SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms

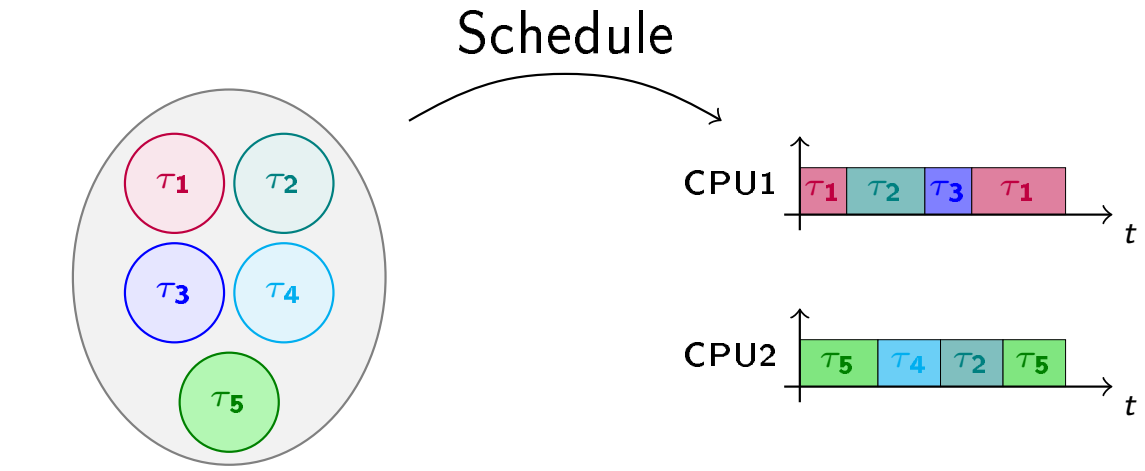
Maxime Chéramy, Pierre-Emmanuel Hladik and Anne-Marie Déplanche

maxime.cheramy@laas.fr

July 8, 2014

5th International Workshop on Analysis Tools
and Methodologies for Embedded and Real-time Systems (WATERS)

