Dynamic Window-Constrained Scheduling

Richard West

Addendum

A. DWCS Characterization

Consider a task, τ_i, defined by a 3-tuple $(C_i, T_i, x_i/y_i)$, where C_i is the service time requirement every request period, T_i, and x_i/y_i is the task's window-constraint. If τ_i represents a periodic task, C_i can be thought of as the *periodic* service time requirement, and T_i defines the interval between consecutive instances of τ_i being ready for service. If τ_i represents an aperiodic task, C_i can be thought of as a service *quantum* and the nth quantum should ideally be serviced in the nth request period. In essense, the end of the nth request period defines the nth deadline for servicing τ_i. If no deadlines are missed after n request periods, τ_i will receive nC_i units of service time.

Under the definition of DWCS, the window-constraint x_i/y_i defines the acceptable maximum number of missed deadlines, x_i, every window of y_i consecutive deadlines. This means τ_i must be serviced for a minimum of $(y_i - x_i)C_i$ units of time every window of $y_i T_i$ time units.

Mok and Wang extended our original work by showing that the general window-constrained problem is NP-hard for arbitrary length tasks [1]. Notwithstanding, DWCS guarantees that no more than x_i deadlines are missed out of y_i deadlines for n tasks, if $U = \sum_{i=1}^{n} \frac{(1-x_i/y_i)C_i}{T_i} \leq 1.0$, given $1 \leq i \leq n$, $C_i = K$ and $T_i = qK$; where $q \in Z^+$ [1], K is a constant, and U is the minimum utilization factor for a feasible schedule.[2]

This implies a feasible schedule is possible even when the system is 100% utilized, given (a) all tasks have constant length (and, hence service time) and, (b) all request periods are the same and are multiples of the constant service time. Although this sounds restrictive, it offers the ability for a DWCS scheduler to proportionally share service amongst a task set, τ. Moreover, each task, $\tau_i \in \tau$, is guaranteed a minimum share of resources over a specific window of time, independent of the service provided to other tasks. This constrasts with fair queueing algorithms that (a) attempt to share resources over the smallest window of time possible (thereby approximating the fluid-flow model) and, (b) do not provide isolation guarantees. In the latter case, the addition of a task to the system can affect the service provided to all other tasks, since proportional sharing is provided in a relative manner. For example, weighted fair queueing uses a weight, w_i for each task, τ_i, such that τ_i receives (approximately) $\frac{w_i}{\sum_{j=1}^{n} w_j}$ fraction of resources over a given window of time.

Having defined the general DWCS task set, we now show the utilization bound for a specific task set, τ, in which each task, $\tau_i \in \tau$, is characterized by the 3-tuple $(C_i = K, T_i = qK, x_i/y_i)$. In what follows, we consider the maximum number of tasks, n_{max}, that can guarantee a feasible schedule. It can be shown that for all smaller task sets (where $n < n_{max}$), a feasible schedule is always guaranteed if one is guaranteed for n_{max} tasks.

Lemma 1: Consider a set of n tasks, $\tau = \{\tau_1, \cdots, \tau_n\}$, where $\tau_i \in \tau$ is defined by the 3-tuple $(C_i = K, T_i = qK, x_i/y_i)$. Without loss of generality, we can assume $K = 1$. If the utilization factor, $U = \sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_i} \leq 1.0$, then $x_i = y_i - 1$ maximizes n.

Proof: For all non-trivial situations, $n > q$, otherwise we can always find a unit-length slot in any fixed interval of size q to service each task at least once. Now, for any window-constraint, x_i/y_i, we can assume $x_i < y_i$, since if $x_i = y_i$ then no deadlines need to be met for the corresponding task, τ_i. Consequently, for arbitrary τ_i, $y_i - x_i \geq 1$.

