
FAULT-TOLERANT ADAPTIVE SCHEDULING FOR EMBEDDED REAL-TIME SYSTEMS

THIS FAULT-TOLERANT ALGORITHM USES A TIME-VALUE SCHEDULING APPROACH TO DETECT FAULTS, SUSTAIN HIGH PROCESSOR UTILIZATION, AND ENSURE TIMELY EXECUTION OF CRITICAL TASKS.

Paul Richardson
University of
Michigan-Dearborn

Larry Sieh
US Army Tank-Automotive
RD&E Center

Ali M. Elkateeb
University of
Michigan-Dearborn

..... Military vehicles, robotic systems, aircraft, and automobiles are among the many applications that rely on complex embedded computer systems to perform critical operations. The real-time tasks these systems execute have specific time constraints and a wide range of values. Failure to meet a task's time constraints can result in degraded performance. Some tasks (for example, vehicle braking and weapons control) are critical in that failure to meet their time constraints can lead to costly damage or serious injury. Such systems must provide long periods of uninterrupted service in harsh and dynamic environments and are subject to extremes in shock, vibration, temperature, and electromagnetic interference (EMI). Inevitably, faults will occur, adversely impacting system behavior. Avoiding catastrophe in the presence of faults requires that these systems sustain the execution of critical tasks in a timely manner.

Faults are malfunctions that result in some type of unspecified system behavior; we can generally classify them as permanent, transient, or intermittent. Permanent faults result from hardware component failure, and having redundant components generally mitigates such faults. Transient faults are temporary malfunctions that eventually subside. Intermittent faults are repeated occurrences of transient faults.

Several studies have concluded that most embedded-system faults are transient.¹⁻³ Their sources include changes in the operational environment, simultaneous arrival of asynchronous events, system exceptions, and hardware-oriented faults in the CPU and peripheral devices related to EMI, power fluctuations, and inaccuracy in electromechanical devices.^{1,2} These faults surface as transient surges in CPU loading, as tasks that need to be reexecuted, or as tasks that exceed their specified execution times or interarrival rates. A transient overload results when transient faults cause processor demand to exceed processor capacity. This makes it inevitable that some tasks will complete late. If critical tasks continue to perform correctly during a transient overload, the system can return to a nominal operating mode without incurring costly outcomes. However, failure to sustain critical tasks can have severe consequences, such as entire system failure, costly damage, or injury.

This article addresses the timely execution of critical tasks during transient faults using the adaptive deadline-monotonic (ADM) scheduling algorithm. Under nonoverload conditions, the time-based deadline-monotonic scheduling (DMS) algorithm schedules all tasks in a timely manner while sustaining optimal processor throughput. During tran-

sient overloads, a value-based scheduling (VBS) algorithm orders task execution on the basis of the tasks' value to the system.

Several important issues affect the ability to sustain critical operations during transient overloads:

- ADM scheduling is practical for embedded systems because it requires no additional hardware and hence avoids hardware's associated weight, space, and dollar costs.
- ADM scheduling provides a tunable sliding-window mechanism to enact scheduling-mode changes from time- to value-based scheduling. This mechanism seeks to avoid the performance degradation associated with switching modes too early or too late.
- ADM scheduling provides a mechanism that detects transient surges during scheduling events.
- During a transient fault, a scheduling algorithm must sustain application task execution to avoid aggravating the situation. To this end, ADM scheduling avoids computationally demanding approaches.

Applications targeted by this effort include the unmanned ground vehicles (UGVs) under development by the US Army Demonstration-3 UGV program. UGVs will perform increasingly important roles in future tactical military operations because they can distance personnel from hazardous situations. The hostile operational environment and the critical nature of military operations make the ability to sustain critical tasks during transient overloads highly desirable. Although ADM scheduling specifically addresses UGVs, the results are applicable to other applications operating under similar constraints.

Background and previous work

Real-time-system development requires assurance that tasks can meet their time constraints. Schedulers play an important role here because they order task execution. Time-based schedulers such as the DMS algorithm are popular because they can meet all time constraints at relatively high levels of CPU utilization. DMS orders the execution of tasks by their deadlines and guarantees all task deadlines will

be met at a minimum CPU utilization of 69 percent. In some cases, utilization can reach 100 percent, depending on the tasks' characteristics. Among fixed-priority scheduling algorithms, DMS can sustain the highest level of processor utilization while guaranteeing all task deadlines will be met.⁴ Mejia-Alvarez, Melhem, and Mosse^{5,12} and Ramos-Thuel and J. Stosnider⁶ provide recent research results concerning fault tolerance and fixed-priority schedulers.

There are many different hardware and software methods for fault detection. Once a transient fault is detected, the next step is response. Slack-stealing—using spare CPU capacity to tolerate transient faults—is an effective response.^{5,6} This approach employs unused system capacity (slack) to respond to a transient fault(s) by reexecuting faulty tasks at their own priority. In another approach, primary/back-up schemes,⁷ a primary process executes, and if no errors are detected, it outputs results. If errors are detected, a back-up process executes instead.

During transient overloads, time-based schedulers perform poorly because they don't consider a task's value when assigning priorities. This results in the unnecessary failure of critical tasks. This drawback was a significant motivation for the emergence of time-value scheduling, in which time and value both serve as scheduling metrics. E. Jensen et al. pioneered time-value scheduling as an approach that considers soft and vague deadlines as well as capturing the effect of system state and time on a task's value.⁸ Several recent efforts follow this approach.^{1,9-12}

Workload characterization and feasibility

UGVs, like many embedded-system applications, are designed to provide one specific set of services over their entire useful life. Thus, the tasks they execute are generally fixed at some point during their development. Additionally, we have significant a priori knowledge regarding the tasks' characteristics. We derive this knowledge from the system's operational requirements and the restrictions the operational environment imposes. Leveraging this information is essential for task-set feasibility analysis.