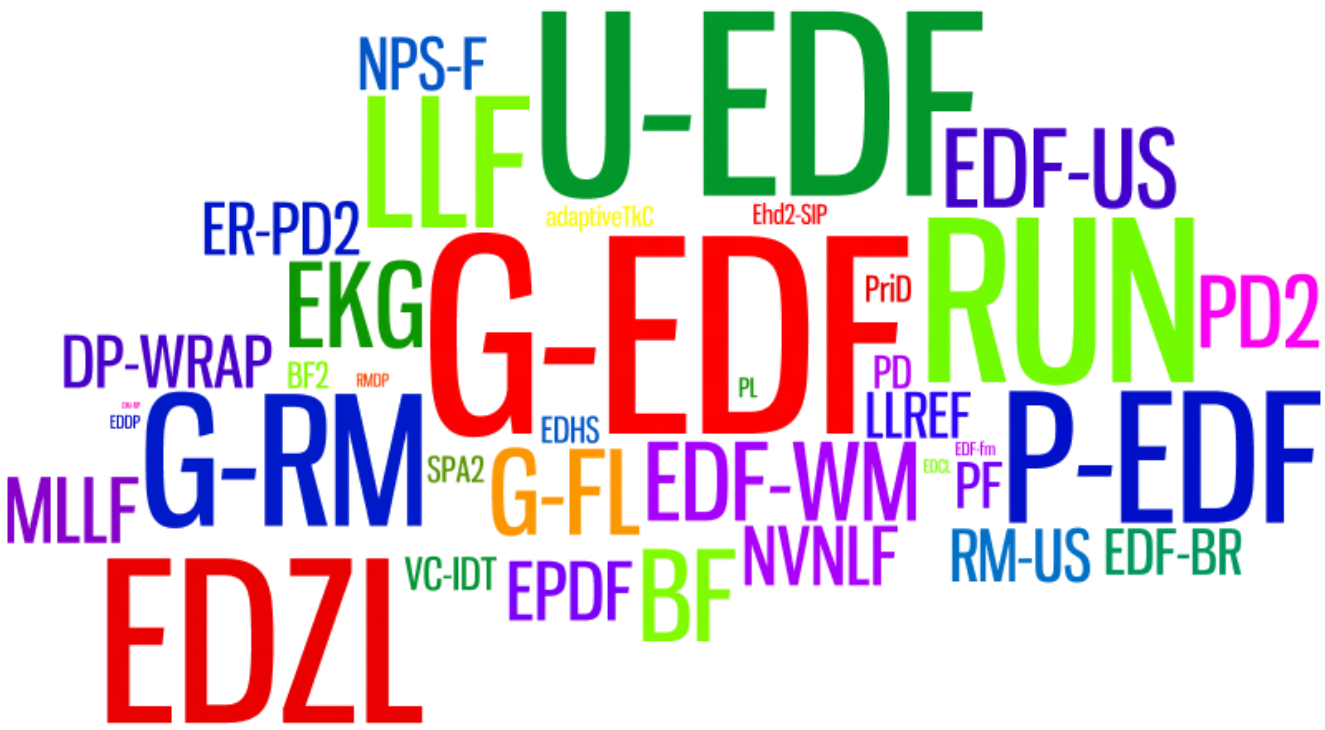
Problem Definition



Independent and periodic tasks

Multiprocessor Architecture

Many Real-Time Multiprocessor Scheduling Algorithms



Objective

To evaluate their behavior and performance.

Several Approaches

Theoretical analysis

Good way to prove schedulability, but hard to take into consideration some aspects

Empirical studies on real systems

Realistic. But require a significant amount of time and skills to handle the tools and the results are specific to a certain platform

Simulations

Easy to use and very flexible but less accurate than a real system

SimSo : A Simulator to Evaluate Scheduling Algorithms

SimSo[1]

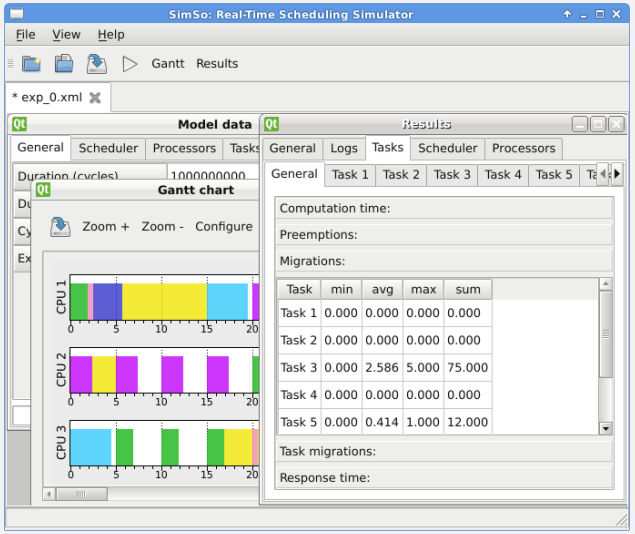
- Simulator dedicated to the study of new scheduling algorithms.
- Designed to be easy to use, and also easy to extend for new needs.
- Open Source : http://homepages.laas.fr/mcheramy/simso/

[1] Simulation of Multiprocessor Scheduling with Overheads

Two ways of using SimSo

Using the graphical user interface

- Configure a new system
- Save and load configurations (in an XML file)
- Simulate a configuration
- Display the resulting schedule as a Gantt chart
- Access usual metrics



Easy to use

Good way to debug an algorithm

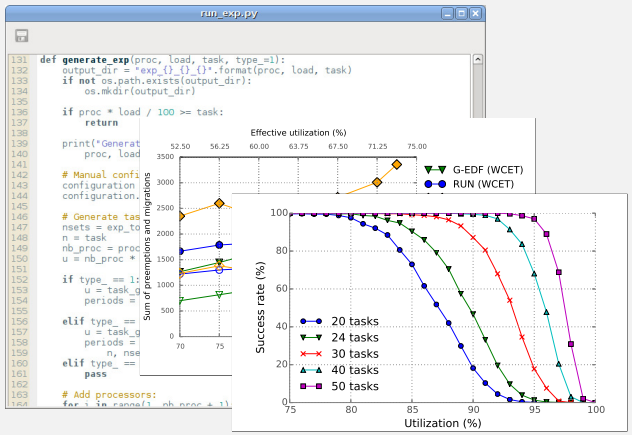
Impossible to automate multiple simulations

May not give you the metric you are looking for

Two ways of using SimSo

Using it as a Python Module

- Generate a set of systems and launch their simulations from a script
- Collect raw data
- Use predefined methods to retrieve common metrics
- ...or develop your methods to compute what you need.



