Real-time on General Purpose Systems

Colin Perkins

http://csperkins.org/teaching/2003-2004/rtes4/lecture17.pdf

Reading for this lecture: chapter 10

UNIVERSITY
of
GLASGOW

Programming Assignment

A few questions raised yesterday:

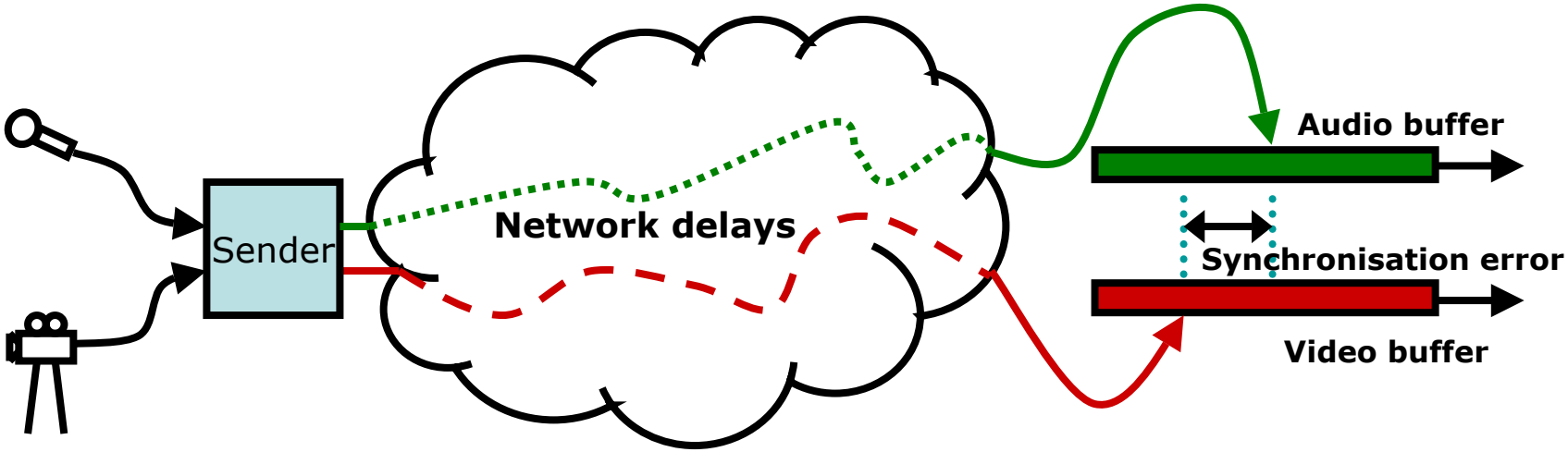
- If the timing graph for one minute is linear and shows little of interest, look at a smaller segment
- Discuss both the long-term and short-term measures
- Expecting relatively short write up: a page or so, plus the graphs is sufficient
- In addition to commenting on what you see in the graphs, don't forget to answer the question: do the systems tested have sufficiently accurate timing to use in a VoIP system?

Due 5pm on Friday, drop box in the usual place.

Lecture Outline

- Scheduling tasks with temporal distance constraints
- Issues with real-time on general purpose systems
  - Flexible computation
  - Approaches to scheduling
  - Implementation strategies

Tasks with Temporal Distance Constraints



- Lip synchronisation is a common operation in multimedia applications
- Audio and video decoding must complete within a short period of time, or the lip-sync is broken

- An example of tasks with temporal distance constraints

Tasks with Temporal Distance Constraints

- Assume a task T_i comprises a set of jobs $J_{i,k}$ for $k = 1, 2, \ldots$
- The first job, $J_{i,1}$, is released at time ϕ_i
- Each subsequent job, $J_{i,k+1}$ where $(k \geq 1)$, becomes ready when its predecessor, $J_{i,k}$, completes
- The finish time of job $J_{i,k}$ is denoted by $f_{i,k}$

- The task has a **temporal distance constraint**, C_i, if:

$$f_{i,1} - \phi_i \leq C_i \qquad \text{(Initial job)}$$

$$f_{i,k+1} - f_{i,k} \leq C_i \quad \text{for } k = 1, 2, \ldots \qquad \text{(Later jobs)}$$

