

# Online Scheduling Switch for Maintaining Data Freshness in Flexible Real-Time Systems

Song Han<sup>1</sup> Deji Chen<sup>2</sup> Ming Xiong<sup>3</sup> Aloysius K. Mok<sup>1</sup>

<sup>1</sup>The University of Texas at Austin

<sup>2</sup>Emerson Process Management

<sup>3</sup>Google Inc

# Outline

Motivation

Maintaining Data Freshness

Utilization-based Scheduling Selection

Scheduling Switch with Validity Constraint

Performance Evaluation

Conclusion

# Motivation

Maintaining data quality in real-time systems is important.

- ▶ Real-time data are used to capture current status of entities in the system.
- ▶ Real-time data have time semantics and their quality degrade with time.
- ▶ Many scheduling policies are available.

# Motivation

Maintaining data quality in real-time systems is important.

- ▶ Real-time data are used to capture current status of entities in the system.
- ▶ Real-time data have time semantics and their quality degrade with time.
- ▶ Many scheduling policies are available.

Many real-time systems exhibit multi-modal behavior.

- ▶ Each mode is characterized by a different task set.
- ▶ Different modes have varying system workloads.

# Motivation

Maintaining data quality in real-time systems is important.

- ▶ Real-time data are used to capture current status of entities in the system.
- ▶ Real-time data have time semantics and their quality degrade with time.
- ▶ Many scheduling policies are available.

Many real-time systems exhibit multi-modal behavior.

- ▶ Each mode is characterized by a different task set.
- ▶ Different modes have varying system workloads.

Application: Aircraft Control Systems

- ▶ Real-time data: aircraft position, speed, direction, altitude, etc.
- ▶ Different modes: landing, takeoff, normal cruise, etc.

# Motivation

## Goals:

- ▶ Maintaining data freshness in flexible real-time systems.
- ▶ Achieving the tradeoff between higher data quality and better schedulability.

# Motivation

## Goals:

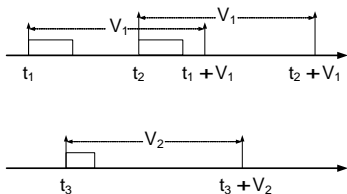
- ▶ Maintaining data freshness in flexible real-time systems.
- ▶ Achieving the tradeoff between higher data quality and better schedulability.

## Problems:

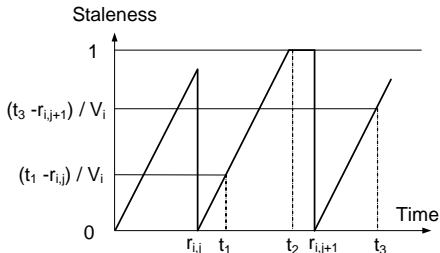
- ▶ Which scheduling policy should be applied to a mode?
- ▶ When to do the switch to maintain data freshness during the transition?

## Temporal Validity and Data Staleness

- ▶ A validity interval is associated with a data value.
- ▶ A data value is fresh within the validity interval.
- ▶ Staleness is used to measure the degradation in data freshness.
- ▶ Increases linearly within the validity interval until the data is refreshed.



$V_1$ : validity length of  $X_1$      $V_2$ : validity length of  $X_2$



# Algorithms for Maintaining Data Freshness

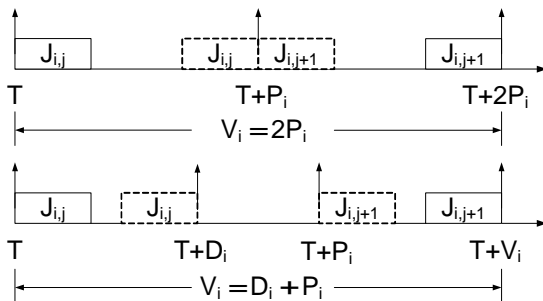
## Half-Half [Ramamritham 93]

- ▶ Period ( $P_i$ ) and relative deadline ( $D_i$ ) of an update transaction  $\tau_i$  are each set to be one-half of the data validity length ( $V_i$ ).

## More-Less [Burns & Davis 96, Xiong & Ramamritham 99]

- ▶ Validity Constraint (to ensure data validity):
  - ▶ Period + Relative Deadline  $\leq$  Validity Length
- ▶ Deadline Constraint (to reduce workload):
  - ▶ Computation Time  $\leq$  Relative Deadline  $\leq$  Period
- ▶ Schedulability Constraint (by deadline monotonic):
  - ▶ Response time of the 1st instance  $\leq$  Relative Deadline
  - ▶ 1st instance response time is the longest response time of all instances of a transaction if all periodic transactions start synchronously

## Algorithms for Maintaining Data Freshness



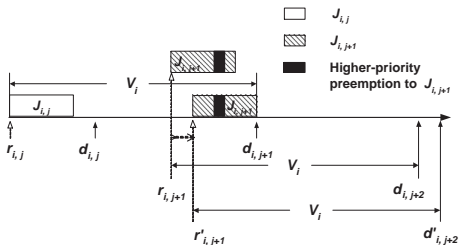
### Remark

- ▶ More-Less is pessimistic because the relative deadline is fixed and equal to the worst case response time of the transaction.

