Computer Science

A Virtual Deadline Scheduler for Window-Constrained Service Guarantees

Yuting Zhang, Richard West, Xin Qi

Boston University

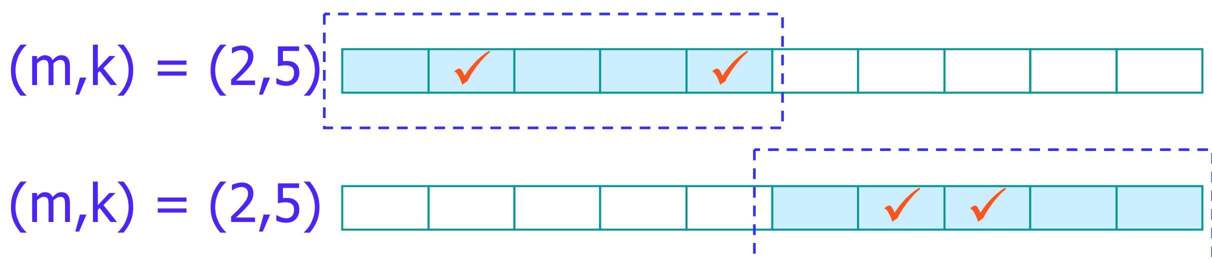
RTSS 2004

Motivating Applications

- Multimedia & weakly-hard real-time systems:
  - Not every deadline needs to be met
    - Impossible to meet every deadline in overload case
    - Can tolerate some deadlines being late or missed without degrading service too much
      - Loss- or window-constraints on service

Window-Constrained Scheduling

- Guarantee a fraction of service over a fixed window of job instances
  - (m,k) window-constraint:
    - At least m out of every k job instances meet their deadlines
  - Example:



Window-Constrained Service

- Provides independent service guarantees
  - Each job gets a minimum fixed share of service without being affected by others
- Is suitable for overload cases
  - Strategically skip some deadlines
    - **Min utilization** may still be 100% for feasible schedule
- Has bounded delay and jitter
  - Within a given window

DWCS

- Dynamic Window-Constrained Scheduling
  - Consider periodic jobs with deadlines at the ends of their request periods
  - Separately considers deadlines and window-constraints to order jobs
  - Can guarantee the service with unit process time, constant request period up to 100% utilization
  - May fail to provide service guarantees with different periods, even when the utilization is fairly low
  - Problem: How to improve service guarantees when periods (or deadlines) are different?

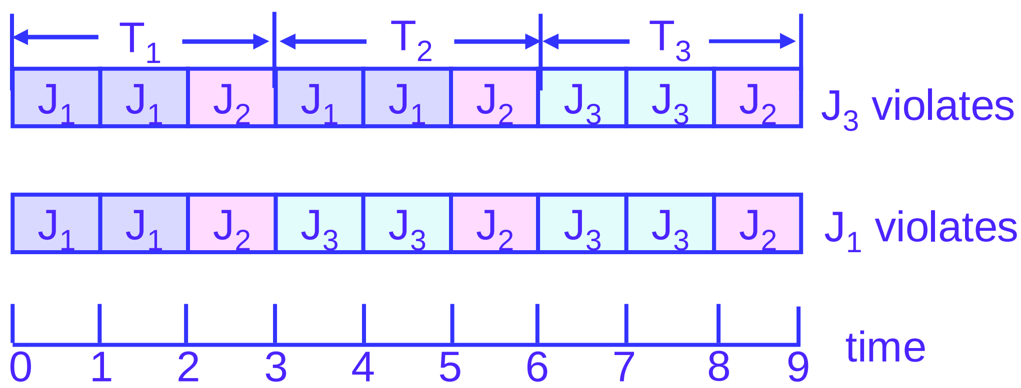
Talk Outline

- ✓ Motivation
- ✍ The relaxed model
- ✍ VDS algorithm
- 💻 Simulations
- 💻 Experiments
- ☺ Conclusions