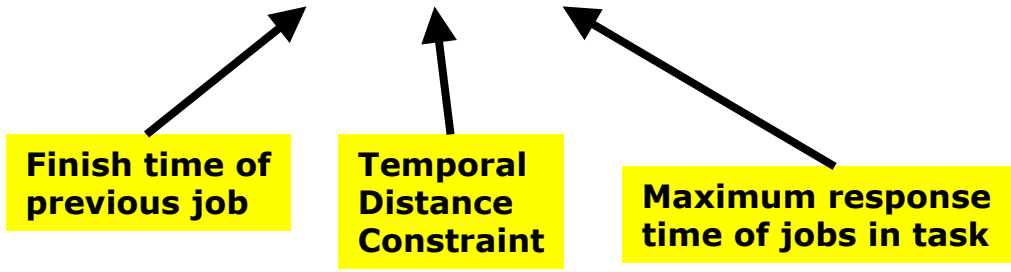
$\Rightarrow$ Jobs must complete within time C_i of their predecessor

Distance Constraint Monotonic (DCM) Scheduling

- Can schedule tasks to explicitly meet distance constraints if appropriate
  - If you care about inter-job timing, as well as each job meeting its deadline
  - Jobs not only meet deadlines, they occur with C_i of the **actual** completion time of an earlier task
- Use a fixed priority scheduling algorithm, similar to deadline monotonic: **Distance Constraint Monotonic** scheduling
- Two elements: priority assignment and job separation
  - Assign task priorities monotonically according to distance constraint
    - Smaller the distance constraint of task T_i, higher the task's priority
    - Jobs run with the fixed priority of the task to which they belong
  - Provide separation between jobs to allow low priority tasks to run and meet their constraints
    - When a job completes, delay it's successor as long as possible, to allow jobs from lower priority tasks to run
    - Improves schedulability

Distance Constraint Monotonic (DCM) Scheduling

- Job $J_{i,k+1}$ is released at time $r_{i,k+1} = f_{i,k} + C_i - W_i$



- The delay $C_i - W_i$ is the **separation constraint**, chosen to ensure that jobs complete and just meet their temporal distance constraint
- Calculate maximum response time W_i iteratively as follows:
  - Find maximum response time W_1 of highest priority task T_1
  - For each T_i for i>1, find W_i after deriving a DCM schedule for all higher priority tasks, assuming all tasks are released at time 0 (worst case response time, when all tasks start at once)
- Assumption: $W_i \leq C_i$
  - otherwise distance constraints will not be met
- Once released, job is scheduled according to task priority
  - Might not execute immediately…

Schedulability of DCM

- Define density of a task, T_i, with execution time e_i and temporal distance constraint C_i is $\delta_i = \frac{e_i}{C_i}$
- Density of the system $\Delta = \sum_{i=1}^{n} \delta_i$

- Assume:
  - distance constraints are harmonic: longer constraints are always integer multiples of shorter
  - separation constraint as previously described

- Theorem: the system is schedulable and meets temporal distance constraints if $\Delta \leq 1$
- Proof: the system devolves into a rate monotonic schedule with period C_i

Schedulability of DCM

- If we relax the system definition to have arbitrary temporal distance constraints, it becomes difficult to prove schedule correct
- Can transform such a system into one with harmonic temporal distance constraints through the **specialization** operation
  - Given a system of tasks T_1, T_2, …, T_n with distance constraints C_1, C_2, … C_n the specialization operation transforms it into a set of **accelerated tasks**
  - The accelerated tasks T_1', T_2', … T_n' have distance constraints C_1', C_2', …, C_n'
  - Where:
    - The execution time of T_i' equals the execution time of T_i
    - The distance constraint $C_i' \leq C_i$
    - The new distance constraints are harmonic
  - Tighten the distance constraints, reducing the schedulable utilisation of the system, but allowing proof of schedulability

