Real-Time Systems

Anno accademico 2009/10
Laurea magistrale in informatica
Dipartimento di Matematica Pura e Applicata
Università di Padova
Tullio Vardanega

Outline

1. **Introduction**
2. **Dependability issues**
3. **Scheduling issues**
4. **More on fixed-priority scheduling**
5. **Task interactions and blocking**
6. **System issues**
7. **Multi-cores and distribution**

Bibliography

- J. Liu, "Real-Time Systems", Prentice Hall, 2000
- A. Burns, A. Wellings, "Concurrent and Real-Time Programming in Ada", Cambridge University Press, 2007
- A. Burns, A. Wellings, "Real Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time C/POSIX", Addison-Wesley, 2009

7.b Multi-cores

Credits to A. Burns and A. Wellings

and to B. Andersson and J. Jonsson for their work in *Proc. of the the IEEE Real-Time Systems Symposium*, WiP Session, 2000, pp. 53–56

Fundamental issues

- Hardware architecture taxonomy
  - Homogeneous or heterogeneous processors
    - Current research is focused on SMP (symmetric multiprocessors) as the scheduling problem is much simpler
- Scheduling approach
  - Global or partitioned or alternatives between these extremes
  - Partitioning is an allocation problem followed by single processor scheduling
- Optimality criteria
  - EDF is no longer optimal
  - EDF is not always better than FPS
  - Global scheduling is not always better than partitioned

Hardware architecture taxonomy

- A multiprocessor (or multi-core) is *tightly coupled* so that global status and workload information on all processors (cores) can be kept current at low cost
  - The system may use a centralized dispatcher and scheduler
  - When each processor (core) has its own scheduler, the decisions and actions of all schedulers are coherent
    - Scheduling in this model is an NP-hard problem
- A distributed system is *loosely coupled* (too costly to keep global status) and there usually is a dispatcher / scheduler per processor

State of the art

- Some task sets may be unschedulable even though they have low utilization (much less than the number of processors)
  - This is known as the *Dhall's effect* [Dhall & Liu, 1978]
- Existing necessary and sufficient schedulability tests have exponential time complexity
  - Existing sufficient tests have polynomial time complexity but are pessimistic
- Rate-monotonic priority assignment is not optimal
- No optimal priority assignment scheme with polynomial time complexity has been found yet

Interference

- We know what is the interference I_i on a task *i* for single-processor scheduling
- For global multiprocessor scheduling with *m* processors interference only occurs for tasks *m*+1; *m*+2; …
- Multiprocessor interference can be computed as the sum of all intervals when *m* higher-priority tasks execute <u>in parallel</u> on all *m* processors

Example (Dhall's effect) – 1

Task	T	D	C	U
a	10	10	5	0.5
b	10	10	5	0.5
c	12	12	8	0.67

On 2 processors

$\sum U_i = 1.67 < 2$

- Under global scheduling, EDF and FPS would run **a** and **b** on each of the 2 processors
- But this would leave no time for **c** to complete
  - 7 time units on each processor, 14 in total, but 8 on neither
- Even if the total system is underutilized (**!**)

Example – 2

Task	T	D	C	U
d	10	10	9	0.9
e	10	10	9	0.9
f	12	12	2	0,2

On 2 processors

$\sum U_i = 2$

- Partitioned scheduling does not work here either
- Task **f** cannot reside on just one processor: it needs to migrate from one to the other to find room for execution
- And it also needs that **d** and **e** are willing to use cooperative scheduling

Global scheduling anomalies

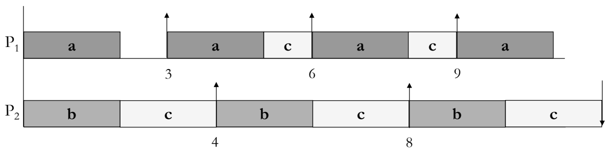
- In real-time scheduling, the deadline miss ratio often highly depends on the system load
- This suggests that increasing the period will decrease the utilization and thus decrease the deadline miss ratio
- **Anomaly 1**: a decrease in processor demand from higher-priority tasks can increase the interference on a lower-priority task because of the change in the time when the tasks execute
- **Anomaly 2**: a decrease in processor demand of a task negatively affects the task itself because the change in the task arrival times make it suffer more interference

Anomaly 1

Task	T	D	C	U
a	3	3	2	0.67
b	4	4	2	0.50
c	12	12	8	0.67

m = 2 processors and $\sum U_i = 1.83$ but task c is *saturated* because $C_c + I_c = D_c$ and thus any increase in C_c makes it unschedulable

If we change T_a to 4 we decrease system load to $\sum Ui = 1.67$ but I_c increases from 4 to 6 (!)

