Technology Maturation Report: Real-Time Scheduling Algorithms  
Theresa Maxino  
17-939, Spring 2005

Scope of the Study  
This study focuses on the maturation of real-time scheduling algorithms for hard real-time systems.

Introduction  
In the world we live in today, we are surrounded by embedded systems. Systems which we don’t give a second thought to and which we would not typically consider as computer systems. Things like cars, airplanes, trains, microwave ovens, and thermostats to name a few. All these have computers, or more specifically, microcontrollers embedded in them. These are examples of real-time systems, systems which interact directly with their environment. In these systems, the concept of temporal correctness is as important as functional correctness. You would not buy a car that was unpredictable in the time it took to brake after you pressed your foot against the pedal. In these real-time systems, having a schedule that works is the key to everything.

Definition of Terms  
Hard real-time systems are systems where the failure to meet a specified deadline could lead to catastrophic consequences. An example of this would be the braking system in a car. If the braking system does not perform its intended function at the correct time then lives could be lost.

Soft real-time systems, in contrast, are systems where the failure to meet a specified deadline reduces the utility of the result and could possibly lead to financial loss, but does not lead to loss of life. An example would be an online stock trading system.

Static or Offline scheduling is a type of scheduling where the schedule of tasks is determined before the application is executed. Dynamic or Online scheduling, on the other hand, is a type of scheduling where the schedule of tasks is determined during application execution.

Another way to classify scheduling is as preemptive and non-preemptive. In preemptive scheduling, the current task’s execution is suspended if a higher priority task arrives. In non-preemptive scheduling, a task is allowed to finish execution without interruptions, even though a higher priority task might arrive.

Task priority can be classified as static and dynamic. A task’s priority is said to be static if it does not change during application execution. It is said to be dynamic if priority can change while the application is running.

Tasks can be generally classified as periodic or aperiodic. Periodic tasks are tasks which occur at fixed time intervals. They are time-driven and are referred to as time-triggered tasks. An example of a periodic task is a task that polls the temperature sensor of a thermostat. Aperiodic tasks are tasks whose occurrences are unpredictable. They are
event-driven and are referred to as event-triggered tasks. An example is the heating function that is activated when the user presses the start button in a microwave oven.

Technology Timeline

Using the Redwine/Riddle technology maturation model [33], a technology’s development can be divided into the following phases: basic research, concept formation, development and extension, enhancement and exploration (internal and external), and popularization. For real-time scheduling algorithms, basic research started in the 1960’s. It then moved on to the concept formation phase in the late 1960’s and continued on until the early 70’s. The next 2 phases, development and extension, and enhancement and exploration (internal and external), have kind of blurred together. To make more sense, they have been grouped into the 3 main areas of research that ensued after the Concept Formation phase. In Figure 1 they can be seen as Multiprocessors, Distributed Nodes, and Periodic & Aperiodic Tasks. These phases span from the late 1970’s to the present. Popularization started in the 1990’s and still continues on in the present day.

![Figure 1. Real-Time Scheduling Algorithms Timeline](image)

Basic Research

In the 1960’s, all task scheduling was done by hand. This was a time-consuming process and more often than not, it could not guarantee that all timing constraints would be met. Task scheduling generally followed the “best fit” principle [1] and the most important tool for the system designers was the Gantt chart. One example of the algorithms developed during this time was the Stability algorithm [3] which relied heavily on the use of Gantt Charts.
Concept Formation

In 1968 Lampson came up with his paper on scheduling theory [4]. This paper introduced most of the concepts and terms we now use in scheduling today. In the early 70’s, two of the seminal papers on scheduling algorithms were published. The first, on preemptive scheduling for multiprocessors, was published by Muntz and Coffman in 1970 [5]. Three years later, in 1973, Liu and Layland published their paper [6] on optimal preemptive scheduling for single processors. They presented two scheduling algorithms, a static preemptive algorithm, Rate Monotonic Scheduling (RMS), and a dynamic preemptive algorithm, Deadline-Driven or Earliest Deadline First (EDF), as it is more commonly known today. RMS assigns the highest priority to the task with the shortest period. EDF, as the name implies, assigns the highest priority to the task with the earliest future deadline.

Development and Extension, Enhancement and Exploration

Multiprocessors

The late 70’s and early 80’s were dominated by papers on multiprocessor preemptive scheduling. The main reason for this was the fact that at this time microcomputers and microcontrollers were still new. Focus of scheduling was on getting the most from the big mainframes prevalent at that time. The papers published during this time can best be classified under the Development and Extension phase of the Redwine/Riddle model [33]. These papers were extensions of the algorithm proposed by Muntz and Coffman [5]. They addressed scheduling on more than 2 processors [7], uniform processors [8], and unrelated processors [9]. In the early 80’s papers began addressing the problem of scheduling tasks with release times, due times, and deadlines [10][11]. These were still on multiprocessors, but, they were closer to the issues dealt with by the Liu and Layland paper [6].

