Foundations of Cyber-Physical Systems

Precedence Constraints

Rupak Majumdar & Björn Brandenburg

Precedence Constraints

- **pipelines** of tasks (or a **transaction**): output of task in one stage is input of task in next stage
- “pipeline” typically means periods are equal → The **task graph** has a common period.

Why?

- **pipelining**: improved throughput by parallel execution of overlapping stages
- **producer-consumer**: cannot execute job until prior stage has produced output
- **freshness constraint**: choose to wait for latest sensor reading if last sample is “too old”
Aside: Multirate Information Flow

What if inputs are consumed at different rates than the rate at which output is produced?

1. **Always read latest value**: new sample overwrites prior sample.
   - Maximum age of consumed values?
   - Indirectly using multiple samples?

2. **Buffer samples until they are consumed**: consumer may get “batch” of samples.
   - Maximum buffer space required?
   - Maximum queuing delays?
Precedence Constraints: Key Questions

1. What is the **end-to-end** maximum response time?  
   → Is the **end-to-end** deadline met?

2. What is the **impact on other tasks**?  
   → Added constraints change schedule.

**Observation:** Jobs of tasks in later stages not immediately available for execution.  
→ **Precedence constraints can induce jitter.**
Precedence Constraints on a Uniprocessor

Given a pipeline $T_i \rightarrow T_j \rightarrow T_k$ with a common period.

→ All three tasks release a job at the same time
→ Periodically or sporadically.

**Question:** How to ensure correct ordering of jobs at runtime?
Precedence Constraints on a Uniprocessor

Given a pipeline $T_i \rightarrow T_j \rightarrow T_k$ with a common period.
→ All three tasks release a job at the same time
→ Periodically or sporadically.

**Question:** How to ensure correct ordering of jobs at runtime?
Trivial! (Assuming absence of self-suspections.)

1. FP: Assign priorities $i < j < k$.

2. EDF: Assign relative deadlines $d_i < d_j < d_k$. 
Precedence Constraints in a Distributed System

Why distribute task graphs across several compute nodes?

- **Physically distributed** sensors and actuators.
  → Example: speed sensors at each wheel to detect loss of traction.

- Processing **capacity** limitations.
  → Example: offload core computation to a DSP or GPU.

- **Spatial redundancy**.
  → Example: spread out replicas across different parts of aircraft.
Review: Jitter

**Jitter** $j_i$: difference between when a job is conceptually *released* and when it *arrives*.
→ Release: determines earliest possible release time of next job
→ Arrival: job is actually available for execution.

**Effect**: moves work into busy window.
→ Increased interference.
→ Non-linear: a little bit of jitter can add an entire extra job.
Jitter Effects due to Precedence Constraints

if successor jobs arrive immediately when predecessor completes

Suppose $T_i \rightarrow T_j$, and $T_i$ resides on different host than $T_j$.

In the best case, $T_i$ experiences no interference and executes for its best-case execution time (BCET). In the worst case, $T_i$ experiences maximum interference and executes for its WCET.

Effect: $T_j$’s jitter is increased by $r_i$, $T_i$’s worst-case response time. $\rightarrow r_i$ depends on $T_i$’s jitter, so jitter accumulates in multi-stage pipeline.
Review: Phase

**Phase** $\phi_i$: release time of the first job $J_{i,1}$ of a task $T_i$ w.r.t. a global time zero (the task set release, i.e., when system operation started).

A **periodic task** $T_i$ releases job $J_{i,x}$ exactly at time:

$$a_{i,x} = \phi_i + (x - 1) \cdot p_i$$
Phase-Modification Protocol (PM)

for periodic pipelines

Observation: worst-case end-to-end response time is determined by maximum delays.
→ Jitter arises from variability of delays.

Idea: Simply force the worst-case delay to avoid variability.

PM: If $T_i \rightarrow T_j$, then $\phi_j \geq \phi_i + r_i$.
→ Output of predecessor $T_i$ always available when $T_j$ releases a job.
PM Discussion

• PM is conceptually very simple
  → end-to-end response time of $T_a \rightarrow \ldots \rightarrow T_z$ is $\phi_z + r_z$

• zero jitter increase: no schedulability penalty
  → improved worst case

• Forced worst case
  → decreased average-case end-to-end response times

• Globally \textit{synchronized} clocks required

• What about sporadic pipelines?

© 2014 B. Brandenburg (MPI-SWS)
Observation: w.r.t. schedulability, jitter is problematic only if it shifts work into a busy window.