Feasibility Condition

- Utilization: $U = \Sigma(C_i/T_i)$
- Minimum Utilization: $U_{min} = \Sigma(m_iC_i/k_iT_i)$
- Feasible iff $U_{min} \le 1$ and service time, $C_i = \Delta$
- NP-hard problem for arbitrary C_i and T_i *[Mok &Wang]*
  - Example: no feasible schedule even if $U_{min} \le 1$

Job	(C,T,m,k)
J_1	(2,3,2,3)
J_2	(1,3,1,3)
J_3	(2,3,2,3)
U_{min}	1

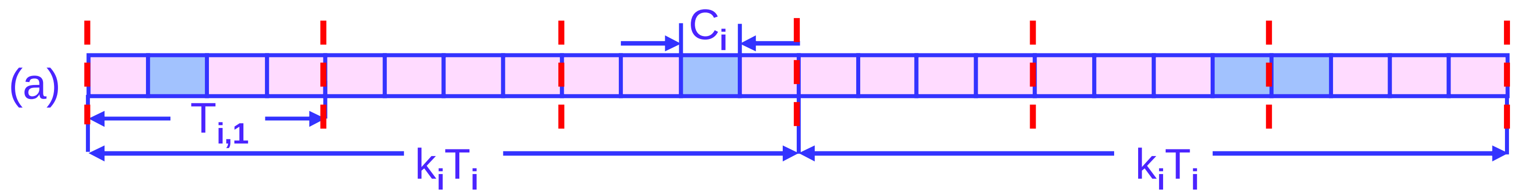


The Relaxed Model

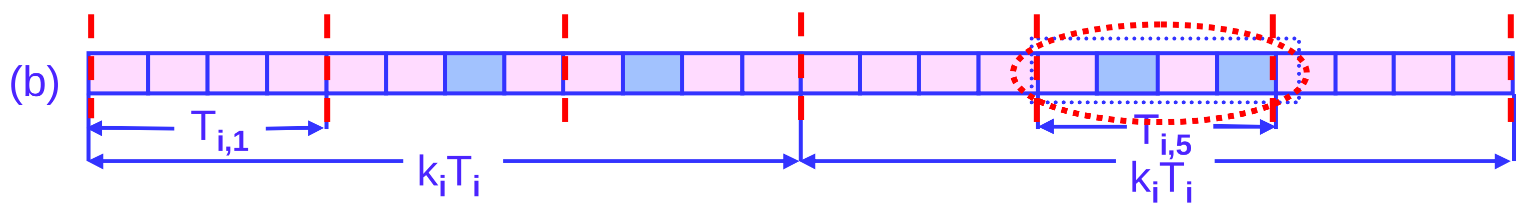
- At least m_i job instances are *served* in every window of k_i requests
  - allowing multiple requests that have arrived in the current window to be serviced in the same period
- The proportional share of resources allocated to a job in a window of size k_iT_i is still m_iC_i/k_iT_i, but…
- Job instances can be buffered & scheduled after their deadlines with the relaxed model

Original versus Relaxed Model

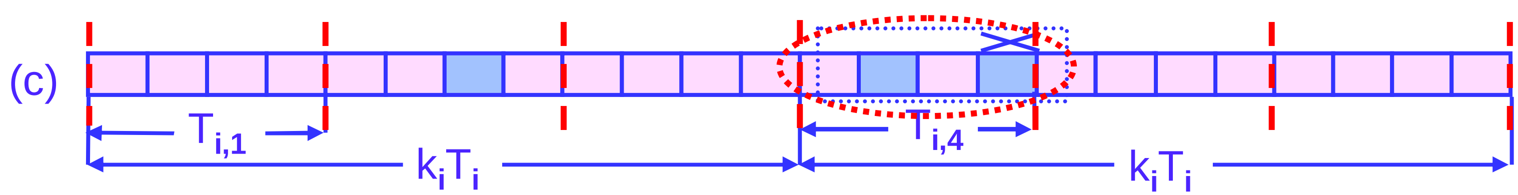
Job J_i: $C_i=1$, $T_i=4$, $m_i=2$, $k_i=3$



--- The original model



--- The relaxed model



--- The relaxed model

VDS Algorithm

- Virtual Deadline Scheduling (VDS) algorithm
  - Works with both relaxed and original window-constrained scheduling models
  - Job with lowest virtual deadline has highest priority
  - Question: How do we calculate virtual deadlines?

