Predictive scheduling of job combinations in SDR systems

David Guevorkian

Tampere University of Technology, Finland

and

Jan Westmeijer

mimoOn GmbH, Duisburg, Germany

Presented by Jarmo Takala

Tampere University of Technology, Finland

Outline

✓ Introduction

✓ SDR as a piece-wise stationary application

✓ Timing policies in scheduler design

✓ Predictive scheduling approach

✓ Analysis

✓ Conclusion

Introduction

- In SDR systems, different combinations of radios must be implemented on top of a shared computational platform.
- Each radio (job) consists of a number of algorithms (tasks) having strict real-time constraints.
- Computational platform typically consists of several types of processing elements and HW accelerators.
- Efficient scheduler design is essential for the SDR system.
- Our target is not only schedulability of different radio combinations but also efficiency in terms of HW and power utilization

SDR: a piece-wise stationary application

- There are relatively small number of jobs (radios) to support in an SDR system.
- Each job is more or less stationary with approximately fixed sequence of tasks (algorithms) with predictable wors-case execution times.
- However, the overall system is far not stationary due to unpredictable sequence of switching radios On/Off.
- We shall call “piece-wise stationary” those applications where several stationary jobs may be initiated or terminated at arbitrary time instances.
- Relatively long periods of static states but with unpredictable changes between these states.

SDR: other specific features

- Because radios should run smoothly when radio combinations are changed, different schedules for the same job combination may be the most suitable depending on the histoty of job combinations:
  - Off-line (complile time) pre-design of schedules for all possible job combinations is infeasible.
- Execution times of radio algorithms (tasks) are very short:
  - Loading times of the programs for tasks are comparable to their execution times.
- Real-time constraints are very hard and strict:
  - In run-time, there is really short time to create schedules and optimize them in the sense of HW or power utilization;

Timing policies in scheduler design

- Schedule creation implies three main steps:
  - processor assignment where a decision is made on allocating tasks to processors or hardware accelerators;
  - decision on the order of execution of tasks on each processor or hardware accelerator;
  - decision on the firing (or, equivalently, execution) times of each actor (task).
- Each of these steps can be implemented either on compile time (static approach) or on run-time (dynamic approach):
- We call the decision on when to implement each of these steps, the timing policy of scheduler design.

Conventional timing policies

Fully dynamic: run-time decision on processor assignment, task ordering and task firing times.

Static assignment: compile time decision on processor assignment; run-time decision on task ordering and on firing instances.

Quasi-static: compile time decision on processor assignment and partially on task ordering and firing instances; partially run-time decision on task ordering and firing instances (for data dependent tasks)

Self-timed: compile time decision on processor allocation (but not on processor communication) and on task ordering; run-time decision on task firing instances.

Ordered transaction: compile time decision on processor assignment, and processor communication as well as on task ordering; run-time decision on task firing instances.

Fully static: compile time decision on processor assignment, task ordering and task firing times.