Therefore, if we let $y_k = max(y_1, y_2, \cdots, y_n)$ it must be that $n \leq qy_k$, since:

$$\frac{n}{qy_k} = \sum_{i=1}^{n} \frac{1}{qy_k} \leq \sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_k} \leq \sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_i} \leq 1$$

$$\Rightarrow n \leq qy_k$$

If all window-constraints are equal, for each and every task, we have the following:

$$\sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_i} \leq 1 \Rightarrow \frac{n(y_i - x_i)}{qy_i} \leq 1$$

[1] Z^+ is the set of positive integers.

[2] In the original RTSS paper, we incorrectly stated $T_i = q_i K$; where $q_i \in Z^+$. The utilization bound proved in this addendum and outlined in the main paper [2] holds for fixed q.

$$\Rightarrow n \leq \frac{qy_i}{y_i - x_i} \leq qy_i$$

if $x_i = y_i - 1$, then $\frac{qy_i}{y_i - x_i} = qy_i$, and n is maximized. $\Box$

From the above Lemma, we now consider the conditions for a feasible schedule, when each task, τ_i, in a set of n tasks, is defined by the 3-tuple $(C_i = 1, T_i = q, x_i/y_i)$. We begin by defining the task *hyper-period* to be $lcm(qy_1, qy_2, \cdots, qy_n)$, and the current window-constraint of τ_i to be x'_i/y'_i. The following theorem can now be stated:

Theorem 1: In each *non-overlapping* window of size q in the hyper-period, there cannot be more than q tasks out of n with *current* window-constraint $\frac{0}{y'_i}$ at any time, when $U = \sum_{i=1}^{n} \frac{y_i - x_i}{qy_i} \leq 1.0$.

Proof: When $n \leq q$, it is clear there are never more than q tasks with current window-constraint $\frac{0}{y'_i}$. For all non-trivial values of n, it must be that $q < n \leq qy_k$, given that $y_k = max(y_1, y_2, \cdots, y_n)$. From Lemma 1, if $y_1 = y_2 = \cdots = y_n$, and $x_i = y_i - 1$, $\forall i$, then $n \leq qy_i$. It can be shown that all lower values of n will yield a feasible schedule if one exists for largest n.

Now, consider a set τ comprising one task, τ_j, that has window-constraint, x_j/y_j, and $n-1$ other tasks, each having window constraint, x_i/y_i. From Lemma 1, it follows that if $x_j/y_j < x_i/y_i$ then $n < qy_i$. In this case, n is maximized if $x_j = y_j - 1$, $x_j + 1 = x_i$, and $x_i = y_i - 1$. Hence, $x_j < x_i$, $y_j < y_i$ and $n < q(x_i + 1)$.

The set τ is scheduled in the various non-overlapping intervals of the hyper-period, resulting in changes to window-constraints, as shown below:

1. Time interval $[0, q)$: Task τ_j is scheduled first since $x_j/y_j < x_i/y_i$. The current window-constraints of each task are adjusted over the time interval (shown above the arrows) as follows:

$\frac{x_j}{y_j} \xrightarrow{q} \frac{x_j}{y_j - 1}$ (one task, τ_j, serviced on time)
$\frac{x_i}{y_i} \xrightarrow{q} \frac{x_i}{y_i - 1}$ ($q-1$ tasks serviced on time)
$\frac{x_i}{y_i} \xrightarrow{q} \frac{x_i - 1}{y_i - 1}$ ($n-q$ tasks not serviced on time)

2. Time interval $[q, q(x_j + 1))$: It can be shown that $n > q(x_j + 1)$ when n is maximized. Furthermore, in this scenario, DWCS will schedule qx_j tasks with the smallest current window-constraints, updated every q time units. As a result, window-constraints now change as follows:

$\frac{x_j}{y_j - 1} \xrightarrow{qx_j} \frac{0}{y_j - 1 - x_j}$ (one task, τ_j, not serviced)
$\frac{x_i}{y_i - 1} \xrightarrow{qx_j} \frac{x_i - x_j}{y_i - 1 - x_j}$ ($q-1$ tasks not serviced on time)
$\frac{x_i - 1}{y_i - 1} \xrightarrow{qx_j} \frac{x_i - 1 - x_j}{y_i - 1 - x_j}$ ($n - q - qx_j$ tasks not serviced)
$\frac{x_i - 1}{y_i - 1} \xrightarrow{qx_j} \frac{x_i - x_j}{y_i - 1 - x_j}$ (qx_j tasks serviced on time)