# Algorithms for Maintaining Data Freshness

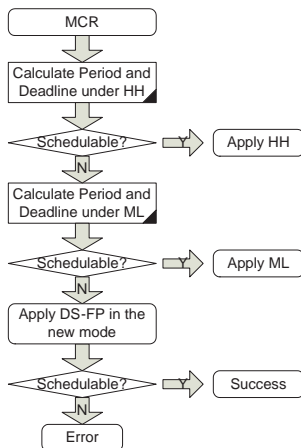
## Deferrable Scheduling with Fixed Priority (DS-FP) [Xiong, Han & Lam 05]

- ▶ Adopts the sporadic task model.
- ▶ Defers the sampling time,  $r_{i,j+1}$ , of  $J_{i,j}$ 's next job as late as possible.
- ▶ Increases the distance of two consecutive jobs as much as possible.



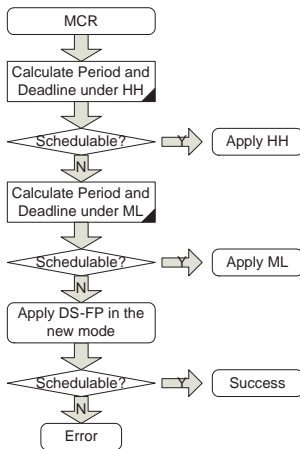
# Utilization-based Scheduling Selection

- ▶ Try *HH*. Use Liu & Layland's schedulability test condition.

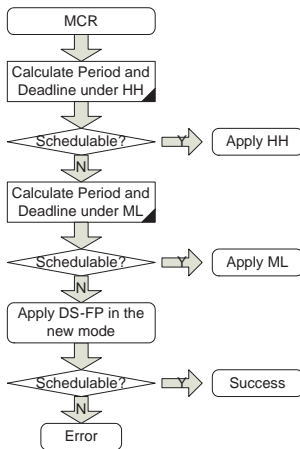


# Utilization-based Scheduling Selection

- Try *ML*. Need to satisfy the three constraints in *ML*.



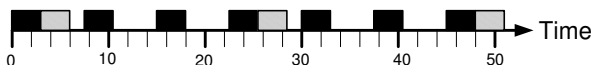
# Utilization-based Scheduling Selection



► Try *DS-FP*.

## An Example: Mode 1 is schedulable under $HH$

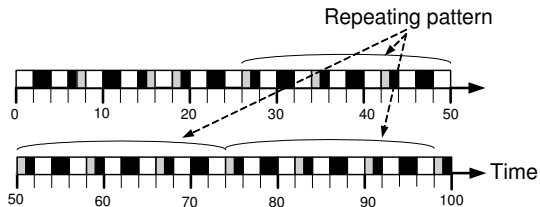
Mode1: ■  $T_2: \{C_2=3, V_2=15\}$  □  $T_3: \{C_3=3, V_3=47\}$



- ▶  $T_i$ : the  $i^{th}$  task in the system.
- ▶  $C_i$ : the computation time of  $T_i$ .
- ▶  $V_i$ : the validity interval of  $T_i$ .
- ▶  $U_{HH} = \frac{2 \times C_1}{V_1} + \frac{2 \times C_2}{V_2} = 0.525 < 0.828$

## Switch to *DS-FP* in mode 2 to increase schedulability

Mode2:  $\square$   $T_1: \{C_1=2, V_1=6\}$   $\blacksquare$   $T_2: \{C_2=3, V_2=15\}$   $\square$   $T_3: \{C_3=3, V_3=47\}$

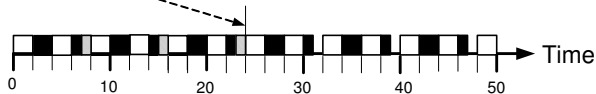


- Mode 2 is not schedulable under *HH* or *ML*, but has a repeating pattern under *DS-FP*

## Switch to *ML* in mode 3 to improve data quality

Mode3:  $\square$   $T_1: \{C_1=2, V_1=6\}$   $\blacksquare$   $T_2: \{C_2=3, V_2=15\}$   $\square$   $T_3: \{C_3=3, V_3=49\}$

$J_{3,0}$  completes before  $V_3/2$



- ▶ Mode 3 is schedulable under *ML* but not *HH*

# Scheduling Switch with Validity Constraint

- ▶ Major Challenges
  - ▶ The temporal validity of the tasks persistent through the switch could be violated during the scheduling switch.
  - ▶ How to choose the switch point to avoid these violations?