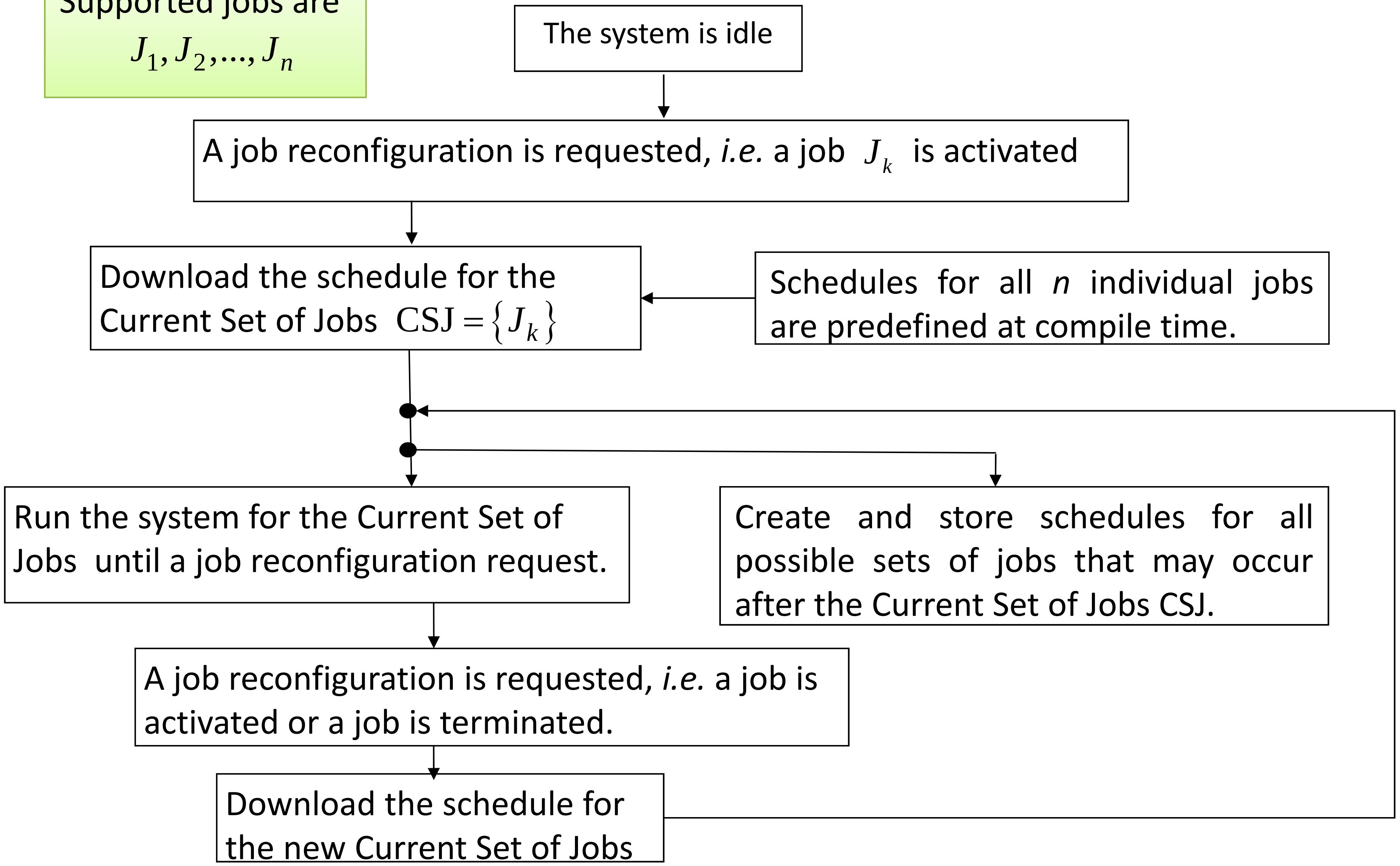
Predictive scheduling

- In partially dynamic scheduling approach, the schedules are created at reconfiguration time according to the change in the set of currently running radios and according to previous state.
- Real-time constraint significantly limits such approach:
  - Even if a valid schedule may be created in an acceptable time, no sophisticated optimization technique may be applied.
- One possibility is to pre-create all schedules for all changes in radio combinations. In an SDR system with *M* radios *2^(2^M)* schedules must be created and stored. Obviously infeasible.
- The idea of predictive scheduling approach is to create and store all possible radio combinations that may arise from current set of running radios by switch on/off of a single radio.
- This allows of optimizing the schedules until the change (switch-on/off) occurs;
- Once the change occurs, the corresponding schedule is loaded and schedules for all potential radio combinations are created.
- In an SDR system with *M* radios, *M* (instead of *2^(2^M)*) schedules have to be created each time

J_k

Predictive scheduling

Supported jobs are $J_1, J_2, \ldots, J_n$



The system is idle

A job reconfiguration is requested, *i.e.* a job J_k is activated

Download the schedule for the Current Set of Jobs $\mathrm{CSJ} = \{J_k\}$

Schedules for all *n* individual jobs are predefined at compile time.