Workload characterization

UGVs provide two primary services implemented by a set of real-time tasks, $\{\tau_n\}$. A

UGV must provide its own mobility, which includes automotive functions such as steering, braking, and speed control. A UGV also uses an array of sensors (infrared, television, microwave radar) to collect information for the controllers and to help plan local autonomous moves.

When reasoning about the feasibility of $\{\tau_n\}$, we must consider several practical issues. One workload-related issue concerns the operational characteristics of UGVs and similar systems. When a UGV encounters a threat, it activates critical tasks, and system loading peaks—that is, the UGV stops, collects and processes sensor data, determines the local path, and moves away from the hazard. This situation is when tasks have their highest value, and the possibility that transient faults will occur in bursts increases. Thus, transient surges are most likely when critical tasks are executing at peak loading. We developed the ADM scheduling algorithm with this type of scenario in mind.

Another issue developers must face is cost-performance trade-offs when selecting system components such as the operating system, CPU, and I/O channels. In an embedded system, we can specify interarrival rates and task execution times only to a degree of certainty. Typically, feasibility analyses use estimates for worst-case interarrival times (WCIT) and worst-case execution times (WCET). When specifying performance goals, designers find that providing estimates for all tasks may be prohibitively expensive. If worst-case instances are rare or a task is not critical, using more-optimistic estimates, such as average-case values, can produce significant savings. However, for the cost savings to be practical, the system must tolerate situations in which violations of more-optimistic estimates result in transient surges. In either case, tasks exceeding their specified interarrival rates or execution times result in a transient surge in the system loading.

To reason about the feasibility of $\{\tau_n\}$, we make certain assumptions about task interarrival rates, execution times, and deadlines. We assume that $\{\tau_n\}$ is fixed and that task arrivals are independent of each other. T_i represents the WCIT for τ_i , and we assume enough information is available to reasonably estimate WCIT for tasks that arrive sporadically. C_i represents the WCET for tasks, and we can reasonably approximate it. Unique deadlines d_i are speci-

fied for each task, and for every task $d_i \leq T_i$. We denote the j th instance of task τ_i as τ_{ij} , whose arrival time is a_{ij} and deadline is $d_{ij} = a_{ij} + d_i$.

Tasks with *hard deadlines* have no value if they are late, and violated deadlines can have severe consequences that affect responses to alarms, braking, weapons control, and so on. Tasks with *firm deadlines* also have no value once they are late; however, the system can tolerate several late instances. For example, a closed-loop control system operating well above its stability criteria can tolerate isolated instances of late computations with perhaps minor performance degradation. Repeated instances would eventually lead to instability. Tasks with *soft deadlines* can tolerate lateness and retain some diminished value. Many tasks have soft deadlines, with lateness resulting in gradual performance degradation.

Time-based scheduling and feasibility

A set of real-time tasks, $\{\tau_n\}$, is feasible for a system if the system can guarantee all of the tasks' deadlines. Feasibility generally depends on the scheduling algorithm used and task characteristics (arrival rates, execution time, and deadlines). To determine feasibility for $\{\tau_n\}$, we use available information about the task, including the assumptions, and employ the demand calculation that Klien et al. describe.⁴ Demand refers to the amount of execution time a set of tasks requires over a given time interval. The set of tasks we're interested in is the pending tasks $\{\tau'_n\}$ with greater priority than τ_i , and the interval of interest is the critical time zone (CTZ) of τ_i .

We define the CTZ for τ_i as the interval between its arrival and deadline at critical instant CI. We define CI as the moment when all tasks have their worst-case response time (WCRT). For the DMS algorithm, this occurs when all tasks arrive simultaneously. The CTZ interests us because it represents the interval when a task experiences its WCRT. If the total execution time required by $\{\tau'_n\}$ and τ_i is less than or equal to the CTZ of τ_i , then the deadline of τ_i can be met; otherwise it cannot. We formalize the determination of demand for each task τ_i as follows. With tasks ordered by their deadlines in descending order, we calculate the demand d'_i for each task at the CI, when all tasks have their WCRT. For each task τ_i , determine

$$a_{i,0} = \sum_{j=1}^i C_j$$

In this step, $a_{i,0}$ represents the initial estimate of demand for τ_i as the sum of all execution times for $\{\tau'_n\}$ plus the execution time of τ_i . While $a_{i,m} \neq a_{i,m-1}$, determine

$$a_{i,m} = C_i + \sum_{j=1}^{i-1} \left[\frac{a_{i,m-1}}{T_j} \right] C_j \quad (1)$$

In this step, $a_{i,m}$ accounts for the execution time of any task $\tau_j \in \{\tau'_n\}$ that has multiple occurrences during the CTZ of τ_i . Once $a_{i,m} = a_{i,m-1}$, the demand for task τ_i is defined as $d'_i \equiv a_{i,m}$. If $\forall i: d'_i \leq d_i$, then the set of tasks $\{\tau'_n\}$ is feasible; otherwise, tasks whose demand $d'_i > d_i$ will miss their deadline during their CTZ. This step checks to see if the execution time that $\{\tau'_n\}$ and τ_i require during the CTZ of τ_i is less than or equal to the deadline of τ_i .