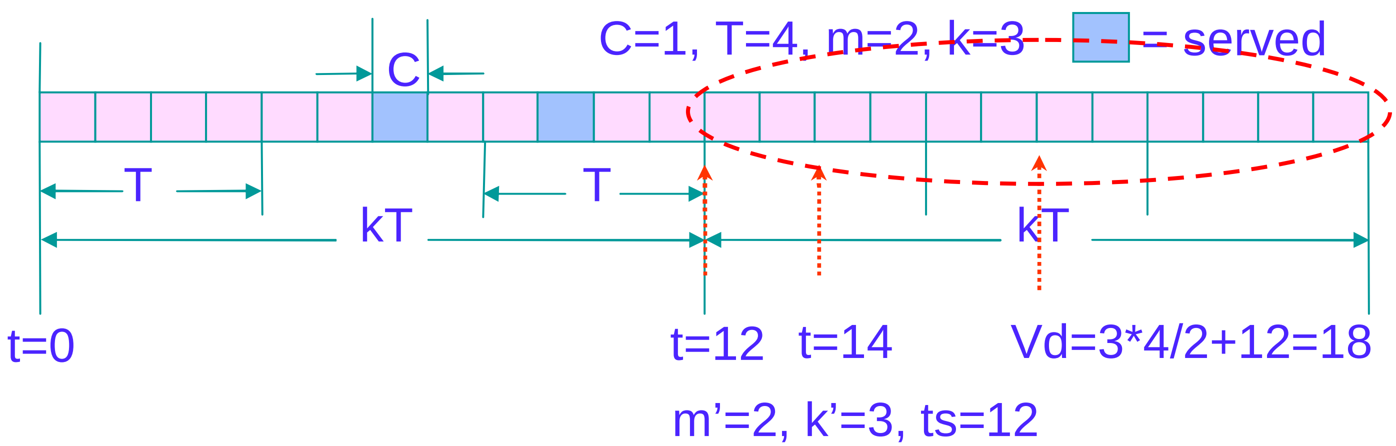
Virtual Deadline

Computer Science

- Function of *request period* and *window constraint*
- If current constraint is (m’,k’), it makes sense to service the next job instance in (k’*T)/m’ time
  - This is for proportional fairness
- Virtual deadline:

$$Vd_i(t) = k_i'T_i/m_i' + ts_i(t)$$

$ts_i(t)$: start of current request period at time unit t



Service Constraint Updates

After serving job J_i with the lowest virtual deadline:

$C_i' = C_i' - \Delta;$

if ($C_i' == 0$) m_i'--;

For every job J_j:

if (($Vd_j <= \Delta + t$) && ($j != I$) && ($C_i' > 0$))

Tag J_j with a violation

if (a new job instance arrives) {

k_j'--; $C_j' = Cj$;

if ($k_j' == 0$) { $m_j' = m_j$; $k_j' = k_j$;} }

if ($m_j' > 0$) update Vd_j

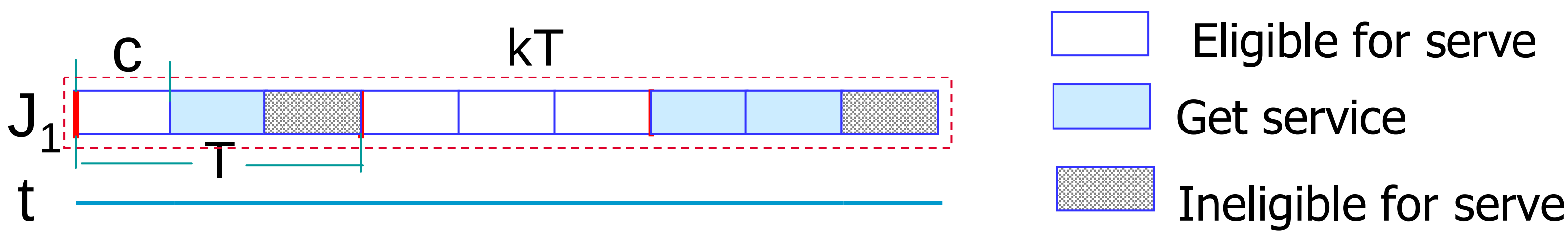
//only for relaxed model

if ((($k_j - k_j'$) >= ($m_j - m_j'$)) && ($C_i' == 0$))