Very flexible

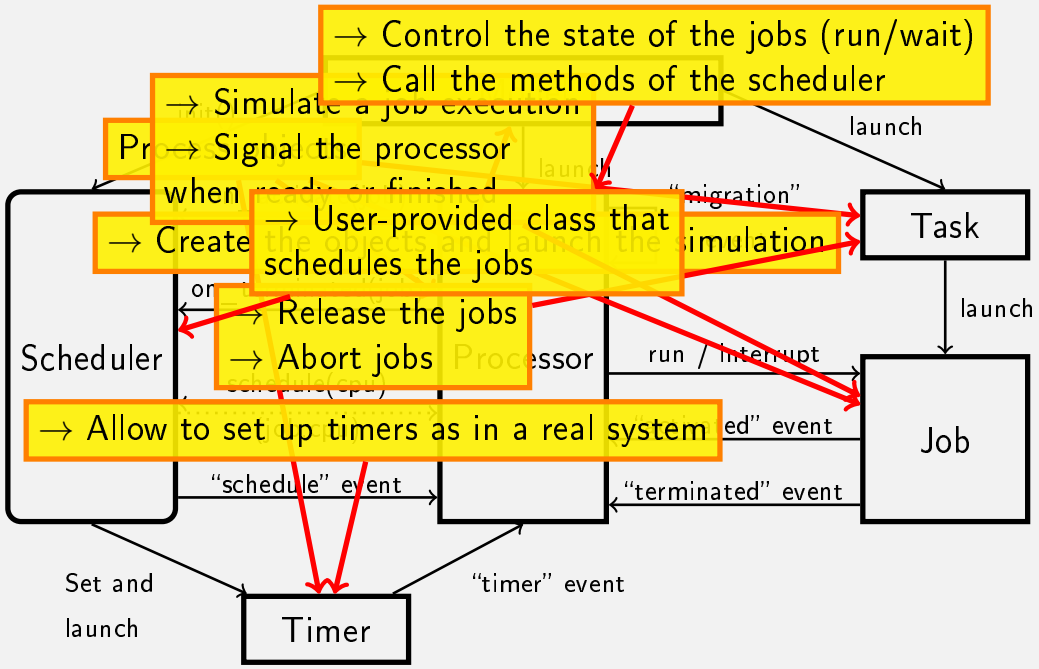
Require to develop and learn the programming interfaces

Software Architecture

Discrete-Event Simulation

SimSo is based on SimPy, a process-based discrete-event simulator.

Main classes



Software Architecture

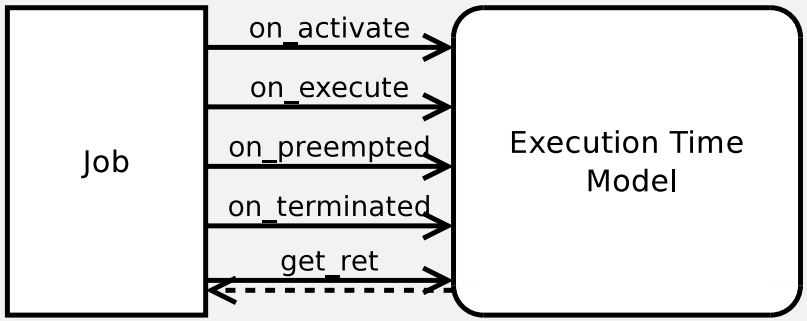
Simulation of the computation time of the jobs

SimSo offers multiple ways to simulate the computation time of a job:

- computation time of the jobs = their WCET
- random durations based on an ACET
- with fixed time penalty after a preemption and migration
- using cache models
- considering the speed of the processor (DVFS)

Software Architecture

Execution Time Model



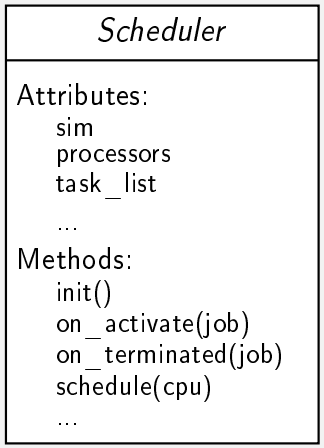
- Only one ETM for all the system.
- Each job informs the ETM of every change of state.
- The ETM determines the remaining execution time of the job.
- A job is terminated as soon as `get_ret` returns 0.

Writing a Scheduler

Interface

A scheduler for SimSo is a Python class that inherits from the abstract class *Scheduler*.

This class is loaded dynamically in SimSo and linked to the simulation core.