Distributed Nodes

In the mid 1980’s microcomputers and microcontrollers were becoming more prevalent and were becoming cheap enough for them to be used in a number of applications, like in avionics and cars. These systems typically have a number of single processor nodes connected to each other over some sort of network. It was not surprising then for the focus of research to shift to single processor scheduling from multiprocessor scheduling. Research done at this time can best be classified in the External Enhancement and Exploration phase of the Redwine/Riddle model [33].

To support distributed nodes, a reliable method of communication between nodes needed to be developed. This gave rise to bus scheduling. Not only did the tasks on the nodes have to meet deadlines, but also, the messages they passed to each other had to be sent and received on time. Lehoczky and Sha first explored this area in their 1986 paper when they applied RMS to bus scheduling [12]. In 1999 Shirero, Takashi, and Kei proposed the real-time round robin scheduling algorithm [26]. Though they did not directly apply their algorithm to bus scheduling, the algorithm finds many similarities in the bus scheduling algorithms used currently. In more recent times, Pop, Eles, and Peng explored the scheduling and schedulability issues of mixed time- and event-triggered distributed systems [28].
Other things investigated in the area of distributed nodes are distributed scheduling and non-symmetric scheduling. Distributed scheduling looked into scheduling tasks deemed to be essential but non-critical [14]. Non-symmetric scheduling investigated the issue of periodic tasks not having the same duration every time [20].

**Periodic and Aperiodic Tasks**

The late 1980’s saw the start of a flurry of papers on the issue of scheduling periodic and aperiodic tasks in a system. This is an important issue because a typical system does not only have periodic tasks, but also, aperiodic tasks. Though these aperiodic tasks are generally not considered critical, they are still essential for the system to function optimally. Ensuring that both periodic and aperiodic tasks meet their deadlines is thus an important consideration for any system designer.

The papers in this area can generally be classified in the Development and Extension and Internal Enhancement and Exploration phases of the Redwine/Riddle model [33]. They typically extend the RMS and EDF scheduling algorithms.

Some of the earliest algorithms proposed to deal with the issue of periodic and aperiodic scheduling are the Priority Exchange (PE) [13], Deferrable Server (DS) [13], and Immediate Server algorithms [16]. These algorithms create a periodic server that services aperiodic tasks. This periodic server is scheduled as one of the periodic tasks, and thus RMS can still be used.

The EPE [13], slack stealing [19], and resource reclaiming [21] algorithms use a different approach to service aperiodic tasks. Periodic tasks are typically scheduled using their Worst Case Execution Times (WCET). These times are generally much larger than the task’s average execution times. The difference between a task’s average execution time and WCET is used by the 3 above-mentioned algorithms to service aperiodic tasks.

The Critical Task Indicating (CTI) [23] [25], Earliest Deadline first with larger Value (EDV) [30], and Value first with Earliest Deadline (VED) [30] algorithms all use tables created offline to help with the scheduling. For CTI the table is used to determine if an aperiodic task can be allowed to run given the critical tasks indicated in the table. For EDV and VED, the table is used to specify a certain task’s priority.

**Popularization**

The popularization phase in the Redwine/Riddle model [33] of real-time scheduling algorithms started in the 1990’s. The reason for this early popularization was the fact that there were already a number of existing real-time systems using the cyclic executive approach of scheduling. During the early part of the popularization phase, these existing systems were taken and refitted with the RMS scheduling approach instead of the cyclic executive approach.

Locke, Vogel, and Mesler showed how RMS can be used to build predictable avionics platforms in Ada [17]. Sha, Rajkumar, and Lehoczky showed RMS support in the Futurebus+ specification [15]. Audsley, Bate, and Burns demonstrated that fixed priority scheduling can replace cyclic executives in safety critical systems like avionics [24].
Real-time scheduling algorithms play a major part in the network scheduling protocols used by cars and avionic systems. Examples of these are the Controller Area Network (CAN) [18], Time-Triggered Protocol (TTP) [22], and FlexRay Protocol [32].

A number of real-time operating systems are available nowadays like VxWorks, RTX, and QNX. These typically use RMS for their scheduling algorithms. Middleware like Real-time CORBA [27] and Real-time Java also support RMS.