At this point consider the $n - q - qx_j$ tasks in state $\frac{x_i - 1 - x_j}{y_i - 1 - y_j}$ after time $q(x_j + 1)$. We know in the worst case, $x_j + 1 = x_i$ to maximize n, so

$$n - q - qx_j = n - q(x_j + 1) = n - qx_i$$

We also know $n < q(x_i + 1)$, so

$$n - qx_i < q(x_i + 1) - qx_i = q$$

Consequently, at the time $q(x_j + 1)$, less than q tasks other than τ_j are in state $\frac{0}{y'_i}$. Even though τ_j is in state $\frac{0}{y'_j}$, we can never have more than q tasks with zero-valued numerators as part of their current window-constraints. We know that, by maximizing n, we have

$$x_j + 1 = x_i, x_j + 1 = y_j \Rightarrow y_j = x_i$$

Therefore, at the time $q(x_j + 1)$, all current window-constraints can be derived from their original window-constraints, as follows:

$\frac{x_j}{y_j} \xrightarrow{q(x_j+1)} \frac{0}{0}$ (one task, τ_j, served once; reset $\frac{0}{0}$ to $\frac{x_j}{y_j}$)
$\frac{x_i}{y_i} \xrightarrow{q(x_j+1)} \frac{0}{1}$ ($n - qx_i$ tasks never serviced on time)
$\frac{x_i}{y_i} \xrightarrow{q(x_j+1)} \frac{1}{1}$ ($q-1$ tasks serviced once on time)
$\frac{x_i}{y_i} \xrightarrow{q(x_j+1)} \frac{1}{1}$ (qx_j tasks serviced once on time)

3. Time interval $[q(x_j + 1), q(x_j + 2))$: At the end of this interval of size q, the window-constraints change from their original values, as follows:

$\frac{x_j}{y_j} \xrightarrow{q(x_j+2)} \frac{x_j}{y_j - 1}$ (one task, τ_j, serviced twice overall)
$\frac{x_i}{y_i} \xrightarrow{q(x_j+2)} \frac{x_i}{y_i}$ ($n-1$ tasks serviced at least once;
reset window-constraint)

4. Time interval $[q(x_j + 2), 2q(x_j + 2))$: At the end of this interval of size $q(x_j + 2)$, the window-constraints change from their original values, as follows:

$\frac{x_j}{y_j} \xrightarrow{2q(x_j+2)} \frac{x_j}{y_j - 2}$ (one task, τ_j)
$\frac{x_i}{y_i} \xrightarrow{2q(x_j+2)} \frac{x_i}{y_i}$ ($n-1$ tasks; reset window-constraint)

Over the entire period $[0, y_j q(x_j + 2)]$, the window-constraints change as follows:

$\frac{x_j}{y_j} \xrightarrow{y_j q(x_j+2)} \frac{x_j}{y_j}$ (one task, τ_j)
$\frac{x_i}{y_i} \xrightarrow{y_j q(x_j+2)} \frac{x_i}{y_i}$ ($n-1$ tasks)

At this point, every task has been served at least once and no more than q tasks ever have zero-valued current window-constraints in any given non-overlapping interval of size q. Observe that the hyper-period is $lcm(qy_1, qy_2, \cdots, qy_n)$ which, in this case is qy_iy_j. Since $x_j + 2 = y_i$, $y_jq(x_j + 2) = qy_iy_j$, and we have completed the hyper-period. All tasks have reset their window-constraints to their original values, so we have a feasible schedule. $\Box$

References

[1] A. K. Mok and W. Wang. Window-constrained real-time periodic task scheduling. In *Proceedings of the 22st IEEE Real-Time Systems Symposium*, 2001.

[2] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, December 2000.