If the earlier assumptions for $\{\tau'_n\}$ and (d_i, C_i, T_i) hold, then the demand calculation implicitly performs admission control.

Figure 1 depicts a time line for a DMS algorithm, the demand calculation, and associated parameters for task τ_{ij} . It shows task arrivals, deadlines, and execution. The CI occurs when all tasks arrive simultaneously. The CTZ for each task extends from the CI until the task's deadline.

In the absence of a transient fault, we can determine a task's WCRT by summing all the execution times of tasks with higher priorities. The WCRT—or equivalently demand d'_i —can serve as a pseudodeadline to detect the occurrence of tasks that exceed their specified execution times or interarrival rates.

Providing fault tolerance

The ADM scheduling algorithm provides fault tolerance by detecting a transient surge and enacting a scheduling-mode change to VBS. If we assume the DMS algorithm performs scheduling tasks and $\{\tau'_n\}$ is fixed and ordered in descending order by deadlines, then we can use the result of Equation 1 to obtain a method for detecting transient surges. For each task $\tau_i \in \{\tau'_n\}$, let's assign pseudodeadline $d'_i = a_{i,m}$, where $a_{i,m}$ is the demand calculated in Equation 1. This abstraction creates a set of pseudotasks that fully utilizes the

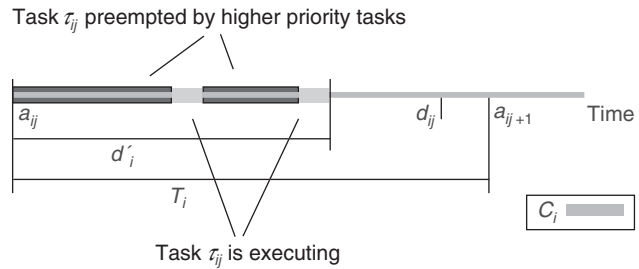


Figure 1. Time line demand calculation for task τ_{ij} at critical instant CI. The deadline-monotonic scheduling algorithm ensures that the task with the nearest deadline is always executed. Task parameters include execution time C_i , interarrival rate T_i , deadline d_{ij} , and demand d'_i . If the arrival time of $\tau_{ij} = a_{ij}$, then $T_i = a_{ij} - a_{ij-1}$. Worst-case response time (WCRT) for τ_i is equal to demand d'_i . CI is when all tasks have their WCRT—that is, when they arrive simultaneously. The critical time zone for each task is CI to $CI + d_i$. Normally, τ_i must complete execution by d'_i ; thus d'_i can serve as a pseudodeadline whose violation indicates a transient fault.

CPU in the sense that any transient surge during task τ_i 's CTZ will cause a pseudodeadline violation in either τ_i or lower-priority task τ_j , $i \leq j \leq n$. Examining each task's pseudodeadline will detect this surge during the next scheduling event. Additionally, any pseudodeadline violation indicates a transient surge in CPU loading. Note that d'_i is purely an abstraction used to detect surges in CPU loading. The ADM scheduling algorithm monitors pseudodeadlines during scheduling events to detect transient surges. This mechanism, alone or in conjunction with other methods, can provide fault detection.

Scheduling-mode changes using a sliding-windows approach

Detecting a transient fault raises the important issue of determining when to initiate a mode change from time- to value-based scheduling. The key issue is whether the resulting transient surge will cause the CPU to overload. The trade-off concerns the better throughput capability of slack-stealing time-based schedulers versus the optimal value capability of value-based schedulers. During a transient surge, if the effective CPU loading doesn't violate the DMS algorithm's scheduling conditions, all task deadlines will be met through DMS, and the ADM scheduler need not initiate a mode change. This slack-stealing approach leverages the DMS algorithm's optimal task throughput. Initiating a mode change

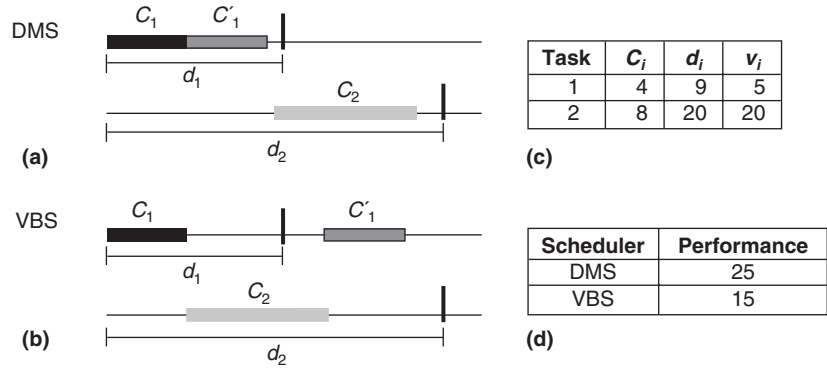


Figure 2. Transient surge not resulting in an overload. Task τ_1 reexecutes, causing a transient surge with effective demand less than the DMS bound. The shaded regions on the time axis labeled C_1 , C_2 , and C'_1 represent the execution time for tasks τ_1 and τ_2 and the reexecution time of τ_1 , respectively. Similarly, the task deadlines for τ_1 and τ_2 are given by d_1 and d_2 . The DMS algorithm schedules all tasks in a timely manner (a). VBS causes τ_1 to violate its deadline (b). These demand time lines have associated task parameters (c) and scheduler performance (d).