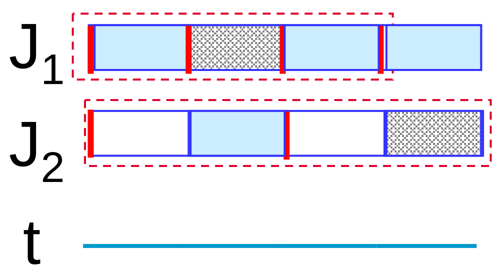
$C_j' = C_j$;

Scheduling Eligibility

- Schedule eligible job with the lowest virtual deadline
  - Eligibility in every request period (C' >0)



  - Eligibility in every request window (m'>0)



(C,T,m,k)= (1,1,1,3)

(C,T,m,k)= (1,2,1,2)

VDS vs. EDF, DWCS

$\underline{Vd_i(t) = k_i'T_i/m_i' + ts_i(t)}$

- If every k_i is a multiple of m_i, VDS reduces to EDF
  - Where C_i = process time, k_iT_i/m_i = request period
- If all T_is are constant, VDS reduces to DWCS
  - $Vd \propto k'/m'$
- In these cases, VDS can guarantee 100% utilization for the same situations as EDF and DWCS
- When T_i, m_i, k_i are arbitrary, VDS more accurately captures information about a job's combined urgency and importance

Example

Job	(C,T,m,k)
J_1	(1,1,2,9)
J_2	(1,3,1,1)
J_3	(1,3,1,1)
U_{min}	8/9

J_3 violates (EDF, at time 3)

J_3 violates (DWCS, at time 3)

	0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8	8–9
EDF	J_1	J_1	J_2	J_2	J_3	J_1	J_2	J_3	J_1
DWCS	J_1	J_1	J_2	J_2	J_3	J_1	J_2	J_3	J_1
VDS	J_2	J_3	J_1	J_2	J_3	J_1	J_2	J_3	J_1

time

	0	1	2	3	4	5	6	7	8	9	
Vd_1	9/2	5	11/2	9	9	9	X	X	9	27/2	…
Vd_2	3	X	X	6	X	X	9	X	X	12	…
Vd_3	3	3	X	6	6	X	9	9	X	12	…

EWDF

Computer Science

- Eligibility-based Window Deadline First
  - Target for the relaxed model
  - A variant of EDF with (service time=m_iC_i, period=k_iT_i)
    - Common window deadline for all job instances in the same window
  - Eligibility test is the same as VDS

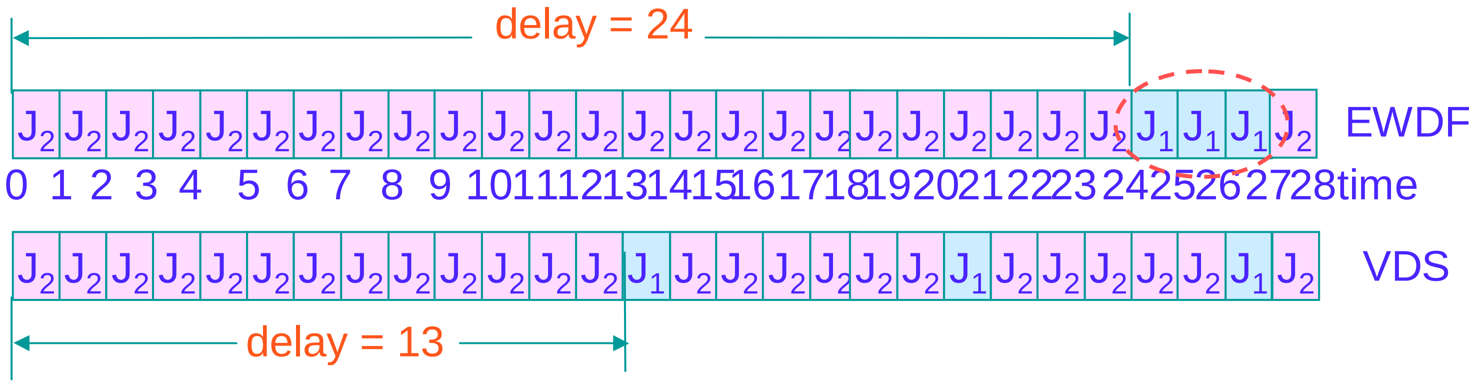
Job	(C,T,m,k)
J_1	(1,1,2,9)
J_2	(1,3,1,1)
J_3	(1,3,1,1)
U_{min}	8/9