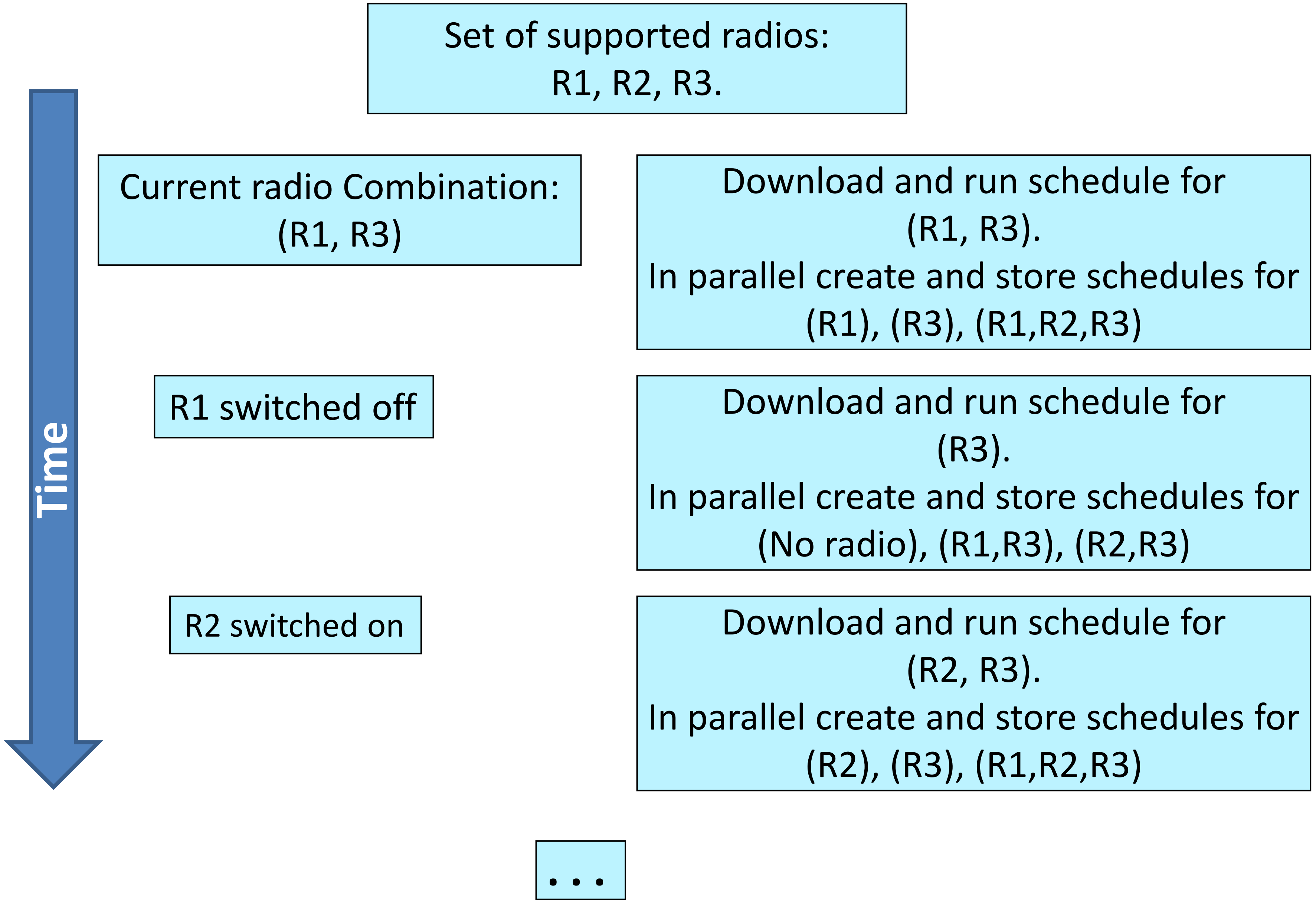
Run the system for the Current Set of Jobs until a job reconfiguration request.

Create and store schedules for all possible sets of jobs that may occur after the Current Set of Jobs CSJ.

A job reconfiguration is requested, *i.e.* a job is activated or a job is terminated.

Download the schedule for the new Current Set of Jobs

Example



Set of supported radios:
R1, R2, R3.

Current radio Combination:
(R1, R3)

Download and run schedule for
(R1, R3).
In parallel create and store schedules for
(R1), (R3), (R1,R2,R3)