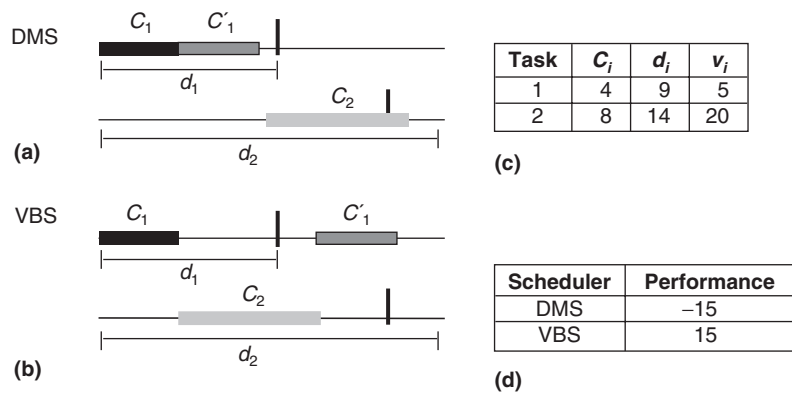


Figure 3. Transient fault resulting in an overload. Task τ_1 reexecutes, causing a transient surge with effective demand greater than the DMS bound. The DMS algorithm schedules a less valuable task on time, causing a critical task to be late (a). The VBS algorithm schedules a critical task on time, causing a less valuable task to be late (b). These demand time lines have associated task parameters (c) and scheduler performance (d).

to VBS might make some task(s) late. If the effective CPU demand does violate the scheduling conditions, then an overload has occurred and some tasks will be late. A VBS algorithm's ability to order tasks on the basis of their value provides the opportunity to sustain critical-task execution while suppressing less-important task execution. Continuing with the DMS algorithm could make critical tasks late.

Consider the following two cases. In Figure 2, a transient surge occurs without resulting in an overloaded system; in Figure 3 a transient surge does cause the system to over-

load. We use a naive VBS algorithm in which a task's value, v_p serves as its priority, and we evaluate system performance as the sum of the task performance measures. If a task completes on time, we add its value to the task's performance measure; otherwise, we subtract it. In Figure 2 the task parameters (C_p , d_p , and task value v_j) and scheduler performance appear in tables to the right. The DMS algorithm schedules all tasks in a timely manner by stealing slack. Initiating a mode change to VBS causes τ_1 to be late, degrading performance. In Figure 3, DMS causes the high-value task

to fail, while the less valuable task completes on time. Conversely, the VBS algorithm schedules the high-value task on time while delaying processing of the less valuable task.

Once the ADM scheduling algorithm detects a transient surge, a sliding window maintained for each task estimates how much available processing time remains before the task will be late. If the window size is greater than or equal to zero, ADM assumes no overload and uses a slack-stealing approach. If the window size is less than zero, an imminent overload is assumed, and ADM initiates a mode change to VBS. The lower window edge (LWE) represents the earliest time we consider a scheduling-mode change. Using the fault detection method described earlier, ADM initializes LWE for each task τ_i to its pseudo-deadline d'_{ij} . Once ADM detects a fault, the LWE for each task increments to the current time, decreasing the task's window size. The upper window edge (UWE) effectively represents the estimate of when a transient surge will result in an overload. This value can be bound on either side, creating a maximum and minimum window size. As a task executes, its UWE increments to reflect the remaining processing time. As a task executes, its LWE increments to reflect current time.

The minimum possible window size, $UWE = LWE$, assumes that any transient surge results in an overload. Using this estimate, the system performs well during a severe overload. However, if an overload doesn't occur, the early scheduling-mode change may cause some tasks to be unnecessarily late.

The maximum window size is the point at which a task is known to be late. We obtain this by setting $UWE = (d_i - C'_i)$, where C'_i represents the amount of processing time left for a task, and d_i is its deadline. Using this estimate, the system performs nominally when a transient surge doesn't result in an overload. However, during a severe overload, the failure to enact an early scheduling-mode change can cause critical tasks to be late. We can reasonably estimate when to enact a scheduling-mode change by setting each task's window length to its worst-case spare capacity.⁴ The spare capacity for a task is the amount of time that can be added to its execution time without causing lower priority tasks to be late. The worst-case spare capacity is the minimum spare capacity that will be available and occurs

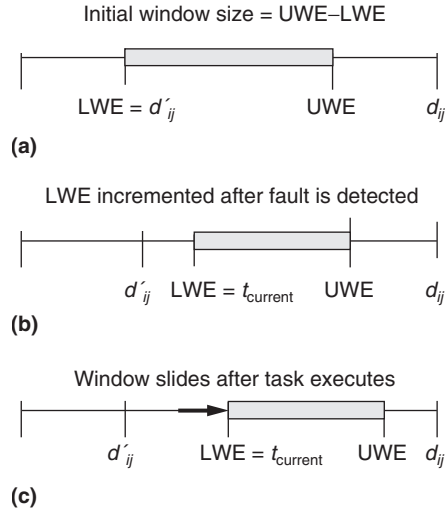


Figure 4. Sliding-window operation. The ADM scheduler initializes the window's lower and upper edges (a). After fault detection, the lower window edge (LWE) increments, reducing the window's size and the time available for task completion (b). After the task executes for a time, the upper window edge (UWE) increments to reflect the remaining processing time the task requires, and the LWE increments to reflect current time (c).

during a task's CTZ. Because we can determine this value beforehand, there would be no added runtime computational complexity. It's possible to either estimate the spare capacity or determine it exactly at runtime;⁶ however, the computational complexity is prohibitive during fault conditions.