time	0	1	2	3	4	5	6	7	8	9	
EWDF	J_2	J_3	J_1	J_2	J_3	J_1	J_2	J_3	J_1		
Wd_1	9	9	9	9	9	9	X	X	9	18	…
Wd_2	3	X	X	6	X	X	9	X	X	12	…
Wd_3	3	3	X	6	6	X	9	9	X	12	…

VDS vs. EWDF

- EWDF
  - Feasible for the relaxed model if $U_{min} \leq 1$
  - Worst case delay: $(k_i T_i - m_i C_i)$
  - More deadlines missed
  - Complexity: O(n) in worst case

Job	(C,T,m,k)
J_1	(1,7,3,4)
J_2	(1,1,24,27)
U_{min}	0.996



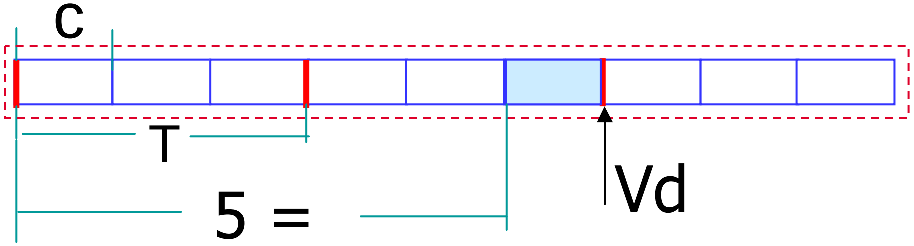
Service Share and Delay Bound

If a feasible VDS schedule exists:

- The minimum service share for each job i is m_iC_i/k_iT_i
- The maximum delay for each job i is $(k_i – m_i +1)T_i – C_i$

EWDF: $k_iT_i-m_iC_i$

J : C=1, T=3, m=2, k=3



(3-2+1)*3-1

(k–m+1)T – C

Feasibility Test

VDS guarantees 100% utilization for a job set with all $C_i = \Delta$, and $T_i = q_i \Delta$ in the relaxed model

- Proof by reduction to a derived EDF scheduling problem
  - Derived EDF: $(C_i, k_i T_i / m_i)$ with only m_i instances
  - VDS equivalent: (C_i, Vd_i) with only m_i instances
  - U(VDS) = U(derived EDF)

  - The relaxed model assures no idle time before overflow
  - Note: VDS allows preemption at the granularity of Δ

Simulations

Computer Science

- Work load:
  - Randomly generate 1,300,000 job sets
  - Variable number of jobs (n) per job set, unit process time C, variable T, m and k for every job

- Performance metrics:
  - $Vtest_s$: # of job sets that violate *service requirement*
  - $Vtest_d$: # of job sets that violate *deadline requirement*
  - V_s: the total *service violation rate* of all jobs
  - V_d: the total *deadline violation rate* of all jobs

Results for the Original Model

Computer Science

- $Vtest_d = Vtest_s \quad V_d = V_s$
- Violation in underload cases:
  - DWCS: $U_{min} > 0.6$
  - EDF-Pfair: $U_{min} > 0.9$
  - VDS: $U_{min} > 0.9$