R1 switched off

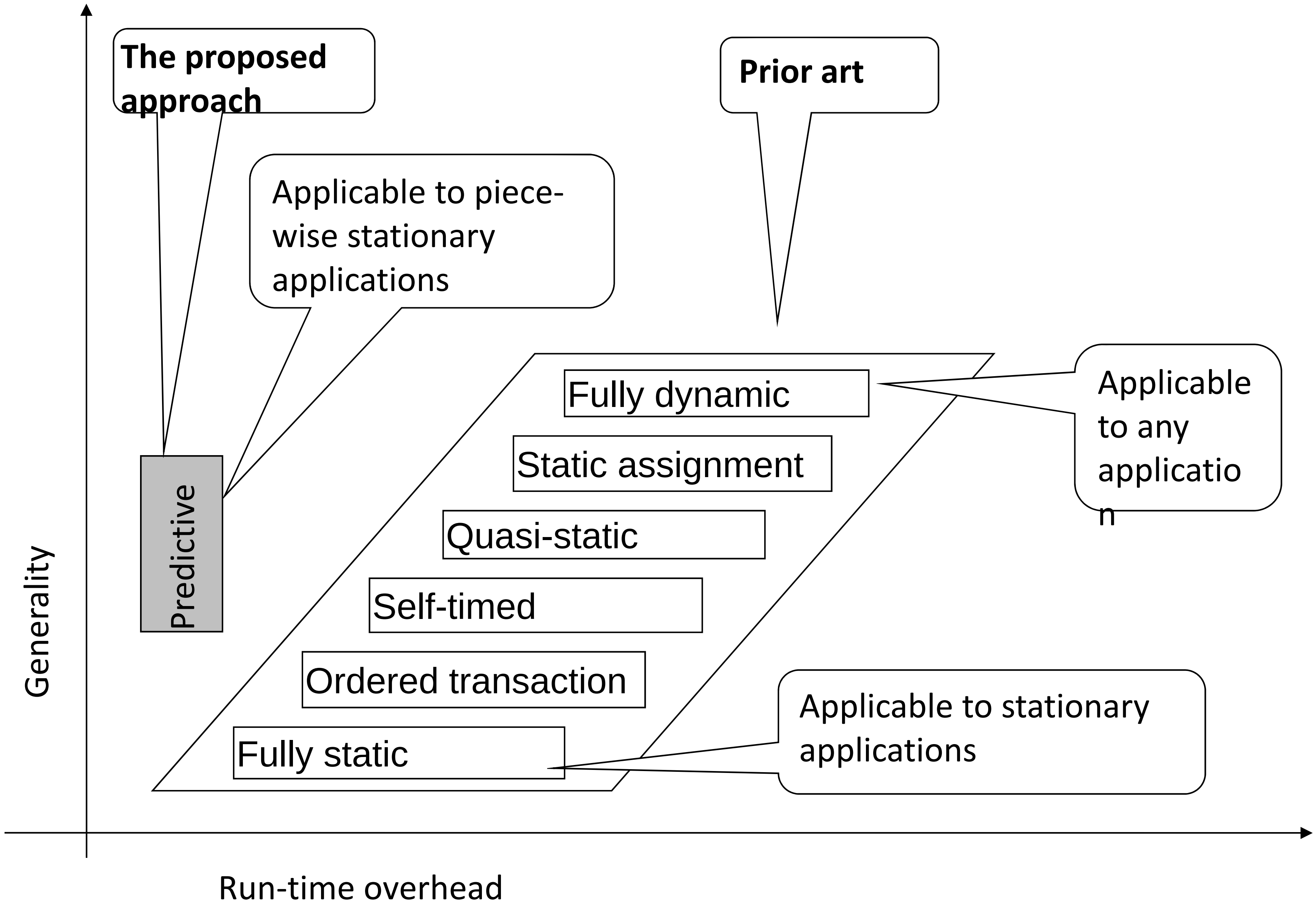
Download and run schedule for
(R3).
In parallel create and store schedules for
(No radio), (R1,R3), (R2,R3)

R2 switched on

Download and run schedule for
(R2, R3).
In parallel create and store schedules for
(R2), (R3), (R1,R2,R3)

. . .

Analysis


The proposed approach
Prior art
Applicable to piece-wise stationary applications
Fully dynamic
Static assignment
Quasi-static
Self-timed
Ordered transaction
Fully static
Predictive
Applicable to any application
Applicable to stationary applications
Generality
Run-time overhead

Conclusion

- ✓ New timing policy for schedule creation is proposed.
- ✓ Schedules are dynamically created every time when system's steady state is changed.
- ✓ The proposed method suits well to scheduler design for SDR systems because it requires minimal overhead and allows dynamic creation of optimized schedules with respect to the history of the system evolution.
- ✓ The proposed method considers only the timing policy.
- ✓ It may be combined to any actual scheduling rule such as fixed-priority or Round Robin scheduling rules.
- ✓ To evaluate the impact of the proposed method quantitatively, several actual scheduling policies could be implemented in combination with the proposed timing policy and experimented.