Reading Assignment #1

Marcella Wilson

“Bounding Completion Times of Jobs with Arbitrary Release Times and Variable Execution Times”

Jun Sun and Jane W. S. Liu

Three Algorithms

- Algorithm ERT (effective response times)
- Algorithm CJA (critical job)
- Algorithm ITR (iteratively applies Algorithm CJA)
  - Completion time bound, complexity

Why is bounding completion time so important?

- A good way to validate system timing constraints

- Guarantee responsiveness of the system

- The job can complete on time

What's original in this paper?

- Ha's work studies validation problem in distributed system
  - System proven to be predictable
- This paper studies validation problem in single processor
  - System proven to be unpredictable
- Provides tighter bounds on algorithms than Ha's algorithms

Unpredictable System

- A job chain can have various schedules
- Difficult to find exact worst-case for job chain
- All jobs can have max **execution** time, but all jobs may not have worst-case **completion** times – unpredictable!
- Focus - finding upper bounds of job completion times

Assumptions

- Job chain – set of jobs, a job cannot execute until the job before it completes

- Independent chain
  - 1st job in chain has no predecessors
  - No precedence constraints between 2 jobs in different chains

- Each job has a fixed priority, preemptible

Assumptions

- Release time, $r_{i,j}$
  - occurs immediately when its predecessor completes
- Ready time, $y_{i,j}$
  - when job released or predecessor completes (whatever is later)
- Completion time, $c_{i,j}$
- Execution time, $[e^{+}_{i,j}, e\text{-}_{i,j}]$
  - range, actual execution time unknown

Assumptions

- Response time = completion time - release time
  - Interval ($r_{i,j}$, $c_{i,j}$)

- Effective response time = completion time – ready time
  - Interval ($y_{i,j}$, $c_{i,j}$)

Algorithm ERT

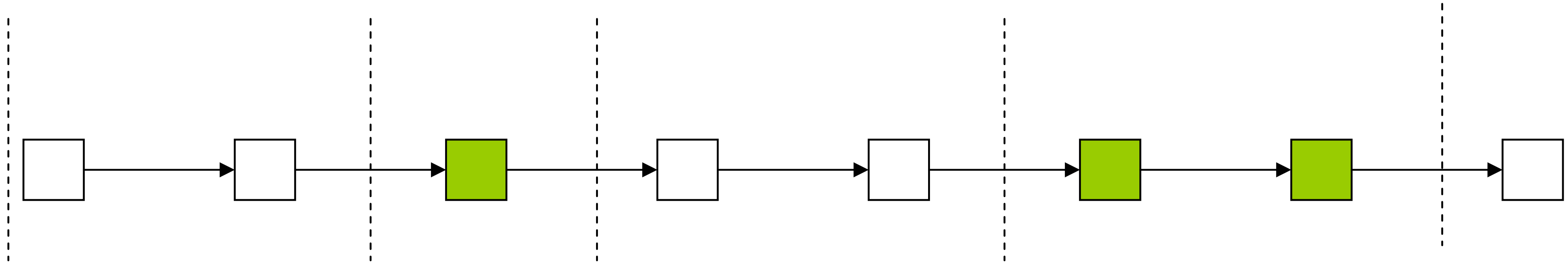
- What does it do?
  - Bounds the effective response time first
  - Then derives a completion time bound (based on the effective response time

Algorithm ERT

- 2 Job Chains: J_i and J_k
- Target job, $J_{i,j}$, in job chain J_i
- Jobs that can execute during J_i interval ($y_{i,j}$, $c_{i,j}$)
  - Jobs have higher priority that $J_{i,j}$
  - Jobs in another job chain

Algorithm ERT (cont)

- J_k divided into subchains called interference blocks
- J_k has m_k interference blocks
  - Shaded-jobs with priority lower than $J_{i,j}$
  - White – jobs with priority equal to or higher than $J_{i,j}$



Algorithm ERT (cont)

- Only 1 interference block can execute in interval

- Now we can bound the execution time of all the jobs in J_k that can delay the completion time of $J_{i,j}$

- First, find max time J_k can delay $J_{i,j}$
  - $M_{k,l}$ = sum of max execution times of jobs in *l*th interference block of J_k
  - Max amount of time that $J_{i,j}$ can be delayed by jobs in J_k is never more than max of $M_{k,l}$

Algorithm ERT (cont)

- But this shows how one job chain J_k is effecting target job $J_{i,j}$
- We can bound the execution time of all the jobs in all the job chains that can delay the completion time of $J_{i,j}$

Algorithm ERT (cont)