# Scheduling Switch with Validity Constraint

## ▶ Major Challenges

- ▶ The temporal validity of the tasks persistent through the switch could be violated during the scheduling switch.
- ▶ How to choose the switch point to avoid these violations?

## ▶ Clean Switch

- ▶ There is no outstanding execution from the old task set at the switch point.
- ▶ The new policy can schedule the new task set independently.

# Scheduling Switch with Validity Constraint

## ▶ Major Challenges

- ▶ The temporal validity of the tasks persistent through the switch could be violated during the scheduling switch.
- ▶ How to choose the switch point to avoid these violations?

## ▶ Clean Switch

- ▶ There is no outstanding execution from the old task set at the switch point.
- ▶ The new policy can schedule the new task set independently.

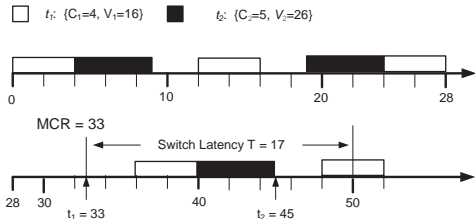
## ▶ Non-clean Switch

- ▶ How to schedule the outstanding executions from the old task set?

# Search-based Switch

## Basic Idea:

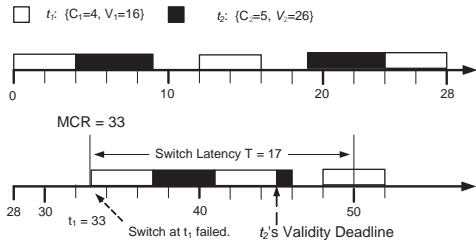
- ▶ Strictly follows the clean switch requirements.
- ▶ We only need to check the temporal validity at the begin of each idle period.



# Search-based Switch

Basic Idea:

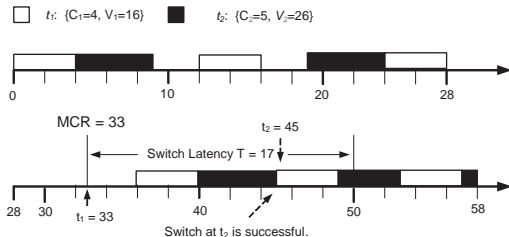
- ▶ Strictly follows the clean switch requirements.
- ▶ We only need to check the temporal validity at the begin of each idle period.



# Search-based Switch

Basic Idea:

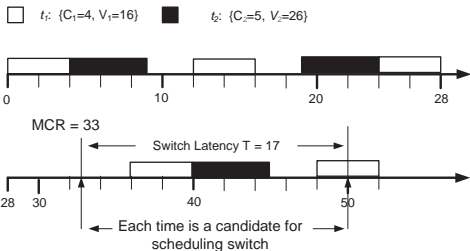
- ▶ Strictly follows the clean switch requirements.
- ▶ We only need to check the temporal validity at the begin of each idle period.



# Adjustment-based Switch

## Basic Ideas:

- ▶ Takes every time point in  $[t_{MCR}, t_{MCR} + t_L]$  as the candidate for scheduling switch.
- ▶ Converts the non-clean switch to clean switch scenario through schedule adjustment.
- ▶ Pushes all outstanding executions back to the switch point  $t_w$ . Adjust the schedule in  $[t_{MCR}, t_w]$  backwards from time  $t_w$  to guarantee the validity constraints.





# Performance Evaluation

## Simulation Model and Parameters

- ▶ Single CPU, main memory based RTDBS and up to 10 modes in the system.
- ▶ The number of real-time data objects in the system varies from 1 to 20.
- ▶ The validity length is uniformly distributed from 50 to 150 time units.
- ▶ the computational time is uniformly distributed from 1 to 5 time units.

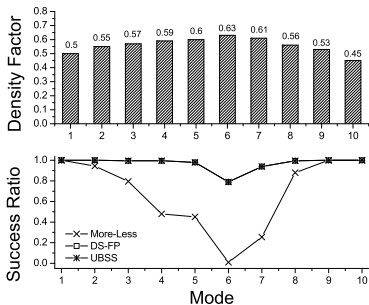
## Experiment Design

- ▶ Single scheduling policy vs. Online utilization-based scheduling switch (UBSS).
- ▶ Search-based Switch (SBS) vs. Adjustment-based Switch (ABS).

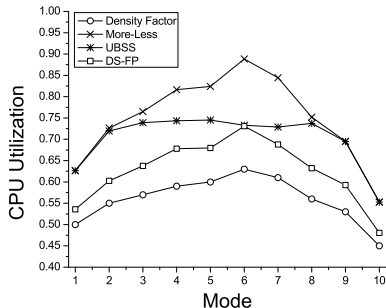
## Performance Metrics

- ▶ CPU Utilization, Scheduling Success Ratio, Data Staleness, and Switch Latency.

