Comparing Heuristics and Linear Programming Formulations for Scheduling of In-Tree Tasksets

Thomas Kothmayr, Jakob Hirscheider, Alfons Kemper
Chair for Database Systems,
Technische Universität München, Germany
{kothmayr, hirschei, kemper}@in.tum.de

Andreas Scholz, Jörg Heuer
Corporate Technology, Siemens AG
{andreas.as.scholz, joerg.heuer}@siemens.com

***Abstract*—Cheap but resource constrained platforms are poised to assume duties in next generation automation systems - the Internet of Things (IoT) is entering the real-time scene. For highly distributed systems like these, service oriented architectures (SOAs) are being increasingly adapted to raise the overall level of flexibility and adaptability. Our approach encompasses a SOA for hard real-time tasks in industrial automation, aimed at IoT class devices. The feasibility of task assignments to machines is verified through computation of a local schedule for the tasks assigned to each device. This reintroduces the need for efficient non-preemptive single machine scheduling. In this paper, we evaluate the efficiency of five heuristics and two linear program formulations for scheduling task sets with release times, deadlines and in-tree precedence constraints.**

I. Introduction

Our vision takes the SOA approach to automation [1] and aims to make resource constrained IoT class devices full members of a hard real-time SOA. We chose a top-down approach to this problem, focusing on the planing and data-flow aspects instead of ontologies and network protocols. In our system, automation tasks are defined as cyclic workflows of distinct subtasks with precedence constraints and end-to-end deadlines. Individual tasks are assigned to machines in a real-time network by an engineer or an automatic planning component. Local constraints (task release times and deadlines) are derived from the underlying network configuration, dependencies of the tasks and the global end-to-end deadline. The feasibility of the assignment is verified by scheduling the tasks on each machine according to their new local constraints. At this early stage, our current work focused on efficient local scheduling as a means to verify a manual task assignment.

II. Formal Problem Description

The scheduling problem analyzed in this paper is as follows: Find a feasible schedule for a set of jobs $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ with a fixed integer processing time $wcet_i$ on a single, non-preemptive machine. Additional constraints are: Each job j_i has either exactly one successor j_k, written as $j_i \prec j_k$, or no successor. A job may be annotated with a deadline d_i and a release time time r_i. Since the generated schedule is expected to be executed cyclically, a maximum schedule length $\mathcal{D}$ is defined. It is enforced by the root element Ω with $wcet_\Omega = 0, d_\Omega = \mathcal{D}$ and $r_\Omega = \mathcal{D}$. If $\emptyset \prec j$ set $\Omega \prec j$, thus transforming the structure of $\mathcal{J} \cap \Omega$ into a single in-tree. If no values for r_i or d_i are given, 0 and $\mathcal{D}$ are assigned by default. The objective is minimizing the number of tardy jobs ΣU_j. We are only interested in finding a schedule that has no tardy jobs at all. In the traditional notation of scheduling theory we can express our problem as $1|r_j, in-tree|\Sigma U_j$.

III. Related Work

Chretienne [2] minimizes the makespan of in-tree task sets with stochastic processing times in a distributed memory environment under the assumption that the processing times are greater than or equal to the data communication times between the jobs. Liu [3] applies heuristics in a branch-and-bound approach to minimize maximum lateness under general precedence constraints and release dates. We use his BLOCK heuristic on our problem set. For minimizing the maximum tardiness, research based on Schrage's Heuristic has proven to be very efficient. Hall and Shmoys [4] improved on Pott's original work which employed Schrage's Heuristic.

IV. Linear Programming Solutions

This paper is using the mixed integer programming (MIP) formulations given by Keha et al. [5]. They compare four different approaches: Start time and completion time variables (F1), time index variables (F2), linear ordering variables (F3), and positional and assignment variables (F4). Because we are looking for feasible instead of optimal solutions we only adapted F1 and F4. These generate the highest amount of feasible solutions in a given amount of time [5]. Due to space constraints, we refer the reader to the original paper for the MIP formulations.

V. Heuristic Approaches

Heuristics pose an attractive alternative for finding feasible solutions. We compare earliest release time time first (ERF), earliest deadline first (EDF), the BLOCK heuristic [3] and Potts' algorithm [4].

Earliest release time time first: The ERF heuristic simply sorts all jobs in $\mathcal{J} \cup \Omega$ by ascending order of their release time time. If two jobs have the same release time, then their deadlines are used as a tiebreaker. Precedence constraints should be mapped to modified release times by applying: $\forall j_i, j_k \epsilon \mathcal{J} \cup \Omega : j_i \prec j_k \implies r'_k = max\{r_k, r_i + wcet_i\}$

Earliest deadline first: In contrast to the ERF heuristic we cannot simply sort jobs by their deadline because that could lead to violation of precedence constraints. EDF instead chooses the leaf of the tree with the earliest deadline j_d and the leaf with the earliest release time j_r. If j_r can be scheduled

before j_d without conflict, i.e. $r_r + wcet_r \leq r_d$, schedule j_r first, otherwise j_d. The scheduled leaf is then removed from the tree and if all predecessors of a job have been scheduled that job is then added to the set of available leaves. Effective deadlines for each job should be computed beforehand as: $\forall j_i, j_k \epsilon \mathcal{J} \cup \Omega : j_i \prec j_k \implies d'_i = min\{d_i, d_k - wcet_k\}$

BLOCK heuristic: The BLOCK heuristic [3] first sets up a schedule by ERF and divides it into blocks of jobs which are executed with no time delay between them. If the schedule is invalid, the heuristic adjusts the block by scheduling jobs with higher deadline towards the end of the block.