- Max delay all jobs chains (except J_i) can delay $J_{i,j}$
  - Give max total execution time of all jobs (except $J_{i,j}$) that can execute in interval
  - Algorithm Interference computes *inter*($J_{i,j}$, **J**)
  - Max total delay that $J_{i,j}$ might suffer

Algorithm Interference

1. Inter = 0
2. For every job chain J_k ($k \neq i$)
   1. Identify interference blocks in J_k
   2. Compute $M_{k,l}$, sum of the max execution times of jobs in the *l*th interference block in J_k
   3. Inter = inter + $\max_{1 \leq l \leq mk}\{M_{k,I}\}$
3. Return inter

How to bound completion times

- *inter*($J_{i,j}$, **J**) allows interval ($y_{i,j}$, $c_{i,j}$) – effective response time – to be bounded
- Max delay each job can suffer
  - If first job in chain
    - $C_{i,1} = r_{i,1} + e^+_{i,1} + inter(J_{i,j}, \mathbf{J})$
  - If not first job in chain
    - $C_{i,j} = max\{C_{i,j-1}\ r_{i,j}\} + e^+_{i,1} + inter(J_{i,j}, \mathbf{J})$
- Use equations to find completion time bound for each job

Final Thoughts on Algorithm ERT

- Runs in $O(N^2)$ time
- In bounding completion time of a job, the same delay may be counted twice.
- This problem is remedied in Algorithm CJA

Algorithm CJA

- Focuses on job chain/subchain instead of just the target job $J_{i,j.}$
- Assumes worst-case schedule
- Completion time using worst-case schedule
  - $r_{i,k}$ + ($r_{i,k}$, $c_{i,5}$)
  - Interval ($r_{i,k}$, $c_{i,5)}$

Algorithm CJA

- **Critical job** of each target job, $J_{i,j}$
  - Last job in J_i whose ready time equals its release time
- **Critical interval** $[r_{i,c(j)}, c_{i,j}]$
  - when critical job is released minus when target job completes
- Bounding duration of critical interval gives tighter bound on completion time of $J_{i,j}$.
- Jobs that can execute in the critical interval
  - Not in job chain J_i
  - have priorities greater than or equal to $\mathbf{J_{i,low}}$

How is the completion time bounded?

- Assumes each predecessor of $J_{i,j}$ and $J_{i,j}$ itself is critical job.
- Find the lowest priority among jobs.
- Compute $b_{i,k} = r_{i,k} + \sum e^{+}_{i,l} + inter(J_{i,low}, J)$
- Takes the max of the bounds, one of these bounds must be correct.

Algorithm CJA Example

- Use Algorithm CJA to bound completion time of $J_{i,3.}$
  - Let $J_{i,1}$ be the critical job. Find job with the lowest priority ($J_{i,1}$). Apply equation = 160.
  - Let $J_{i,2}$ be the critical job. Find job with the lowest priority ($J_{i,3}$). Apply equation = 140.
  - Let $J_{i,3}$ be the critical job. Find job with the lowest priority ($J_{i,3}$). Apply equation = 185.
  - Final bound is max of {160, 140, 185} = 185.

Final Thoughts on Algorithm CJA

- Each bound of Algorithm CJA is always tighter than the corresponding one computed by Algorithm ERT

- Higher complexity - $O(N^3)$

Algorithm ITR

- Previous algorithms flaw - Release time of jobs not taken into account.
- Result – Jobs whose release time is later than the target job’s completion time are considered.
- Solution - Don’t consider jobs that cannot interfere with execution of target job.

Algorithm ITR

- Remove jobs that do not execute in the critical interval of target job.

- Use Algorithm Interference on to obtain tighter bound on max delays target job may suffer.

- Two approaches
  - Pessimistic iteration
  - Optimistic iteration

Pessimistic Iteration

- First, use Algorithm CJA to get initial completion time bound for each job
- Then, iteratively apply modified Algorithm CJA to get new completion time bounds
  - Removes jobs that don’t execute in critical interval of target job
- Iteration stops new bounds in current step equal bounds in previous step

Problem with Pessimistic Iteration

- Sometimes, jobs that should be pruned, are not.
- Generally, but doesn't always, improves the completion time bounds

Optimistic Iteration

- First, gets an optimistic bound
  - Assumes each job interfered only by jobs in same job chain.
- Then, iteratively apply modified Algorithm CJA to get new completion time bounds
  - Based on bounds from previous or initial step
- Iteration stops when new bounds = corresponding bounds in earlier step

Algorithm ITR

- Has the tightest bounds of all the algorithms
- Worst time complexity - $O(N^6)$
- Works best for off-line schedulability analysis

Thank you for your attention.
Any questions?