represented (for example in units of time) or the interfaces used to communicate them to a system vary depending on the implementation.[1] Table 2 lists the types of parameters.

3. The guarantee that the process is willing to accept from the operating system as a criteria for executing. Table 3 lists the types of guarantees.

### 2.3 Family Taxonomies

A class of an application is identified by the three components that describe its execution model, constraint model, and guarantee requirement. Using the components above, we may now distinguish which classes of applications belong to each of the three processing families:

**Best-effort** Members of the best-effort family of applications may use any execution model, may specify any type of constraint, but never require any guarantees from the operating system.

```
BE := any-any-NONE
```

**Soft real-time** Members of the soft real-time family of applications use any execution model except for IO and CPU. They may specify any type of constraint, and although they may require some guarantee in order to run, it is never strict.

```
SRT := [PERIODIC|RATE|APERIODIC|SPORADIC]-
any-[NONE|MINIMUM|AVERAGE]
```

[1] Therefore it is assumed that, for any system, the constraints are expressed in a form that matches the kind of guarantees the operating system scheduler provides.

**Hard real-time** Members of the hard real-time family of applications use any execution model except for IO, CPU or APERIODIC (aperiodic applications cannot be supported because unbounded arrival rates make guarantees impossible). Constraints include worse case estimates or require continuous resource and strict guarantees, except for the imprecise model which allows minimum guarantees.

```
RT := [PERIODIC|RATE|SPORADIC]-
[DEADLINE+WCE|RATE]-STRICT |
[PERIODIC|RATE|SPORADIC]-{DEADLINE}-
MINIMUM
```

## 3 Class Equivalence

With the creation of a taxonomy, we plan to supply rules for scheduling different classes using the same or similar algorithms, i.e. for treating different classes equivalently for scheduling purposes (often independently of the scheduling algorithm). The following theorem is adapted from the HLS framework [16].

**Theorem 1** *Any schedule that guarantees a rate $x$ over interval $y$ to a task of type a-*RATE*-b will also guarantee that a task of type a-*DEADLINE+WCE*-b with a worst-case execution time $x$ and deadline $y$ meets its deadline.*

**Proof:** If a process is given of continuous guarantee of $x$ units in every interval of length interval $y$, then any job with release time $r$ and absolute deadline $r+y$ will receive at least $x$ units before its deadline.

Theorem 1 demonstrates orthogonal properties of different classes. Showing that a scheduler may treat classes equivalently simplifies the creation of systems that support multiple classes. Applying the above theorem lets us execute the two classes on the same scheduling algorithm without a transformation of parameters.

The inverse of the above theorem is also true, but requires a transformation of constraints:

**Theorem 2** *Any schedule that guarantees that a task of type a-*DEADLINE+WCE*-b with a worst-case execution time $x$ and deadline $y$ will meet its deadline will also guarantee that a task of type a-*RATE*-b will meet a guarantee of rate $x$ over interval $2y-x$.*

**Proof:** If a process is meets deadlines requiring $x$ over every period of $y$, there isn't a guarantee that it will receive $x$ over any interval $y$, only for intervals beginning at the periodic release time. However, in the worst case, a periodic processes may receive all of its processing at the beginning of its first period, and at the end of the subsequent period. Thus over any interval of $2y-x$, it must receive at least $x$.

**Table 1. Types of Processing Behavior**

| Type | Description |
|---|---|
| CPU | A CPU-bound task consumes long bursts of CPU without interruption, such as a scientific application that does complicated mathematical computations. |
| IO | An IO-bound task spends more time waiting for input than consuming CPU. A command shell is an example; it mostly waits for keystrokes that usually require little computation to process. |
| PERIODIC | A periodic task consumes CPU at fixed, periodic intervals. A video player that decodes frames as they are displayed is an example, and its period is the inverse of the video frame rate. |
| RATE | A rate-based task proceeds according to a fixed rate, consuming roughly the same amount of CPU during any interval of its execution. Network packet processing for fixed-rate stream of a router is an example. |
| APERIODIC | An aperiodic task releases job at an irregular rate. An example is an interrupt that services a device that is triggered by an external, random event. |
| SPORADIC | A sporadic task releases jobs at an irregular rate; unlike an aperiodic task, the rate is bounded. A keyboard interrupt is triggered at an irregular rate, but is bounded by the speed of the keyboard bus. |