Idea: maintain minimum separation of jobs during busy window.
When the first job in a busy window incurs a large jitter, just make sure later jobs in the busy window don’t have less.
Release Guard Protocol — Rules

Let $t_a$ denote the arrival of job $J_{i,j-1}$. Let $t_{in}$ denote the input for job $J_{i,j}$ becomes available.

1. If there was no idle instant during $(t_a, t_a + p_i)$, then $J_{i,j}$ arrives at time $\max(t_a + p_i, t_{in})$.

2. If the system first becomes idle at time $t_{idle} \in (t_a, t_a + p_i)$, then $J_{i,j}$ arrives at time $\max(t_{idle}, t_{in})$. 
Release Guard — Discussion

• **Event-driven** protocol: no clock synchronization required. → local clock required, still sensitive to **clock drift**

• **Simple** to implement: just maintain a **release guard** (= “earliest-possible arrival time”) variable for each task, reset on idle instant.

• **No scheduling penalty**: does not make the worst case worse. → zero jitter increase
  → end-to-end response time = sum of individual response times

• Average case (= short busy windows) not much affected.
Exploiting Offset Information in Schedulability Analysis

• Precedence constraint $T_i \rightarrow T_j$
  $\rightarrow$ jobs of $T_i$ and $T_j$ never pending at the same time.

• In particular, with PM (or phases in general), there may be a gap between jobs of $T_i$ and $T_j$.

• How can we exploit this?
  $\rightarrow$ Need more sophisticated response-time analysis.\textsuperscript{Tin94}

Dealing with Cycles

- What if the task graph has cycles?
- Example: feed back information about actuation time.
- Need to **break cycles** during analysis to compute response times.\textsuperscript{JA09}

Dealing with Communication Delays

What if intermediate values are communicated over shared network? → **Communication delays** that increase end-to-end response time.

**Approach:** model delays explicitly as **communication** (or network) **tasks**.²⁹⁴

\[ T_i \rightarrow T_{com(i,j)} \rightarrow T_j \]

→ Need response-time analysis for transmission task \( T_{com(i,j)} \) ...

Response-time analysis for transmission task $T_{com}(i,j)$

Schedulability Analysis of Non-Preemptive Fixed-Priority Scheduling for the CAN Bus
The Controller Area Network (CAN) Bus

• CAN was designed (by Bosch 1983-1987) as a real-time bus to reduce the number of cables required in modern cars.
  → Multiplex many signals on a few wires.

• Now used in many areas, including automotive, robotics, factory automation, etc.

• A comprehensive treatment of CAN is a course topic in itself.

• Focus here: how to carry out schedulability analysis.
CAN Packets & Bus Arbitration

Carrier Sense Multiple Access / **Collision Resolution** (CSMA/CR)
Hosts start sending simultaneously → lowest-ID message wins.
Non-Preemptive Fixed-Priority Scheduling

• Just like regular fixed-priority scheduling...
  → Message types = tasks
  → message = job
  → WCET = message length / bit rate
  → period, deadline, jitter, phase as before.

• ...but lower-priority executing tasks / messages in transmission cannot be preempted.
  → Need to account for priority inversions.
Classic Analysis (for Constrained Deadlines)

**Intuition:** It’s just fixed-priority RTA with maximum priority inversion length $\approx$ maximum lower-priority message length.

$$R_i = e_i + b_i + \sum_{h<i} \left\lfloor \frac{R_i}{p_h} \right\rfloor \cdot e_h,$$

where $b_i = \max_{l>i} \{e_l\}$. Reinder Bril$^{Bril06}$ showed this to be wrong...

---

Counter Example

Response time of first instance of message C: **3ms.**
Response time of second instance of message C: **3.5ms.**

Naïve classic analysis does not reflect that higher-priority demand is “pushed through” due to non-preemptive execution.
Revised Analysis$^{\text{DBBL07}}$

As with arbitrary deadlines, one must consider all messages in the busy window...

\[ R_i = \max\{w_i(q) \mid q = 1, 2, \ldots\}, \]

where \( w_i(q) \) is the smallest positive solution to the recurrence

\[ w_i(q) = \max_{l>i}\{e_l\} + q \cdot e_i + \sum_{h<i}\left[\frac{w_i(q) + (q-1) \cdot p_i}{p_h}\right] \cdot e_h - (q-1) \cdot p_i. \]