Figure 4 shows how the sliding window operates. Using parameters that include workload and the expected occurrence of transient faults, we can tune window size to any value between the maximum and minimum.

Value functions and value-based scheduling

Each task has an assigned value function to capture the relative cost or benefit of its missing or meeting its time constraints.⁸ We evaluate a task's performance by examining the mean or minimum values assigned to it for each arrival. Evaluating cumulative values provides an effective way to measure scheduler performance during transient surges. Value functions also serve in scheduling tasks during overload conditions.

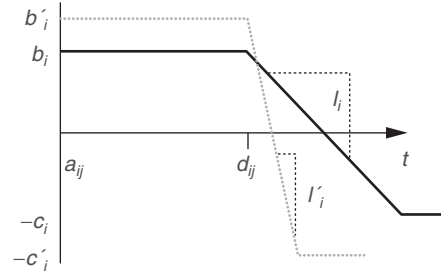


Figure 5. The value function of task τ_i , with arrival time and deadline given by a_{ij} and d_{ij} respectively. The solid line indicates maximum benefit b_i , before its deadline, maximum cost $-c_i$, after its deadline, and tardiness factor l_i . The dashed line depicts modification of value function parameters to b'_i , c'_i , and l'_i to reflect changes in system state.

The goal of VBS is to maximize system reward by always selecting the highest value task pending and if necessary suppressing lower-value tasks. Although less successful than the DMS algorithm with respect to number of tasks completed on time, this approach supports timely execution of critical tasks during overloads. Because of concerns for scheduling during transient faults, we adopt a relatively simple value function. The objective is to avoid computational complexity that would exacerbate fault conditions and cause further performance degradation.

We define value function $v_i(b_i, c_i, l_i)$, for each task τ_i , where b_i is the task's maximum benefit if it's on time, c_i is the maximum cost if it's late, and l_i represents the impact of tardiness for tasks with soft deadlines. Figure 5 depicts the time line for task τ_i . Before its deadline d_i , the task has maximum value b_i . After its deadline, that value begins to decline by l_i to its maximum cost $-c_i$. Thus

$$\text{if } t \leq d_i, \text{ then } v_i(b_i, c_i, l_i) = b_i \quad (2)$$

and τ_i is on time. Otherwise,

$$v_i(b_i, c_i, l_i) = \min \left(b_i, b_i \left(1 - \frac{\Delta t}{l_i} \right) \right),$$

$$\text{if } b_i \left(1 - \frac{\Delta t}{l_i} \right) \geq 0$$

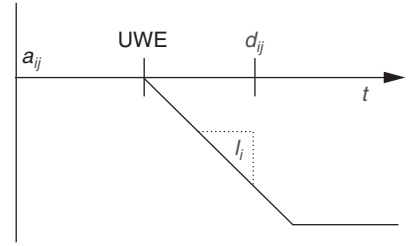


Figure 6. Priority assignment using value functions after window size is less than zero.

$$v_i(b_i, c_i, l_i) = \max \left(-c_i, c_i \left(1 - \frac{\Delta t}{l_i} \right) \right),$$

$$\text{if } c_i \left(1 - \frac{\Delta t}{l_i} \right) < 0$$

where $\Delta t = t - d_i$ is the amount of time that τ_i is late.

A positive value for the term $(1 - \Delta t/l_i)$ indicates that the task still has some benefit, while a negative value indicates the task will incur a cost (penalty). When this term goes to zero, the task has neither benefit nor cost. The choice of l_i determines how fast the benefit declines and the cost grows once the task is late. Maximum cost $c_i = \infty$ represents system failure. A value of $l_i = 0$ represents a step function decline in value for tasks with firm or hard deadlines.

The ADM scheduler can modify value function parameters to reflect changes in system state. Figure 5 depicts parameter modification for benefit, cost, and tardiness to b'_i , c'_i , and l'_i to reflect a change in system state resulting from mode change, late task, and so on.

During transient overloads, the ADM scheduling algorithm uses value functions to order task execution. This ensures that the processor always executes the most valuable pending task. Once a task's window size reaches zero, the ADM scheduler assigns its priority using the value function. Thus, for task τ_i ,

$$\text{if } \text{LWE} < \text{UWE}, \text{ then priority} = d_i \quad (3)$$

and the scheduler uses DMS scheduling. Otherwise,

Table 1. Task parameters for the UGV task model are task processing time C_i , deadline d_i , and interarrival rate T_i . Value function parameters are maximum benefit b_i , maximum cost c_i , and tardiness impact l_i .

Task no.	Task	C_i (ms)	d_i (ms)	T_i (ms)	b_i	c_i	l_i	Arrivals
1	Vehicle braking	3	10	10,000	100	100	0	Sporadic
2	Hazard response, local path planning	23	200	10,000	80	80	d_2	Sporadic
3	Sensor data fusion	10	80	500	60	60	d_3	Periodic
4	Steering control loop	4	20	20	40	40	0	Periodic
5	Steering set point	3	60	400	40	40	$0.5d_5$	Sporadic
6	Velocity control loop	4	20	20	30	30	0	Periodic
7	Velocity set point	3	60	400	30	30	$0.5d_7$	Sporadic
8	System management	5	50	100	15	15	d_8	Periodic
9	CPU status	2	100	1,000	5	5	T_9	Periodic
10	Electrical-system status	2	100	1,000	5	5	T_{10}	Periodic
11	Power train status	2	100	1,000	5	5	T_{11}	Periodic

$$\text{priority} = \max \left(-c_i, c_i \left(1 - \frac{\Delta t}{l_i} \right) \right)$$

and the scheduler uses VBS scheduling. The parameter $\Delta t' = t - UWE$ is the elapsed time since the task's window closed. In this situation, a task's priority is negative and will continue to decrease until it completes execution or reaches the maximum cost. Lower values have higher priorities, thereby maintaining consistency with the DMS algorithm. Figure 6 shows the priority assignment for a task once the window size reaches zero.