**Table 2. Types of Processing Constraints**

| Type | Description |
|---|---|
| NONE | No constraint, although most multi-tasking systems support at minimum a relative time-sharing priority, such as the process *nice* parameter settable in UNIX-based operating systems. It allows adjustment above or below the default "fair-share" of CPU assigned to processes. |
| DEADLINE | A task may specify a deadline for each job without knowing its the execution time of jobs (a video player where the execution time is data dependent, for example); in this case the only time constraint is that each job finishes before its deadline. In many cases, for periodic processes deadline is the same as period. Without knowing anything about execution time, it is impossible to know if a task is schedulable. |
| DEADLINE+WCE | If both the period and worst-case execution of any job of a task is known, task constraints may be used for hard guarantees. This constraint type is used to check for schedulability in the periodic task model [10]. |
| DEADLINE+ACE | For variable period workloads, such as media in which the processing depends on the content of the data, the average case execution time of jobs may be known (also known as a variable processing time class [4]). Knowing a model of the variability, it is possible to provide some probabilistic guarantees [17, 5] |
| RATE | The task must make a fixed amount of progress during any fixed-length, but arbitrarily positioned, time interval [7] (also called the continuous class [9]). Alternately, an equivalent specification is expressed using an interval and a percentage of CPU [14]. |
| FIRM | The task must meet $m$ out of $k$ deadlines [6]. |
| {DEADLINE} | An imprecise constraint specification states a mandatory deadline constraint for each job, with optional deadlines associated with subprocesses that may be met in best effort fashion [11]. |
| any+u() | Any of the above constraints may be paired with a utility function that associates a value with meeting the constraint. Utilities functions may consist of a fixed value or be represented as a time-dependent curve [8, 12]. |
| {any+u()} | Any of the above parameters may be specified as of set of multiple constraints—these applications adapt to available resources by adjusting their constraints. These systems adapt allowing the system allowing dynamic QoS control [18, 15]. |

**Table 3. Types of Processing Guarantees**

| Type | Description |
|---|---|
| NONE | The task is able to run without any guarantee. This is provided by best-effort systems, and many SRT real-time applications are willing to accept it; obviously, RT applications cannot. |
| MINIMUM | The task may run only if it receives a minimum amount of resource. This guarantee supports imprecise constraints and many SRT applications. |
| AVERAGE | The task is willing to except a reservation in which constraints are not met all the time, but are met on average. Most SRT systems provide these kinds of guarantees [17, 1, 17, 5, 3]. |
| STRICT | The task may only run if it is guaranteed to meet all timeliness constraints. Hard real-time systems are designed to provide this type of guarantee. |

Like the above, in many cases we expect classes to be equivalent only after a transformation of constraints:

**Theorem 3** *Any schedule that provides an imprecise guarantee of x units to a task $T_1$ of type with* PERIODIC-DEADLINE-MINIMUM *can also a schedule a task $T_2$ of type* PERIODIC-DEADLINE+ACE-AVERAGE*, by converting the average use estimate to an imprecise guarantee.*

**Proof:** Part of $T_2$'s constraint is a specification of the average case distribution of CPU usage for each period. Knowing the distribution, enough CPU per period may can be reserved so that on average, the process will meet its deadline. For example, if the mean is close to the median, and the task receives at least a minimum CPU equal to its mean, then it will meet half of its deadlines. An imprecise guarantee can ensure that it receives a minimum of CPU each period, therefore by converting the constraints of $T_2$ to the imprecise constraint of $T_1$ the process will receive an average guarantee.

The future work is to establish an entire set of rules and transformations for identifying equivalent ways to schedule many types of classes.

## 4 Conclusion

The taxonomy provides a way to describe the multiple classes of time-constrained applications taken from both literature and experience. To complete the taxonomy we must identify any missing types and a full set of equivalence classes like those discussed above.

We will also develop flexible operating system interface for processes to specify their processing constraints in a general way.

## References


[1] A. K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, Madrid, Spain, Dec. 1998. IEEE.

[2] M. Beck, H. Böhme, M. Dziadzka, U. Kunitz, R. Magnus, and D. Verworner. *Linux Kernel Internals*. Addison–Wesley, 2nd edition, 1998.

[3] H. Chu and K. Nahrstedt. A soft real time scheduling server in UNIX operating system. In *European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, Sept. 1997.

[4] H. Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *Proceedings of the 1999 IEEE International Conference on Multimedia Computing and Systems (ICMCS '99)*, June 1999.

[5] M. K. Gardner and J. W. Liu. Analyzing stochastic fixed-priority realtime systems. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Joint European Conferences on Theory and Practice of Software*, volume 1579 of *Lecture Notes in Computer Science*, pages 45–58. Springer-Verlag, Mar. 1999.

[6] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, Apr. 1995.

[7] K. Jeffay and D. Bennett. A rate-based execution abstraction for multimedia computing. In *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, Apr. 1995.

[8] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *Proceedings of the 6th IEEE Real-Time Systems Symposium (RTSS 1985)*, Dec. 1985.

[9] M. B. Jones, D. Roşu, and M.-C. Roşu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97)*, pages 198–211, Oct. 1997.

[10] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[11] J. W. Liu, K. Lin, W. Shih, A. C. Yu, J. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 25(5):58–68, May 1991.

[12] C. D. Locke. *Best-Effort Decision Making for Real-Time Scheduling*. PhD thesis, Carnegie-Mellon University, May 1986.

[13] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System*. Addison–Wesley, 1996.

[14] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the 1994 IEEE International Conference on Multimedia Computing and Systems (ICMCS '94)*, pages 90–99, May 1994.

[15] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS 1997)*, Dec. 1997.

[16] J. Regehr. *Using Hierarchical Scheduling to Support Soft Real-Time Applications on General-Purpose Operating Systems*. PhD thesis, University of Virginia, May 2001.

[17] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. Wu, and J. W. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the Real-Time Technology and Applications Symposium (RTAS95)*, pages 164–173, May 1995.

[18] H. Tokuda and T. Kitayama. Dynamic QoS control based on real-time threads. In *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 114–123, 1993.