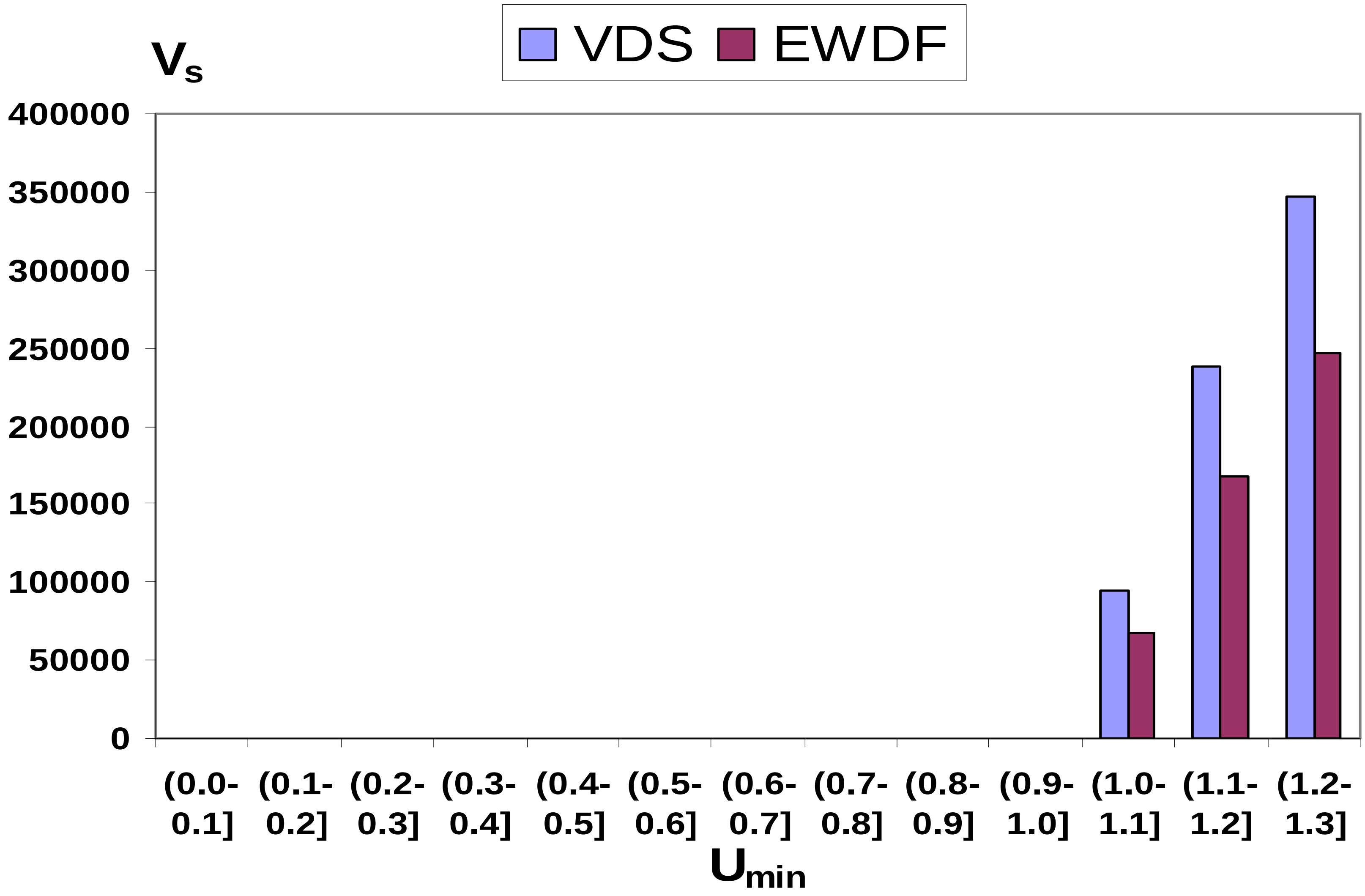
$0.9 < U_{min} \leq 1.0$

	Vtestd	Vd
DWCS	14555	340.46707
EDF-Pfair	77	4.679056
VDS	14	0.6

- VDS has more violations in overload case
  - Tries to maintain proportional fairness