Performance evaluation

We use a simulation-based performance evaluation to measure the ADM scheduling algorithm's ability to sustain critical-task execution during transient surges.

Task model and simulation description

We derived the task model from the model for the central processor of a UGV under development for the US Army. This vehicle is designed to traverse hazardous terrain while collecting audio and image data. Control works through a preprogrammed path or remotely over a wireless link. Encountering unforeseen hazards triggers a semiautonomous capability for making local path decisions. In Table 1, 11 tasks appear in order of decreasing importance. Tasks 1 through 7 concern the vehicle's primary functions and thus have higher values. Tasks 8 through 11, concerned

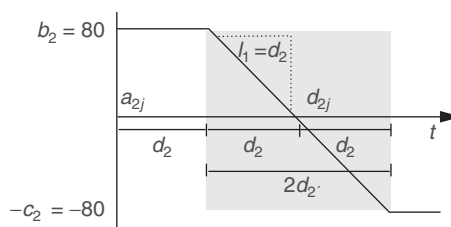


Figure 7. Value function for task τ_{2j} , the j th instance of task τ_2 , showing its decline from b_2 to $-c_2$ once it's late.

with monitoring and managing the vehicle's subsystems, have lower values. The various columns in the table give task and value function parameters. We derived the value function parameters from interviews with soldiers to help in assessing performance and ordering tasks during overloads. The last column in Table 1 indicates whether the task interarrivals are periodic or sporadic.

Figure 7 shows an example of the value assignment for task τ_2 . If the task finishes on time, its value would be assigned benefit $b_2 = 80$. If it completes late, it would be assigned a value linearly distributed in the range from b_2 to less than $-c_2$. Since $b_2 = d_2$, according to Equation 3 the value of the j th instance of τ_2 would equal 0 at time $t = a_{2j} + 2d_{2j}$, and at time $t = a_{2j} + 3d_{2j}$ it would reach $-c_2$.

We evaluated the ADM scheduling algorithm's performance using three different window sizes: minimum, maximum, and spare

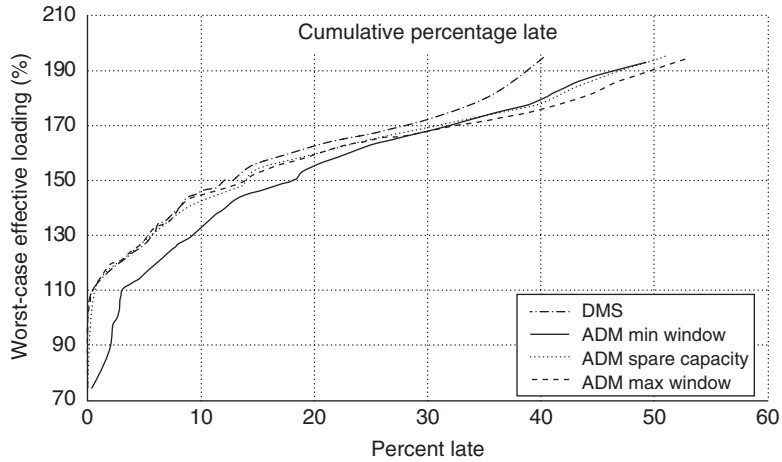


Figure 8. Cumulative percentage of late task completions for worst-case loading ranging from nominal to 200 percent.

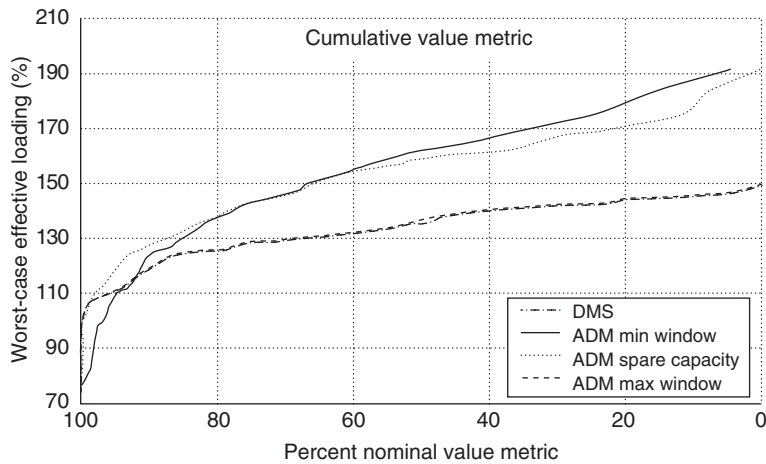


Figure 9. Cumulative percentage of value metric for worst-case loading ranging from nominal to 200 percent.

capacity. For reference, we also simulated the DMS algorithm's performance without scheduling-mode changes. We refer to the different sets of results as ADM min window, ADM max window, ADM spare capacity, and DMS.

Each simulation ran for 300 seconds of simulated time, during which we inserted three types of transient faults: tasks that needed to be reexecuted, tasks that exceeded their WCET, and sporadic tasks that exceeded their WCIT. Effective CPU demand includes normal task loading and the effects of transient faults. We determine this by adding any extra computation time or task arrivals caused by transient faults to the demand calculation of Equation

1. The worst-case effective loading is the worst-case ratio between any task's effective demand and its deadline. The worst-case effective loading between different simulation runs ranged from 67 percent without faults to worst-case effective demand of 200 percent. The interval between consecutive simulation runs was approximately 8 percent (27 simulation runs).