Example

Code for a Global EDF (one possible implementation).

```
from simso.core import Scheduler

class G_EDF(Scheduler):
    """Global Earliest Deadline First"""
    def init(self):
        self.ready_list = []

    def on_activate(self, job):
        self.ready_list.append(job)
        # Send a "schedule" event to the processor.
        job.cpu.resched()

    def on_terminated(self, job):
        # Send a "schedule" event to the processor.
        job.cpu.resched()

    def schedule(self, cpu):
        decision = None # No change.

        if self.ready_list:
            # Look for a free processor or the processor
            # running the job with the least priority.
            key = lambda x: (
              1 if not x.running else 0,
              x.running.absolute_deadline if x.running else 0)
            cpu_min = max(self.processors, key=key)

            # Obtain the job with the highest priority
            # within the ready list.
            job = min(self.ready_list,
                      key=lambda x: x.absolute_deadline)

            # If the selected job has a higher priority
            # than the one running on the selected cpu:
            if (cpu_min.running is None or
                  cpu_min.running.absolute_deadline >
                  job.absolute_deadline):
                self.ready_list.remove(job)
                if cpu_min.running:
                    self.ready_list.append(cpu_min.running)
                # Schedule job on cpu_min.
                decision = (job, cpu_min)

        return decision
```

Available Schedulers

Uniprocessor

RM, DM, FP, EDF, LLF, M-LLF, Static-EDF, CC-EDF

Partitioned

Any uniprocessor scheduler combined with various bin packing algorithms (Best Fit, First Fit, Next Fit, Worst Fit...)

Global

G-RM, G-EDF, G-FL, EDF-US, PriD, EDZL, LLF, M-LLF, U-EDF
PFair: PD^2, ER-PD^2
DP-Fair and BFair: LLREF, LRE-TL, DP-WRAP, BF, NVNLF

Semi-Partitioned

EDHS, EKG, RUN

Generation of Tasksets

Task utilization

The following generators are available:

- RandFixedSum
- UUniFast-Discard
- and other generators

Task periods

The periods can be generated using:

- Uniform distribution
- Log-Uniform distribution
- Discrete uniform distribution

Generation of Tasksets

Use of an external generator

- When SimSo is used from a Python script, it is possible to manually create the tasks with any characteristics.
- It is also possible to generate an XML file compatible with SimSo.

Simulation

Ready for simulation

When a system is fully configured (tasks, processors, simulation parameters, etc.), it can be simulated.

Speed

SimSo is able to run thousands simulations per hour (depending on the scheduler and system configuration).

Simulation output

Output

The output of the simulation is a trace of the events that occurred.

Example of events: start of a job (together with its processors), interruption, resumption, scheduling decision, etc.

Analysis

It is possible to parse the trace to retrieve any specific metric.

But, in order to ease the retrieval of usual metrics, some functions are provided.

Objective

To show how SimSo can be used to conduct an experiment on scheduling policies.

Example

Comparison of the number of preemptions and migrations in function of the number of tasks.

Studied Schedulers: G-EDF, NVNLF, EKG[2], RUN and U-EDF.

[2] with K=number of processors

Generation of the Configurations

Parameters

- Number of tasks: 20, 30, 40, 50, 60, 70, 80, 90, 100
- Number of processors: 2, 4, 8
- System utilization: 85%, 95%
- Periods: log-uniform distribution in [2, 100ms]
- Simulation on [0, 1000ms]
- Execution Time Model: ACET (average value=75% of WCET)

Simulations

For each configuration (tasks, processors, utilization), 20 tasksets are generated.

Simulation

Duration

5400 simulations executed in ∼2 hours.

