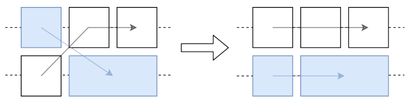
Entropy-based scheduling performance in real-time multiprocessor systems

Daniel E. Rivas S.*, Carlos A. Rincón C.*
*University of Houston, TX, Houston, USA
Email: derivassanchez@uh.edu, carincon@uh.edu

UNIVERSITY of HOUSTON | COMPUTER SCIENCE

Entropy-based scheduling

An entropy-based layer for task scheduling can be used for reducing the number of task migrations on multiprocessor systems. The idea is to reduce the complexity of the problem by selecting the best permutation of job-to-processor mappings.



Multiple scheduling algorithms

Since the entropy-based layer is non-intrusive, it can be easy to integrate into existing algorithms without too much effort. It works with all algorithms tested so far, including:

Earliest deadline first (EDF)

Least laxity first (LLF)

PFair-based algorithm (PD2)

Benchmark environment

All the benchmarks were done by comparing the original implementation of each scheduling algorithm against its their entropy-enabled implementation. In general, the original scheduling algorithm remains unchanged, as the entropy-layer is only executed when mapping pending jobs to processors. To get accurate results, we executed each scenario ten times (ten experiments) and assigned the average of all experiments as the result for that specific scenario.

The **dependent variables** in our experiments were the *number of preemptions*, *job migrations*, and *task migrations.* Our **independent variables** were the *number of processors*, and *utilization percentage*.

2, 4, 8
Number of CPUs

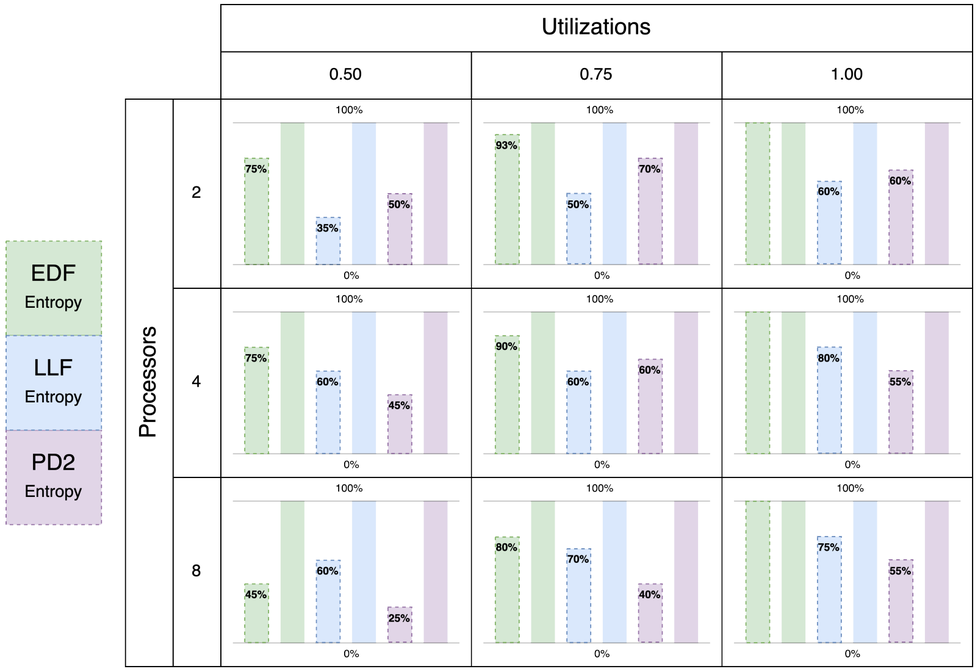
50%, 75%, 100%
Utilizations

Benchmark results

Each of the matrix plots in this section shows the relative amount of **task migrations** of each entropy-enabled implementation against its original counterpart.

How to read the charts

60% means that the entropy-enabled implementation showed only 60% of the task migrations observed in the original algorithm (meaning lower is better).



Bench.py command-line interface

All the benchmarks in this presentation were executed with the help of **bench.py** [1]. A command-line interface written in Python that integrates with SimSo [2] for efficient batch processing and execution of scheduling algorithms.

The tool itself is easy to integrate with your own analytics tools since its input-output format is based on *SQLite*. You can analyze the results using raw SQL queries or using more specialized tools like Excel.

python™

SimSo

SQLite

Result analysis

EDF vs EDF Entropy

In this scenario, we compared *Earliest deadline first* against its entropy-enabled implementation. With the latter executing the entropy layer each time a new job was to be scheduled.

The results on the left show that the number of *task migrations %* depend mostly on the CPU utilization, with the former showing improvement the less utilization we have. Which makes sense since the entropy layer will have more freedom to schedule the best possible permutation.

LLF vs LLF Entropy

In this scenario, we compared *Least laxity first* against its entropy-enabled implementation. In this case, the entropy layer is executed each time the scheduler is interrupted (each tick) to compute the laxities.

We see higher improvement in this scenario since the original *LLF* does not have any heuristic for selecting the processor, it just uses whatever is available.

PD2 vs PD2 Entropy

For this scenario we, compared *PD2* against its entropy-enabled implementation. The latter executes the entropy layer each time the scheduler is interrupted (each tick) to compute the virtual jobs.

We gain improvements here for the same reason as *LLF* since *PD2* does not use any heuristic for selecting a processor. However, the entropy-layer overhead was higher due to the scheduler being interrupted more often.

Conclusions

Most of the benchmarks presented here show improvement when comparing the entropy-enabled implementations against their original counterparts.

However, the entropy-layer currently selects the **best processor** to use for a given job by working through the possible mappings of said job and all processors. Future work will attempt to reduce the overhead of this greedy approach, finding a good compromise between computation time and the selected processor.

References

1 https://gitlab.com/uh-spring-2021/cosc-4396-senior-research-project
2 http://projects.laas.fr/simso/