Temporal Distance Constraints and DCM

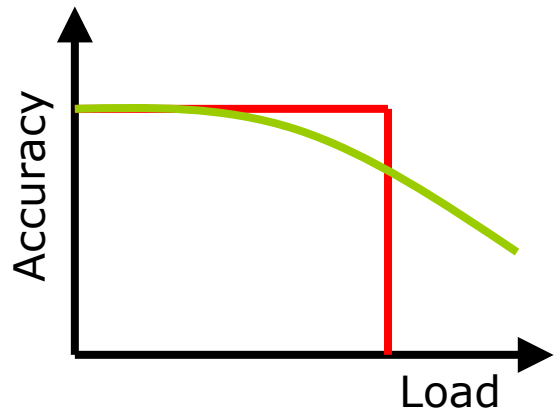
- Some applications care about the time between consecutive jobs in a task
- Can schedule these using the Distance Constraint Monotonic algorithm
  - Similar to deadline monotonic scheduling
  - Ensures actual completion of jobs within fixed period of the previous job in the task, in addition to meeting deadline
- Can often arrange rate monotonic schedules or cyclic executives to meet the temporal distance constraints without special scheduling

Real-time on General Purpose Systems

- Although RTOS are desirable, many real-time systems are built using general purpose operating systems
- Examples running on standard PC hardware:
  - Internet telephony; Streaming audio and video
  - DVD player
  - CD burner
- Operating system may provide limited real-time support
  - POSIX scheduling extensions, or similar
- …but not engineered for robust real-time operation, with many sources of unpredictability
  - Virtual memory and/or disk activity
  - Limited timer resolution
  - Limited scheduler granularity
- Need to engineer applications around these constraints
  - Consider how to make your application flexible

Flexible Computation

- The ability to trade-off, at run time, quality of results for the amount of time and resources used to produce those results
- As a system moves into overload, it gracefully degrades rather than suddenly failing
- Assumption: a timely result of poor quality is better than a high quality, but late, result



- Examples:
  - Multimedia: a fuzzy picture is better than no picture
  - Air traffic control: prefer system to keep working, with error bars, than to fail completely on overload
    - A timely warning of collision, with estimated location better than an exact location, delivered too late to avoid collision

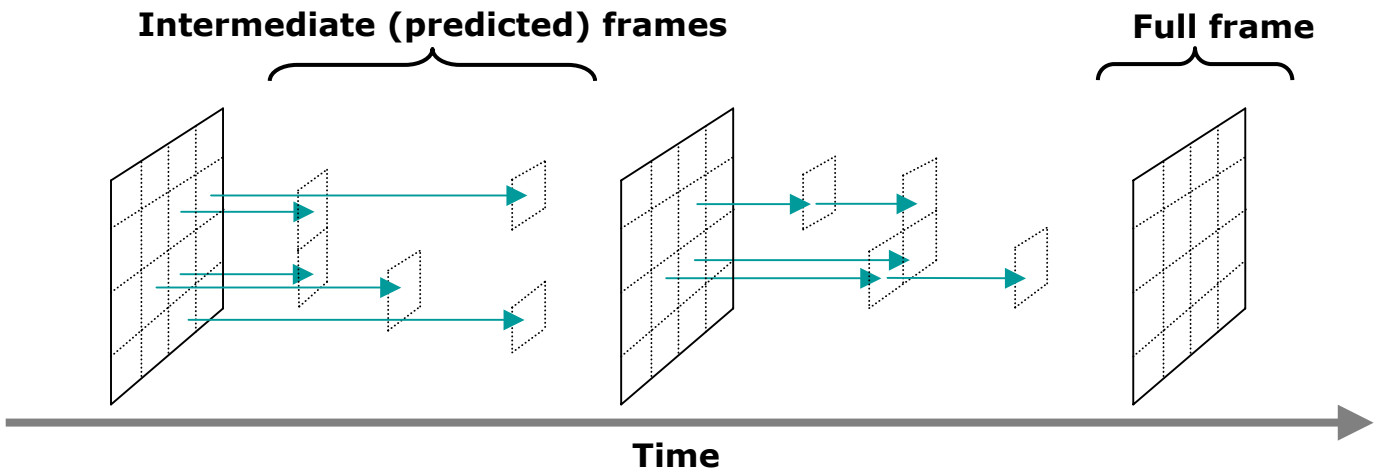
Implementing Flexible Computation

- Jobs have an optional component and a mandatory part
  - If sufficient resources, both mandatory and optional parts complete; a **precise** result
  - If limited resources, the optional component is discarded, giving an **imprecise** result