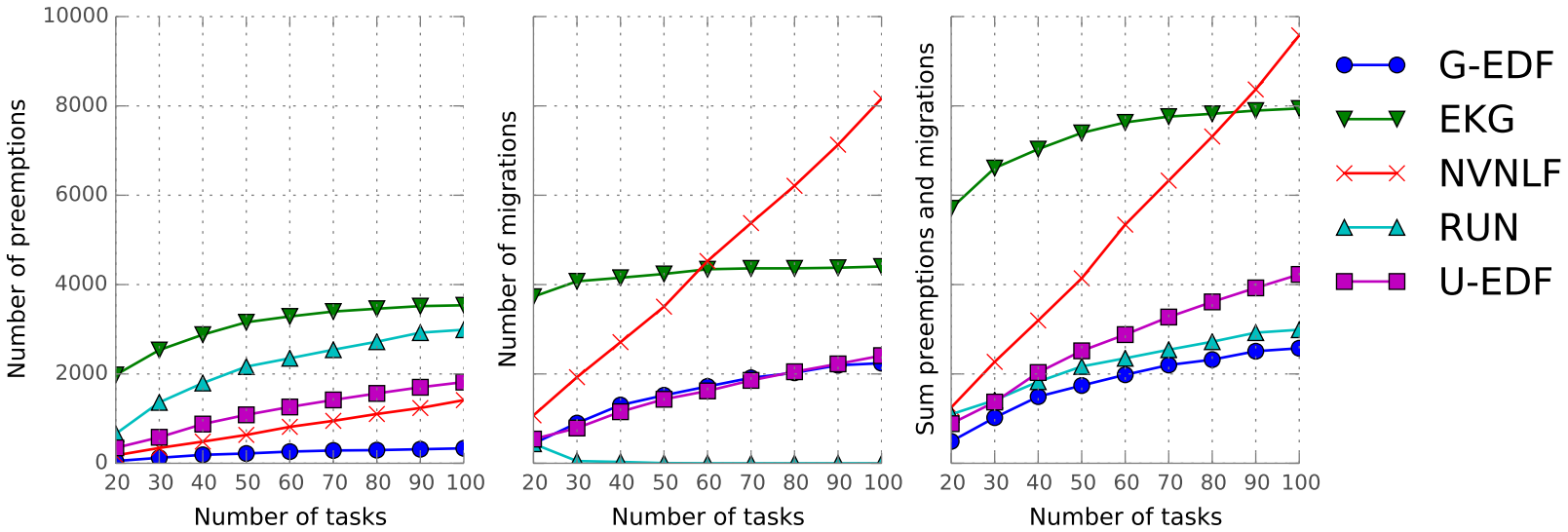
Collection of the results

The number of preemptions and migrations are extracted from each simulation and stored in an SQLite3 database.

Analysis

Results

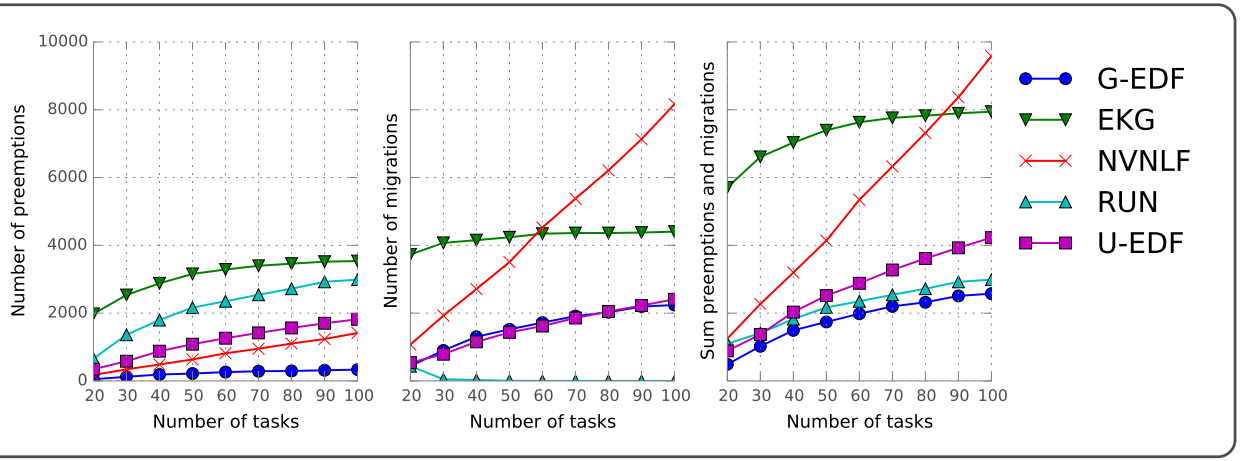
Preemptions and migrations depending on the number of tasks. Results for 8 processors and a total utilization of 95%:



Analysis

A few comments

- EKG generates a lot of migrations
- the results of NVNLF are getting better with more processors
- G-EDF gives the best results but a few jobs missed their deadline
- RUN acts as a partitioned scheduler with more than 30 tasks
- RUN and U-EDF results are good



Analysis

Possible improvements

- EKG could do less migrations with a better choice of K and with other improvements
- the implementation of RUN could benefit from minor improvements (see paper about RUN presented at ECRTS)
- U-EDF could probably do better with clustering or with some improvements (eg. by using the idle times)

Current and Future Work

Improvement in SimSo

Inclusion of other task models (eg. task dependencies, sporadic tasks).

Validation of the results regarding the use of cache models.

Experiments

SimSo was used to run approximatively 500 000 simulations.

We are currently analyzing the results. In particular, many existing experiments were reproduced.

Conclusion

SimSo

SimSo is a simulator dedicated to the evaluation of scheduling algorithms.

Thanks to its design, it is easy to extend it to new models.

Schedulers

- Implementing a scheduler is simplified
- More than 25 scheduling algorithms were implemented
- SimSo is capable of handling partitioned, global and hybrid scheduling approaches.

Questions?

Thank you for your attention.