Simulation results and performance analysis

The simulation data consists of cumulative results for task set $\{\tau_i\}$ with respect to increasing levels of CPU loading, and individual task performance for selected levels of CPU loading. Figure 8 shows cumulative performance data for the total percentage of late tasks. Figure 9 shows the overall value metric compared with increasing CPU loading. In Figure 8, DMS has the lowest percentage of late task completions, reflecting its ability to sustain overall task throughput. During transient faults that don't overload the system, the percentage of late tasks is comparable for DMS, ADM max window, and ADM spare capacity. ADM min window has the highest percentage of late task completions when CPU loading is under 170 percent. As CPU loading exceeds 140 percent, the percentage of late tasks for ADM max window and ADM spare capacity exceeds that of DMS and begins to converge to that of ADM min window.

These results verify our expectations regarding the behavior of VBS during transient surges. The optimistic approach using ADM max window parallels that of DMS until loading exceeds 140 percent. The relatively high percentage of late tasks for ADM min window verifies its tendency to make the system overreact in the case of transient surges that don't cause overload. ADM spare capacity doesn't cause an overreaction to transient surges, indicating that selection of spare capacity as the window size wasn't overly pessimistic. The divergence of DMS from all ADM variants as loading increases reflects the tendency of VBS to trail DMS in on-time completion ratios.

The cumulative value metric, shown in Figure 9, provides more insight into how the schedulers performed with respect to task values. When a transient surge doesn't result in an overload, DMS, ADM max window, and ADM spare capacity achieve maximum value. This indicates that neither of these approaches caus-

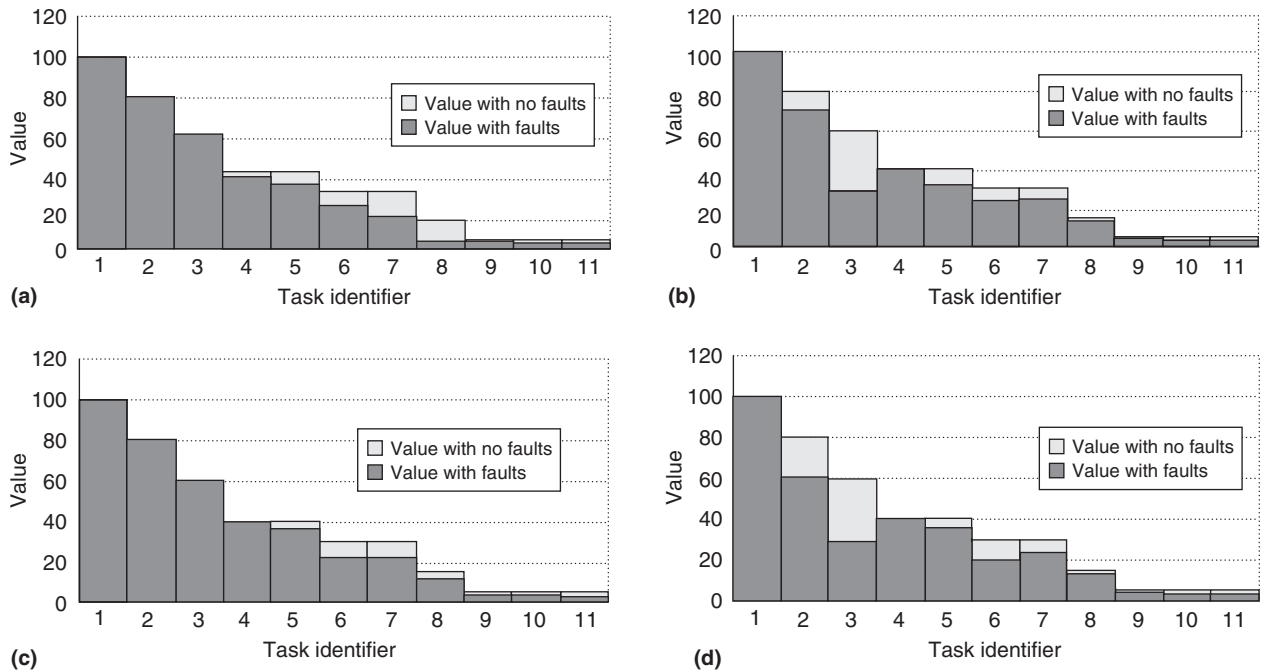


Figure 10. Value matching at 125 percent worst-case effective loading for ADM min window (a), ADM max window (b), ADM spare capacity (c), and DMS (d) algorithms.

es overreaction to a transient surge. ADM min window causes overreaction by assuming an overload condition and triggering a mode change to VBS. In this case, late task completion results in a reduced cumulative value metric.

Once the CPU is overloaded, DMS's cumulative value decreases at the highest rate, reflecting its inability to consider a task's value. The performance of ADM max window surpasses that of DMS up to about 128 percent of loading. Beyond this point, ADM max window begins to cause an underreaction by waiting too long to initiate a mode change to VBS. Its cumulative value falls off rapidly, until its performance parallels that of DMS. Studies of individual task performance show a marginal benefit advantage for ADM max window over DMS for high-value tasks and a marginal cost disadvantage for medium- and lower-value tasks.