- How to implement?
  - Sieve method
  - Milestone method
  - Multiple version method

Sieve Method

- A flexible task comprises a mixture of mandatory and optional jobs
- In times of overload, some optional jobs are discarded in their entirety
- Example: Encoding MPEG video

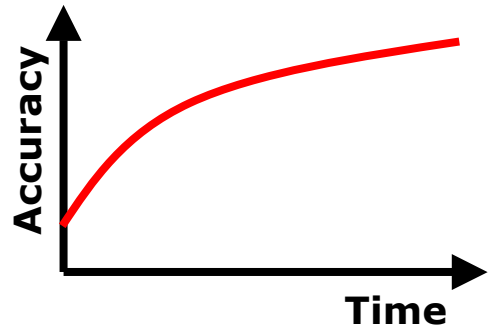


- Can be flexible by either:
  - stop encoding predicted frames ⇒ reduced frame rate
  - delay encoding full frame ⇒ reduced bandwidth, error tolerance

Milestone Method

- The system regularly checkpoints the result of the optional job as a set of **milestones**
- When the deadline is reached, the job terminates and the latest milestone is retrieved

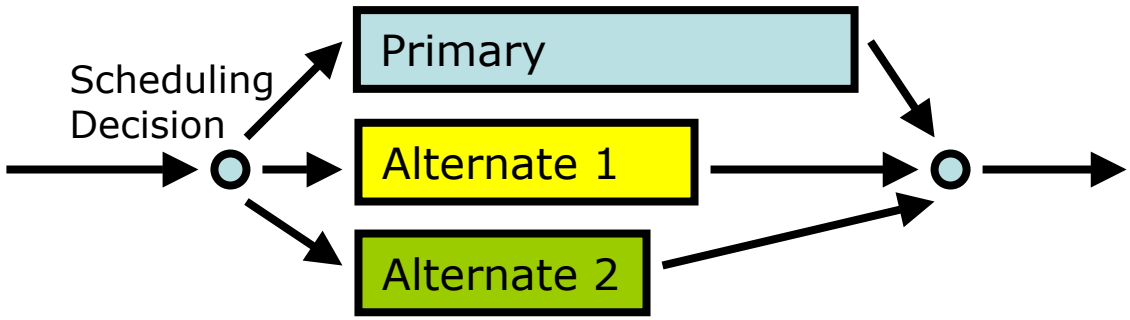
- A **monotone** is a job where the optional component can be stopped at any time, and the quality of the result always increases with longer execution
  - Iterative numerical computation
  - Iterative statistical computation
  - Layered video encoding



- A monotonic job makes the scheduling decision easier, since longer execution, after the mandatory part, always improves quality
  - Otherwise needs watch result quality, to know when to stop

Multiple Versions

- The flexible job is implemented in multiple versions:
  - Primary is high quality, but has a larger execution time and resource usage
  - Alternates are lower quality, but execute quicker or use less resources



- The scheduler must make an *a-priori* decision on which version to execute, based on load at the start of the job
  - Requires more intelligence in the scheduler than sieve or milestone methods
- Little gain from having more than one alternate

Flexible Workload Model

Definitions:

- To schedule flexible computations, need a workload model
- As usual a task, T, is comprised of a series of jobs J_i
- Each flexible job, J_i, is logically decomposed into a chain of two jobs, M_i and O_i which are the mandatory and optional components
- The release times and deadlines of M_i and O_i are the same as J_i but O_i is dependent on M_i
- Execution time $e = e_m + e_o$

- A generalisation of the model we've used previously:
  - non-flexible jobs scheduled as-if e_o is zero

Flexible Workload Model

- Jobs are scheduled so mandatory tasks meet their deadline:
  - A schedule for a flexible application is **valid** if J_i is allocated processor time at least equal to e_m and at most equal to e
  - The schedule is **feasible** if each job is allocated at least e_m units of processor time before its deadline
  - Exactly the same definitions we saw in lecture 2 for non-flexible tasks, adapted to allow for e_o