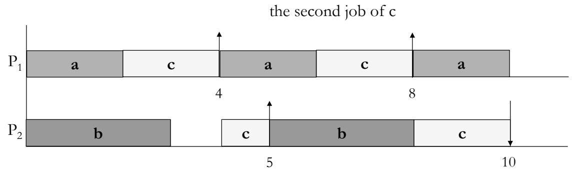


Anomaly 2

Task	T	D	C	U
a	4	4	2	0.5
b	5	5	3	0.6
c	10	10	7	0.7

m = 2 processors and $\sum U_i = 1.8$ but task c is *saturated*

If we change T_c to 11 we decrease system load to $\sum Ui = 1.74$ but I_c increases from 3 to 5 (!) as becomes visible in the second job of c



P-fair scheduling [Baruah et al. 1996]

- *Proportional progress* is a form of proportionate fairness (P-fairness)
  - Each task τ_i is assigned resources in proportion to its *weight* $W_i = C_i/T_i$ hence it progresses proportionately
  - Useful e.g., for real-time multimedia applications
- At every time t task τ_i must have been scheduled either $\lfloor W_i \times t \rfloor$ or $\lceil W_i \times t \rceil$ time units
  - Preemption is assumed to only occur at integral time units (without loss of generality) and the workload model is periodic

P-fair scheduling – 2

- **lag** (S, τ_i , t) is the difference between the total resource allocations that task τ_i should have received in [0,t) and what it received under schedule S
- For a P-fair schedule S at time t
  - Task τ_i is *ahead* iff **lag** (S, τ_i , t) < 0
  - Task τ_i is *behind* iff **lag** (S, τ_i , t) > 0
  - Task τ_i is *punctual* iff **lag** (S, τ_i , t) = 0
- $\alpha(\tau_i , t)$ is the *characteristic substring* of task τ_i at time t
  - Finite string of over {-, 0, +} of $\alpha_{t+1}(x)\ \alpha_{t+2}(x)\ldots\alpha_{t'}(x)$
  - Where t' = min i : i > t : $\alpha_i(x)=0$
  - $\alpha_t(x) =$ **sign** $(w_x \times (t+1) - \lfloor w_x \times t \rfloor - 1)$

P-fair scheduling – 3

- Task τ_i is *urgent* at time t iff τ_i is *behind* and $\alpha_t(\tau_i) \neq$ -
- Task τ_i is *tnegru* (inverse of urgent) at time t iff τ_i is *ahead* and $\alpha_t(\tau_i) \neq +$
- Task τ_i is *contending* otherwise
- General principle of P-fairness
  - Every *urgent* task must be scheduled at time t to preserve P-fairness
  - No *tnegru* task can be scheduled at time t without failing P-fairness
- Possible pitfalls for n_0 *tnegru*, n_1 *contending*, n_2 *urgent* at time t with m resources and $n=n_0+n_1+n_2$
  - If n_2>m the scheduling algorithm cannot schedule all *urgent* tasks
  - If n_0>n-m the scheduling algorithm is forced to schedule some *tnegru* tasks

P-fair scheduling – 4

- The **PF** scheduling algorithm
  - Schedule all *urgent* tasks
  - Allocate the remaining resources to the highest-priority *contending* tasks according to the total order function $\supseteq$ with ties broken arbitrarily
    - x $\supseteq$ y iff α(x, t) ≥ α(y, t)
    - And the comparison between the characteristics substrings is resolved lexicographically with - < 0 < +
- With PF we have $\sum_{x\in[0,n)} w_x = m$
  - A dummy task may need to be added to the task set to top utilization up
- The earlier pitfalls <u>cannot</u> happen with the PF algorithm

Example (PF scheduling) – 1

Task	C	T	W
v	1	3	0.333…
w	2	4	0.5
x	5	7	0.714…
y	8	11	0.727…
z	335	462	3-U

- m = 3 processors
- n = 4 tasks
- Task **z** is a dummy used to top system utilization up
- In general its period is set to the system hyperperiod
  - This time we halved it
- With PF we always have $n_2 > m$ and $n_0 \leq$ n-m