Research Method

Most of the papers in the real-time scheduling area fall under the Newman Pro Forma Abstract classification of Enhanced Solution [34]. The seminal papers like the RMS paper [6] can be classified as a Radical Solution paper [34]. Using Shaw’s classification types [35], the papers typically have the following characteristics: Question = Development Method, Result = Technique, and Validation = Analysis.

The papers generally use the same research method, that of “Proofs validated by example”. This type of research method is utilized by papers in the Concept Formation, Development and Extension, and Enhancement and Exploration phases of the Redwine/Riddle model [33].

The typical research paper has the following 3 parts: Algorithm Description, Theorems & Proofs, and Validation.

The first part, algorithm description, generally presents similarities with existing algorithms, deficiencies of the existing algorithms, and improvements over the existing algorithms. An explanation of the new proposed algorithm is then presented.

In the next part, theorems & proofs, mathematical/logical proofs are presented to show the correctness and feasibility of the algorithm. Papers presenting entirely new algorithms rely more heavily on this than papers presenting extensions of existing algorithms.

The last part, validation, can be presented in a variety of ways. Papers can use simple examples augmented with explanations, provide sample code, or show simulation results of the proposed algorithm in comparison to existing algorithms. Paper presenting extensions of existing theories rely more heavily on this than papers presenting entirely new theories.

It may be asked, “Why the need for validation when it has already been proven to be mathematically correct?” The answer to this is that when the algorithms are proven to be mathematically correct, certain assumptions are made. However, when the algorithms are used in real-world applications, these assumptions may not hold. It is to show that these assumptions still hold that the validation part is included. In some cases, the assumptions do not hold and the validation part shows which algorithm performs best under these scenarios, an example of which is a transient overload case.
Annotated Bibliography


   *** doesn’t seem to be very good. Based on Machine Shop Scheduling. Uses Gantt charts.

   *** talks about tasks, deadlines, soft real time and hard real time; scheduling is thru a task list which contains a list of all tasks in order of servicing priority; still uses Gantt charts; introduces stability algorithm (Standard Gantt Chart projective task list)

   *** introduces concept of scheduler and processes which can be either ready, blocked, running or suspended; ready list, priority list, wakeup list, sleep, wakeup, preemption are also introduced

   *** optimal preemptive scheduling algorithm for multiprocessor systems

   *** proposes 3 algorithms: rate-monotonic (static), deadline-driven (dynamic) [same as EDF], mixed

   *** level algorithm extension of Muntz and Coffman for any number of processors

*** extension of Horvath. Provides time complexity of $O(n)$ vs. $O(mn^2)$ for Horvath.

   *** shows that no more than $O(m^2)$ preemptions are necessary to schedule $n$ jobs on $m$ unrelated processors

   *** preemptive scheduling for tasks with due times and release times

   *** $O(m^2n^4 + n^5)$ algorithm for $n$ jobs on $m$ uniform machines

   *** RMS applied to bus scheduling

   *** EPE extends the PE algorithm. EPE performs significantly better than polling, background processing, Priority Exchange (PE), and Deferrable Server (DS) especially in cases where the worst case periodic load is high. Based on RMS.

   *** scheduling algorithms for essential (non-critical) tasks in a hard real-time distributed system: random, focused addressing, bidding, flexible

   *** RMS support


*** Immediate Server (IS) algorithm. Periodic server has highest priority but not necessarily the shortest period. Relaxes the rate-monotonic constraint. Uses a feasibility checking algorithm to determine if all deadlines can be met.


*** shows how RMS can be used to build predictable avionics platforms in Ada


*** slack stealing algorithm. Based on RMS.


*** Nonsymmetric Scheduling (NS) algorithm. Based on RMS.


*** Basic Reclaiming and Reclaiming with Early Start.


*** extends the CTI algorithm to include hard aperiodic tasks


*** fixed priority scheduling shown to be a viable alternative to cyclic executive scheduling for safety critical systems like avionics
***proposes the CTI algorithm. Algorithm uses a CTI table created offline to determine if aperiodic tasks will be allowed to execute at runtime. Uses deadlinewise preassignment to schedule tasks offline.

***proposes real-time round robin algorithm. Uses Earliest Deadline First to schedule within task groups. Each task group has its own time slot set. Deadline misses are contained within each task group.


***algorithm for holistic scheduling and schedulability analysis of mixed time- and event-triggered tasks and messages. Bus has static and dynamic slots (TDMA).

***extension of fault-tolerant EDF

***proposes 2 new algorithms: Earliest Deadline first with larger Value (EDV) and Value first with Earliest Deadline (VED)

***extends RMS and EDF to multiprocessors