- Optional components of each job execute if there is time before the deadline
  - An optional job completes it if receives e_o before the deadline
  - An optional job never executes beyond its deadline
    - May be terminated, and revert to the last milestone
    - May be pre-empted, and continue to execute at low priority if killing the job would leave the system inconsistent

Flexible Jobs with 0/1 Constraints

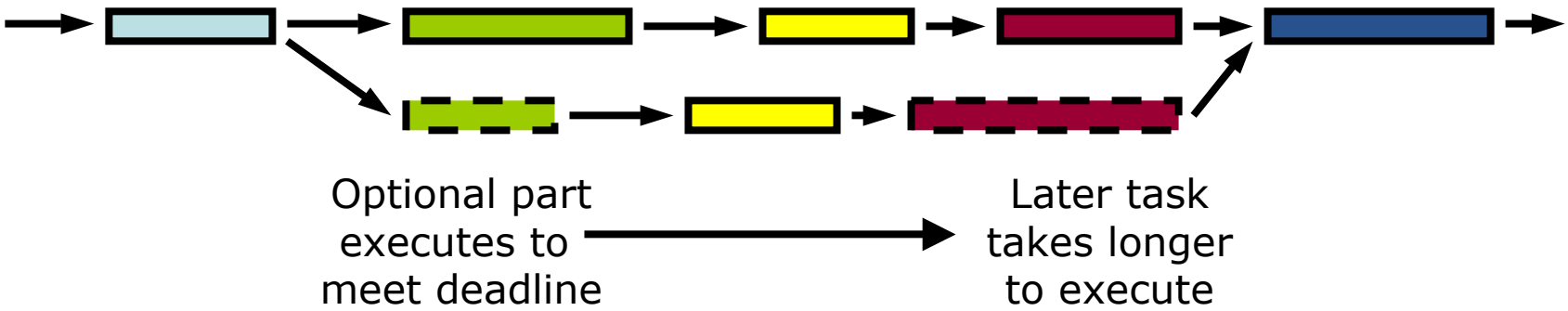
- If the sieve or alternate methods are used, there is no point running part of an optional component
  - The optional component has a **0/1 constraint**
    - Either it runs to completion, or not at all
    - Admission control for jobs

- For optional jobs according to the sieve method:
  - When the optional jobs becomes eligible to run, make a choice to run the job based on available execution time

- For optional jobs according to the alternate method:
  - Model the alternates as mandatory and optional parts
  - Let e_m be execution time of the alternate, e_o be the difference in execution time between primary and alternate
  - After scheduling the mandatory part for e_m, the optional part is scheduled. If e_o available before its deadline, this corresponds to the primary version being scheduled. Otherwise, only the alternate can be scheduled