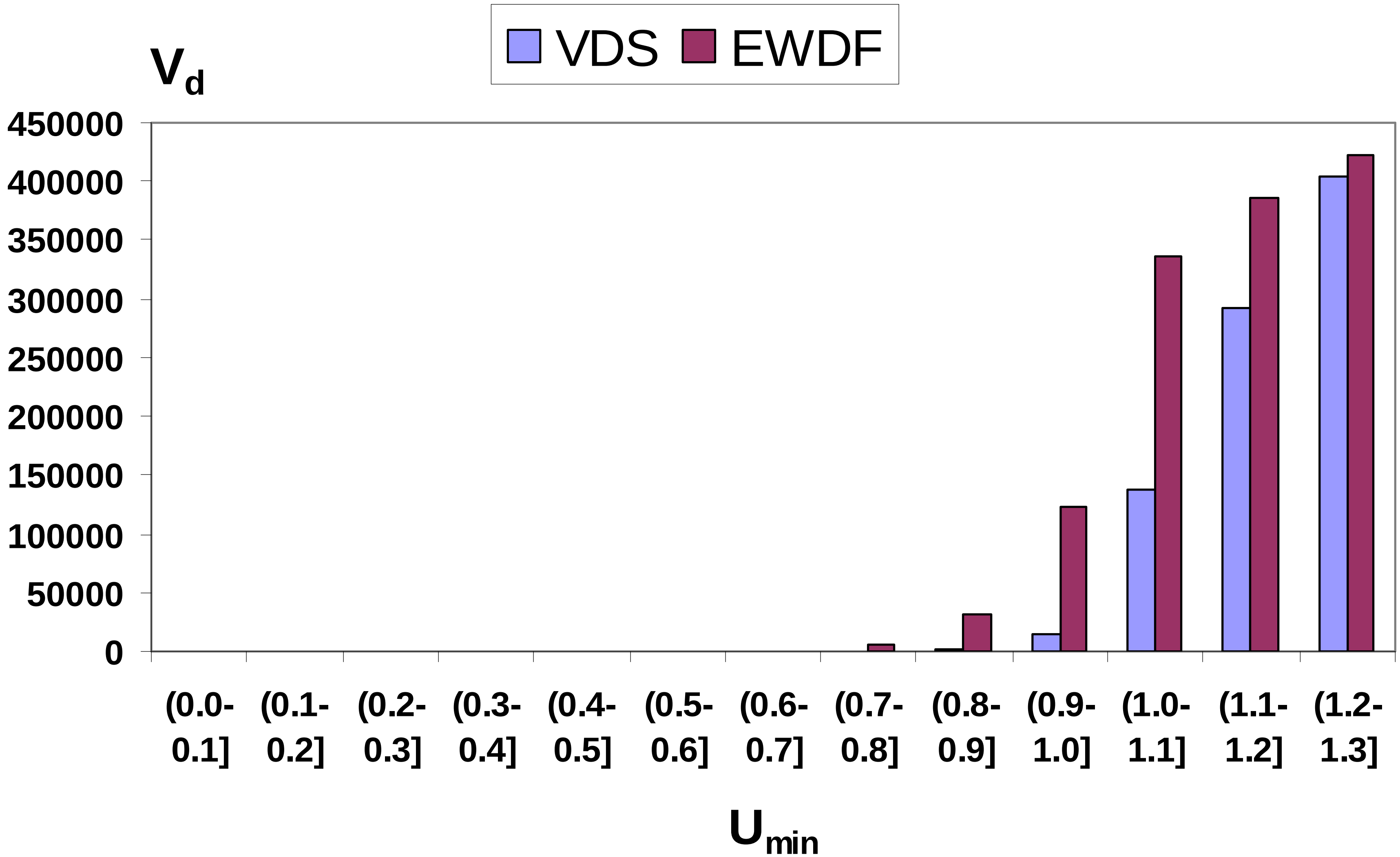
Results for the Relaxed Model

Computer Science

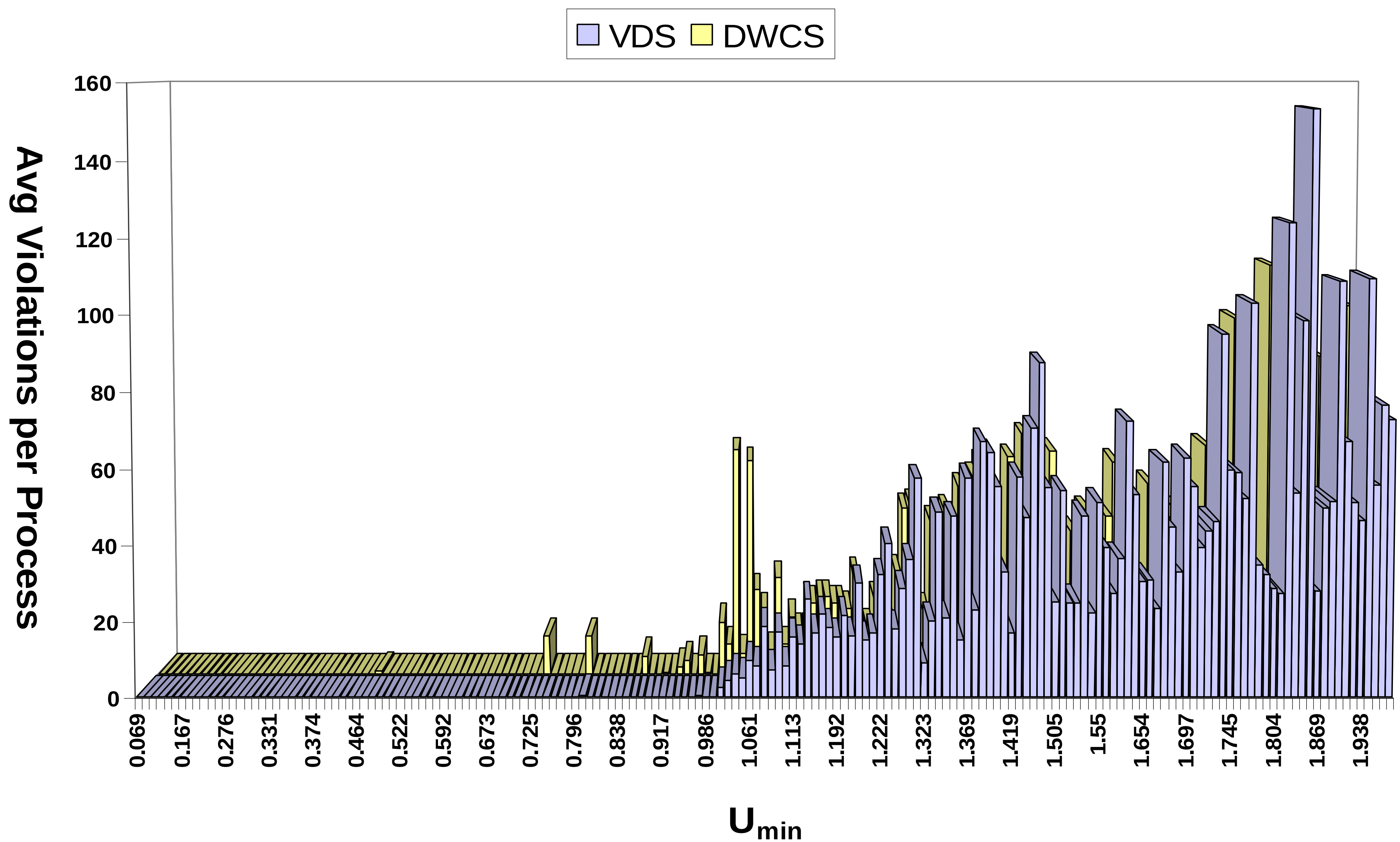


Results for the Relaxed Model

Computer Science



CPU Scheduling – Linux Kernel



Conclusions

- We propose a relaxed (m,k) window-constrained model
  - Appropriate for many classes of applications
  - e.g., multimedia streaming & real-time data sampling
- We present a new algorithm: VDS
  - Can make full use of resources while guaranteeing window-constraints
- Benefits of VDS shown via simulations and real implementation in the Linux kernel