Example (PF scheduling) – 2

t	lag × period					characteristic string					urgent tasks	contending tasks	tnegru tasks
	v	w	x	y	z	v	w	x	y	z			
0	0	0	0	0	0	−	−	−	−	−	{}	$y > z > x > w > v$	{}
1	1	2	−2	−3	−127	−	0	+	+	+	$\{w\}$	$y > z > x > v$	{}
2	2	0	3	−6	−254	0	−	+	+	+	$\{v, x\}$	$w > y > z$	{}
3	0	−2	1	2	81	−	0	−	−	−	{}	$y > z > x > v$	$\{w\}$
4	1	0	−1	−1	−46	−	−	+	+	+	{}	$y > z > x > v = w$	{}
5	2	2	−3	−4	−173	0	0	+	+	+	$\{v, w\}$	$y > z > x$	{}
6	0	0	2	−7	162	−	−	0	+	+	$\{x, z\}$	$w > y > v$	{}
7	1	−2	0	1	35	−	0	−	−	−	{}	$y > z > x > v$	$\{w\}$
8	2	0	−2	−2	−92	0	−	+	+	+	$\{v\}$	$y > z > x > w$	{}
9	0	2	3	−5	−219	−	0	+	+	+	$\{w, x\}$	$y > z > v$	{}
10	1	0	1	−8	116	−	−	−	0	−	{}	$z > x > v = w$	$\{y\}$
11	−1	2	−1	0	−11	0	0	+	−	+	$\{w\}$	$y > z > x$	$\{v\}$
12	0	0	4	−3	−138	−	−	+	+	+	$\{x\}$	$y > z > w > v$	{}
13	1	2	2	−6	−265	−	0	0	+	+	$\{w, x\}$	$v > y > z$	{}
14	−1	0	0	2	70	0	−	−	−	−	{}	$y > z > x > w$	$\{v\}$
15	0	2	−2	−1	−57	−	0	+	+	+	$\{w\}$	$y > z > x > v$	{}
16	1	0	3	−4	−184	−	−	+	+	+	$\{x\}$	$y > z > v = w$	{}
17	2	2	1	−7	−311	0	0	−	+	+	$\{v, w\}$	$x > y > z$	{}
18	0	0	−1	1	24	−	−	+	−	−	{}	$y > z > x > w > v$	{}
19	1	2	−3	−2	−103	−	0	+	+	+	$\{w\}$	$y > z > v = x$	{}

Some results – 1

- For the simplest workload model made of independent periodic and sporadic tasks
  - A *P-fair* scheme can theoretically schedule up to a total utilisation U = *M* for *M* processors, but its run-time overheads are excessive
    - Especially because tasks incur very many preemptions and are frequently required to migrate across processors
  - Partitioned FPS first-fit (on decreasing task utilization) can sustain

$$U \le M(\sqrt{2}-1)$$

  - i.e., 0.414 × *M*
  - But this is a sufficient test only [Oh & Baker, 1998]

Some results – 2

- Partitioned EDF first-fit can sustain

$$U \le \frac{\beta M + 1}{\beta + 1}$$

Per task

$$\beta = \left\lfloor \frac{1}{U_{\max}} \right\rfloor$$

- For high U_{max} gets rapidly lower than 0.6 × *M*, but can get close to *M* for some examples
- Again this is a sufficient test only [Lopez *et al.*, 2004]

Some results – 3

- Global EDF can sustain

$$U \le M - (M-1)U_{\max}$$

- For high U_{max} can be as low as 0.2 × *M* but also close to *M* for other examples
- Again, only sufficient [Goossens *et al.*, 2003]

Some results – 4

- Combinations
  - FPS (higher band) to those tasks with $U_i > 0.5$
  - EDF for the rest

$$U \le \left(\frac{M+1}{2}\right)$$

  - Again, only sufficient [Baruah, 2004]

Multiprocessor PCP – 1

- Proposed by [Sha, Rajkumar, & Lehoczky, 1988] for globally shared resources
- Assumes tasks and resources statically bound to processors
  - The host processor for a resource is called the *synchronization processor* for that resource
  - The FPS scheduler for each synchronization processor knows the priorities and resources requirements of all tasks requiring access to its globally shared resources
- We need actual locks to guarantee protection from true parallelism (which makes lock-free algorithms attractive)
  - The task that holds a lock should not be preempted locally
  - The task that is denied a lock spin-locks (!)

Multiprocessor PCP – 2

- Access to globally shared resources is controlled locally on the synchronization processor according to the Priority-Ceiling Protocol (PCP) except that
  - Access to a globally shared resource is modeled as the task executing a global critical section on the synchronization processor for the resource
  - All global critical sections are executed at higher priorities than local tasks on the synchronization processor

Blocking under M-PCP

- Consequently task T_i incurs five types of blocking
  - *Local blocking time* due to contention for local resources
  - *Local preemption delay* due to the preemption by global critical sections used by remote tasks on T_i's local processor
  - *Remote blocking time* due to contention with lower-priority tasks for remote resources on their synchronization processors
  - *Remote preemption delay* due to preemption by higher-priority global critical sections on synchronization processors of the remote resources required by T_i
  - *Deferred blocking time* due to the suspended execution of local higher-priority tasks

Summary

- Issues and state of the art
- Dhall's effect: examples
- Scheduling anomalies: examples
- P-fair scheduling
- Sufficient tests for simple workload model
- Incorporating global resource sharing