

Statement Of the problem v0.4

03/14/2006

- 1 Object
- 2 Tasks and Jobs
- 3 Two-level Scheduling Model
- 4 Global Scheduling algorithms
- 5 Local Scheduling algorithms
- 6 Restrictions

1 Object

Different strategies can be applied in R-EDF such as First Fit, Last Fit, Best Fit and Worst Fit; we can also schedule background jobs in many ways. Different scheduling methods will give us very different response time for background jobs depending on the strategy used. Meanwhile, since this is a real-time system, we need to make sure the system will not miss any real-time deadline.

So, the object of this project is to find out the best strategy which gives us the shortest response time of background jobs without missing any real-time deadline.

2 Tasks and Jobs

Real-time Task

A real-time Task can be described as (p, e) :

p: period

e: execution requirement

Real-time job

Each real-time job can be described as (a, e, d) :

a: arrival time

e: execution requirement (the worst-case estimation)

d: deadline

Jobs of the same real-time task have the same period and worst-case execution requirement, but since e is a worst-case estimation, the actual execution time could be less or equal to this estimation. Most of the time, the actual execution time will be less than this estimation, and the difference is the dynamic slack

Background job

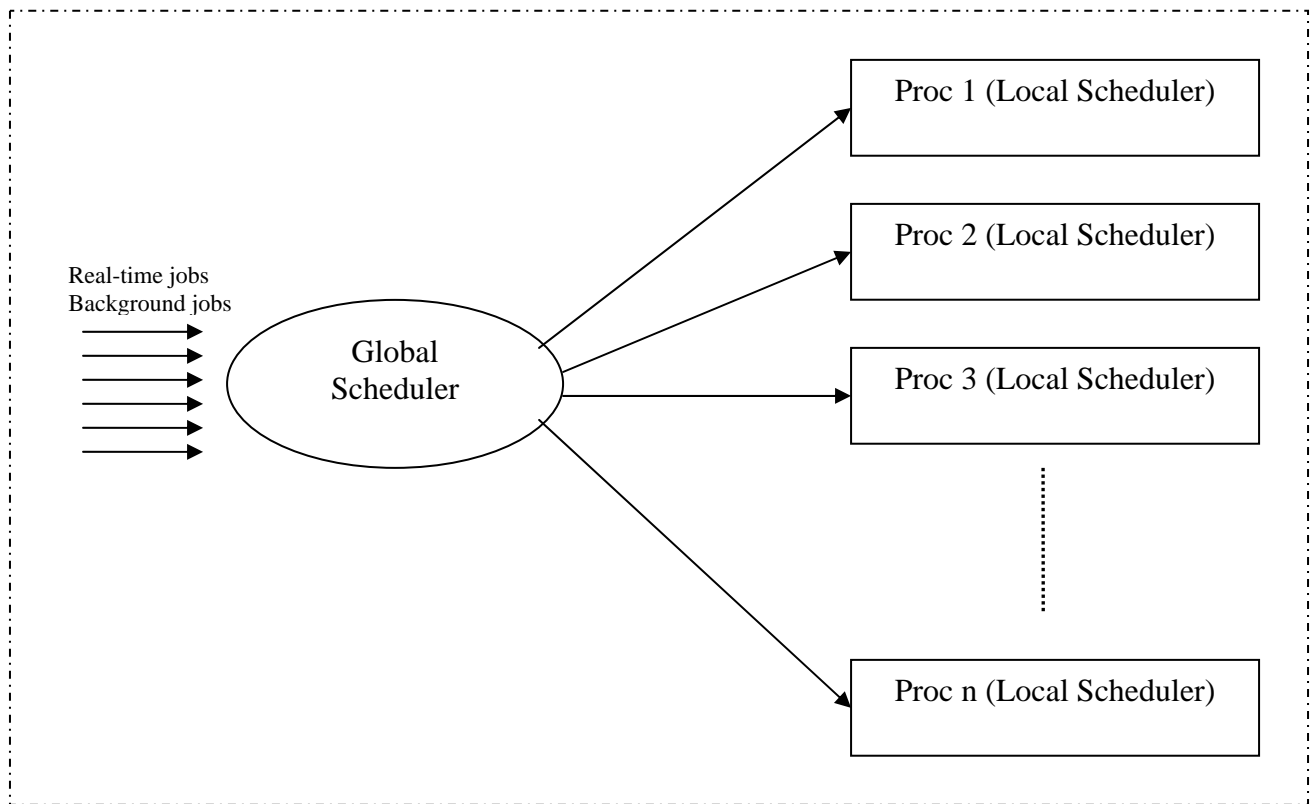
Each background job can be described as (a, e) :