ADM min window demonstrates the benefit of causing an overreaction when a severe overload occurs. This approach attains the highest value metric at CPU loads in excess of 160 percent; however, it is unnecessarily pessimistic during moderate transient surges. ADM spare capacity achieves the highest value metric for CPU loads up to 160 percent and

thereafter nearly parallels the pessimistic ADM min window's performance. This approach doesn't cause an overreaction during moderate transient surges, nor does it cause an underreaction during severe overloads.

Simulation results for individual tasks provide a more detailed look into the schedulers' behavior during transient surges. These results also help us determine how individual tasks performed. Individual task data regarding value matching for worst-case loading equal to 125 percent appears in Figure 10. Value matching represents the mapping of ideal values to actual recorded values for individual tasks at a given CPU loading level. Data collected at other levels of system loading was consistent with these results.

Figure 10 shows tasks ordered in descending level of value from 1 to 11. For DMS and ADM max window, high-value tasks 2 and 3 have marginal benefit. Since these tasks concern detecting hazards and obstacles and fusing sensor data to build terrain knowledge, the vehicle may not be able to detect and respond to hazards. For ADM max window, poor high-value-task performance results from waiting too long to initiate a scheduling-mode

change. Value matching for tasks concerned with vehicle steering and speed, tasks 4 through 7, indicates that steering and acceleration would also have degraded.

Value matching for ADM min window and ADM spare capacity indicates that tasks 2 and 3 performed correctly. ADM min window's tendency to cause an overreaction appears in the degradation of steering capability, which was deemed more important than velocity control. System management (task 8) also performed better for ADM spare capacity.

Future efforts will focus on integrating the ADM approach into interUGV communications for scheduling message transmissions on a high-loss channel. MICRO

ACKNOWLEDGMENT

This work was partially supported by the US Army Vetronics Institute.

References

1. G. Butazzo, *Hard Real-Time Computer Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, Norwell, Mass., 1997.
2. R.K. Iyer, D. Rossetti, and M. Hsueh, "Measurement and Modeling of Computer Reliability as Affected by System Activity," *ACM Trans. Computer Systems*, vol. 4, no. 3, Aug. 1986, pp. 214-237.
3. D.P. Siewiorek, "A Case Study of C.mmp, Cm* and C.vmp: Part 1—Experiences with Fault Tolerant Multiprocessor Systems," *Proc. IEEE*, vol. 66, no. 10, Oct. 1978, pp. 1,178-1,199.
4. M. Klien et al., *A Practitioners Handbook for Real-time Analysis*, Kluwer Academic Publishers, Norwell, Mass., 1994.
5. S. Goshe, R. Melhem, and D. Mosse, "Fault Tolerant Rate Monotonic Scheduling," *J. Real Time Systems*, vol. 15, no. 2, Sept. 1998, pp. 149-181.
6. S. Ramos-Thuel and J. Stosnider, "Scheduling Fault Recovery Operations for Time-Critical Applications," *Proc. Fourth IFIP Conf. Dependable Computing for Critical Applications*, Elsevier Science, New York, 1995, pp. 161-179.
7. B. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading, Mass., 1989.
8. E.D. Jenson, C.D. Locke, and H. Tokuda, "A Time-Value Driven Scheduling Model for Real-Time Operating Systems," *Proc. Symp. Real-Time Systems*, IEEE CS Press, Los Alamitos, Calif., 1985, pp. 112-122.
9. A. Burns and D. Prasad, "Value Based Scheduling of Flexible Real-Time Systems for Intelligent Autonomous Vehicle Control," *Proc. Third IFAC Symp. Intelligent Vehicles*, Elsevier Science, New York, Mar. 1998, pp. 127-132.
10. A. Burns et al., "Probabilistic Scheduling Guarantees for Fault-Tolerant Real-Time Systems," *Proc. Seventh Int'l Working Conf. Dependable Computing for Critical Applications (DCCA 99)*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 361-379.
11. G. Buttazzo, M. Spuri, and F. Sensini, "Value vs. Deadline Scheduling in Overload Conditions," *Proc. IEEE Real-Time Systems Symp. (RTSS 95)*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 90-100.
12. Mejia-Alvarez, R. Melhem, and D. Mosse, "An Incremental Approach to Scheduling during Overloads in Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.*, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 71-82.

Paul Richardson is an assistant professor in the Department of Electrical and Computer Engineering at the University of Michigan-Dearborn. His research interests include embedded real-time systems, in-vehicle networks, and mobile networks. Richardson has a BSE in computer engineering, an MSE in computer and electrical engineering, and a PhD in systems engineering, all from Oakland University. He is a member of the IEEE and ACM.

Larry Sieh is a research associate at the US Army Research Center in Warren, Michigan. His research interests include wireless combat networks, multimedia networks, network simulation, combat vehicle simulation, and software engineering. Sieh has a BSE in computer engineering and MS degrees in electrical engineering, systems engineering, and computer science, all from Oakland University. He is a PhD candidate in software engineering at the Naval Postgraduate School and a member of the IEEE.

Ali M. Elkateeb is an associate professor in the Department of Electrical and Computer

Engineering at the University of Michigan-Dearborn. His research interests include embedded RISC architectures, reconfigurable computing, and high-speed networks. He holds BSc, MSc, and PhD degrees in computer science. He is a member of the IEEE.

Direct questions or comments about this article to Paul Richardson, Dept. of Electrical and Computer Engineering, University of Michigan-Dearborn, 4901 Evergreen Rd., Dearborn, MI 48128-1491; paul_richardson@umich.edu.

For further information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.