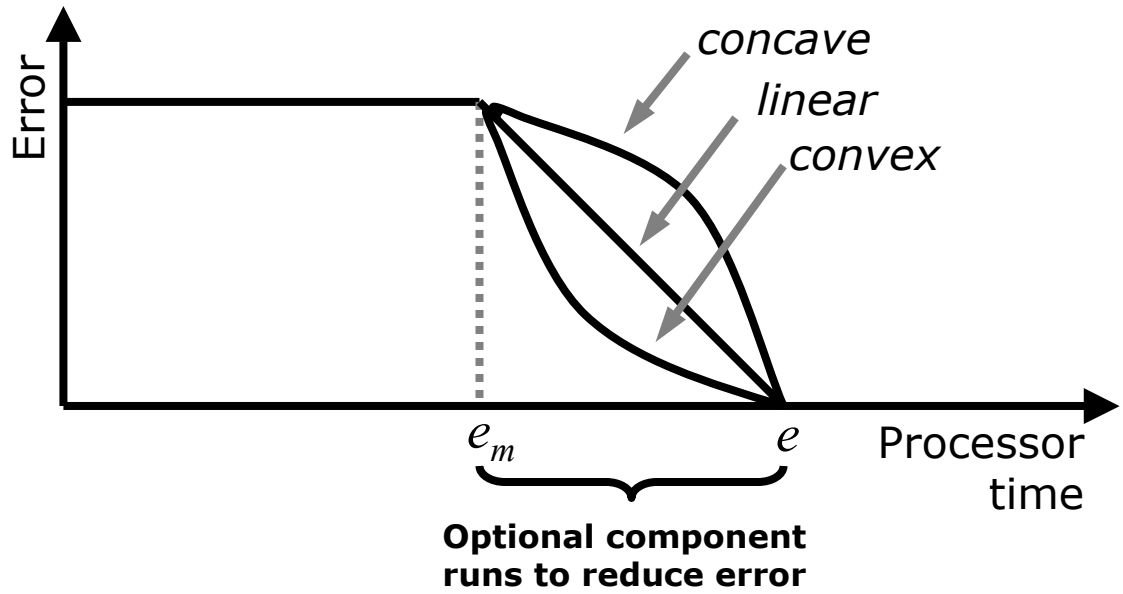
Dependent Jobs

- Assumption of the previous: the execution time of a job is independent of the previous jobs
- In some systems, saving time in an early job – by skipping its optional component – makes a later job in the task take longer
  - Often occurs if errors are cumulative: eventually need to run the full computation periodically, to bring the error back to an acceptable level
- Need to take this into account when building the schedule, by modelling both branches of the task graph



Criteria of Optimality

- Correctness: finding a feasible schedule that ensures all mandatory jobs complete
- Quality of result: try to fit in as many optional jobs as possible, to reduce the error in the result
  - Measure the error according to some domain specific metric
    - Can be difficult to characterise…
  - Clearly desirable if the error function is convex
    - May influence choice of algorithm, for milestone based jobs



Criteria of Optimality

Try to reduce the error in the result… which error:

- The sum of the total errors for all jobs?
- The maximum error for an individual job?
- The average error for all jobs?

Heavily domain dependent…

Scheduling Flexible Applications

- Algorithms for scheduling flexible applications can be either **on-line** or **off-line**
- Given a set of mandatory and optional tasks, an off-line algorithm aims to derive a static schedule that minimises some particular error metric
  - Can be executed during design, with hard coded schedule
    - Undesirable, system during design the system can be engineered to meet all deadlines
  - Can be executed at run-time, as a result of a mode change that causes more tasks to run
    - Off-line algorithms typically heavy-weight, so this is a rare event on a significant reconfiguration of the system
- Generally reduces to a linear programming or constraint optimisation problem
- Is NP-hard, and unrealistic, for real-world error functions
  - 0/1 constraints
  - non-linear error functions

Scheduling Flexible Applications

- All useful scheduling algorithms for flexible applications use heuristics or are otherwise imprecise
- Two general approaches: mandatory first and slack stealing
- Mandatory first algorithms schedule the mandatory parts of the system with higher priority than the optional parts
  - Use a fixed priority algorithm, like rate monotonic, to schedule the mandatory parts
  - Then schedule optional parts to minimise error:
    - dynamic least-attained-time suitable if error functions are convex, since diminishing returns for tasks that have attained most time
    - dynamic best-incremental-return suitable if knowledge of error functions, since run the task which will most reduce the error
  - If don't know error functions (common case):
    - Rate monotonic or earliest deadline schedule of optional parts
    - Earliest deadline always achieves zero average error, if possible
- Slack stealing run optional tasks in slack time of mandatory tasks, dynamically according to EDF

Implementation Strategies

- Flexible applications are only useful if a system can be overloaded
- Typically only useful on soft real time systems, generally running on a general purpose operating system
  - Otherwise, engineer the system to avoid overload
- Implication: don't have good scheduling support
  - Given knowledge of current time and deadline, application will decide to shed work
    - sieve, incremental with milestones, alternate algorithm
  - Very much heuristic driven, rather than explicitly scheduled
  - Inherently imprecise, and difficult to reason about

- If you're building these systems:
  - program defensively
  - measure behaviour
  - adapt accordingly, based on domain specific heuristics and error functions

Summary

By now, you should know…

- Scheduling tasks with temporal distance constraints
- Issues to consider when running on a general purpose system, and when designing a flexible application