a: arrival time

e: execution requirement (the average-case estimation)

Background jobs have no deadlines. Since e is the average-case estimation of execution requirement, the actual execution requirement could be less, equal or more than this estimation.

3 Two-level Scheduling Model



Two-level scheduling model

Level 1: Global Scheduler

Real-time Tasks generate real-time jobs, these jobs and background jobs will be assigned to a particular processor based on different strategies.

Level 2: Local Scheduler

The local scheduler will schedule real-time jobs and background jobs, it should try to minimize the response time of background jobs and guarantee no real-time deadline will be missed.

Processors are uniform multiple processors, which may have different speeds.

4 Global scheduling Algorithms

For Real-time Jobs:

The global scheduler will schedule jobs using R-EDF with First Fit, Last Fit, Best Fit and Worst Fit.

First Fit

For each new arrived real-time job, choose the fastest processor whose dynamic slack is bigger than the utility of this new job

Last Fit

For each new arrived real-time job, choose the slowest processor whose dynamic slack is bigger than the utility of this new job

Best Fit

For each new arrived real-time job, choose the processor which has the least dynamic slack, which is required to be bigger than the utility of this new job

Worst Fit

For each new arrived real-time job, choose the processor which has the biggest dynamic slack, which is required to be bigger than the utility of this new job

Utility of a real-time job

We know for any real-time task, the utility of this task is the *execution requirement* divided by its *period*, but how to define the utility of a real-time job?

Real-time job utilization = corresponding real-time task utility

Dynamic slack of a processor

For each processor, in any particular time t , we have an associated unfinished real-time job set.
Processor Speed – the Summation of *Utilization* of all unfinished real-time jobs.

How to Computer the dynamic utilization of a processor:

- (1) Initial value of total utilization is set to be 0
- (2) Increase the total utilization by the utilization of the new real-time job when it arrives
- (3) Decrease the total utilization by the utilization of a real-time job when its deadline comes
- (4) Set the total utilization of the processor to be 0 when the real-time job queue is null
- (5) Do not decrease the total utilization when the job queue is empty even if the deadline comes

For Background Jobs:

There are two options

- (1) Select the Processor with the largest slack at arrival time
- (2) Select the Processor based on execution requirement, like background jobs with large execution requirements will be assigned to fast processors, and ones with small requirements will be assigned to slow processors

5 Local Scheduling Algorithms

5.1 How to simulate the actual running situation

We need to simulate the actual running situation of each real-time jobs and background jobs.

For Real-time jobs:

Suppose the worst-case estimation of execution requirement is E_{qe} .

We will generate a random number $RAND$ which falls with the range $(0, 1]$ with different means like 0.8, 0.5 and 0.2, then the actual execution requirement is $RAND * E_{qe}$

For Background jobs:

Suppose the average-case estimation of execution requirement is E_{qe} .

We will generate a random number $RAND$ which falls with the range $(0, 1.2]$ with different means like 1.0, 0.8 and 0.5, then the actual execution requirement is $RAND * E_{qe}$

5.2 Local scheduling algorithm

Real-time jobs: EDF

Background Jobs: When the processor is idle, we schedule background jobs, and we choose background job based on first come, first served.

6 Restrictions

No migration of Real-time jobs

Real-time tasks can be assigned to different processors depending on the global scheduler, but once a real-time job is assigned, it can not migrate to other processors.

No Migration of Background jobs is allowed

When a background job arrives, it will be assigned to a processor by the global scheduler, during the lifecycle of the background job, it will be allowed to migrate to other processors.