# Success Ratio vs. CPU Utilization



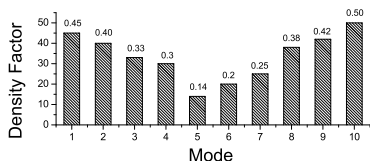
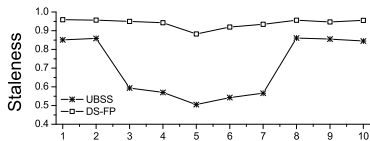
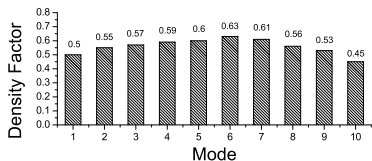
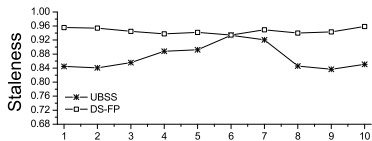
- ▶ *DS-FP* performs the best.
- ▶ The CPU utilization of *UBSS* is between the *ML* and *DS-FP*.



- ▶ The success ratio drops along with the increase of the density factor.
- ▶ *DS-FP* and *UBSS* outperform *ML*.

# Data Staleness Improvement with UBSS

## Heavy System Workload vs. Light System Workload





# Conclusion

- ▶ We studied the problem how to maintain the temporal validity of real-time data in the presence of mode changes in flexible real-time systems.
- ▶ We proposed the online utilization-based scheduling switch (UBSS) to tradeoff between higher data quality and better schedulability.
- ▶ Two algorithms are proposed to search for the proper switch point online to maintain the data temporal validity during the transition.

## Appendix: Search-based Switch Algorithm

---

### Alg 1 Search-based Switch Algorithm

---

**Input:**  $\mathcal{T}_k, \mathcal{T}_{k+1}, \Psi_k, \Psi_{k+1}$ , the search start time  $t_0$  and  $t_L$ .

**Output:**  $t_w$ .

```

1: for  $t = t_0$  to  $t_0 + t_L$  do
2:   if  $t = t_1$  then
3:     //  $t_1$  is the begin point of an idle period
4:      $f = true$ ;
5:     // Whether  $t$  is a possible candidate for  $t_w$ 
6:     for each  $\tau_i \in \mathcal{T}_k \cap \mathcal{T}_{k+1}$  do
7:        $t_s =$  release time of  $\tau_i$ 's last job in  $\Psi_k$ ;
8:        $l =$  time to finish  $\tau_i$ 's first job in  $\Psi_{k+1}$ ;
9:       if  $t - t_s + l > V_i$  then
10:         $f = false$ ;
11:     if  $f = true$  then
12:       return  $t$ ;
13: return no  $t_w$  exists;

```

---

## Appendix: Adjust-based Switch Algorithm

---

### Alg 1 Adjustment-based Switch Algorithm

---

**Input:**  $\mathcal{T}_k, \mathcal{T}_{k+1}, \Psi_k, \Psi_{k+1}, t_0$  and  $t_L$ .

**Output:**  $t_w$ .

```

1: for  $t = t_0$  to  $t_0 + t_L$  do
2:   //  $\sum_i \Gamma_i(t)$  is accumulated outstanding execution at  $t$ 
3:   if  $I(t_0, t) < \sum_i \Gamma_i(t)$  then
4:     continue;
5:   else
6:     // Adjust the schedule of  $\mathcal{T}_k$  in  $[t_0, t]$ 
7:      $f = \text{ScheduleAdjustment}(\mathcal{T}_k, t_0, t)$ ;
8:     if  $f = \text{fail}$  then
9:       continue;
10:    else
11:      for each  $\tau_i \in \mathcal{T}_k \cap \mathcal{T}_{k+1}$  do
12:         $t_s =$  adjusted request time of  $\tau_i$ 's last job in
13:         $l =$  time to finish  $\tau_i$ 's first job in  $\Psi_{k+1}$ ;
14:        if  $t - t_s + l > V_i$  then
15:          // The temporal validity is violated.
16:           $f = \text{fail}$ ;
17:        if  $f = \text{success}$  then
18:          return  $t$ ;
19: return no  $t_w$  exists;

```

---

## Future Works

- ▶ Suppose the MCR latency requirement is infinite, if the old and new task sets are schedulable under the old and new scheduling policies respectively, does there exist a proper switch point using SBS or ABS?
- ▶ How to handle the situation when the proper switch point cannot be found?
- ▶ SBS and ABS are both synchronous algorithms. Can we design asynchronous algorithms? If so, how should scheduling switch be conducted?