Potts' algorithm: Potts' algorithm [4] is setup by sorting tasks by their deadlines in topological order. If the schedule is invalid, it analyzes the critical sequences of the schedule, i.e. blocks of jobs where at least one job has an invalid deadline (= critical job j_{crit}). An interference job j_{int} is a job within a critical sequence that is scheduled before j_{crit} but has a higher deadline than d_{crit}. $r_{int} < r_{crit}$ must hold, since j_{int} would otherwise not have been scheduled this early. Interference jobs are thus scheduled after their corresponding critical jobs to reduce the amount of tardy jobs.

VI. Evaluation

The evaluation was performed on over 40 000 randomly generated scheduling problems with 16 to 128 jobs. The amount of deadline and release time constraints per workflow is uniformly distributed between one and $|\mathcal{J}|$. Similarly, the tightness factor ($\Sigma wcet_i/\mathcal{D}$) was uniformly distributed between one (maximum tightness) and zero. Since the goal of the evaluation is to evaluate the efficiency of a scheduling algorithm, i.e. for how many of the feasible workflows it can find a valid schedule within a given CPU time budget, infeasible workflows have to be discarded first. As no optimal algorithm for the non-preemptive case exists we employ the preemptive least laxity first algorithm (LLF) to filter out definitely unschedulable workflows. LLF has been shown to be optimal for the preemptive single machine case [6]. Any workflow that is not schedulable in the preemptive case will remain so in the non-preemptive case. LLF discarded about half of the generated workflows as unsolvable, the remaining 20 503 jobs where then scheduled with each of the methods described in Sections IV and V and with simulated annealing (SA). For only two of these jobs no solution could be found with any of the employed methods, meaning that we have 20 501 jobs which comprise our set of feasible test cases.

We use Gurobi[1] in version 5.5 for solving the LP formulations for feasibility, not optimality. The simulated annealing portion is based on the Opt4J[2] framework using a simple linear temperature function and ran for 250 000 iterations. The test machines are equipped with an Intel Q6700 CPU at 2.66GHz and 8 gigabytes of RAM.

The time budget for each algorithm in Figure 1 was 10 seconds. The MIP approach is outperformed by all the heuristics, ruling them out for productive use. In the case

[1]http://www.gurobi.com/

[2]http://opt4j.sourceforge.net/

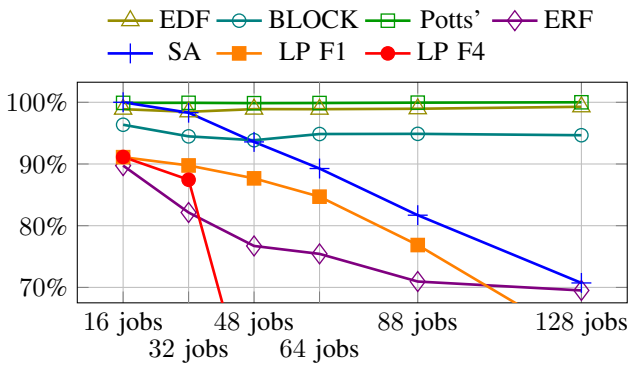

Fig. 1: Percent of test cases solvable by each method

of LP F4 the efficiency sinks to 10% for 128 jobs. ERF naturally also struggles to achieve a high efficiency because it does not take deadlines into account. However, it is more stable in the number of jobs. Simulated annealing performs well for small problem instances but looses efficiency for larger workflows because the state space it has to explore is growing exponentially. EDF, BLOCK and Potts' algorithm all perform well with over 95% efficiency on average. Potts' algorithm consistently performs at near 100% efficiency, there were only 18 out of 20 501 test cases where it did not find a solution, which equals an overall efficiency of over 99.9%. The combination of Potts' Algorithm, BLOCK and EDF finds a solution for all but 8 test cases. It is worth noting that ERF, EDF, BLOCK and Potts' algorithm all run in polynomial time and, on average, need less than one millisecond to generate a solution.

VII. Conclusion and Future Work

This paper shows that heuristics, especially Potts' algorithm, are able to solve our scheduling problem within a CPU time budget of 10s in nearly 100% of our test cases. This makes them a good fit for the scheduling component in our real-time SOA. MIP formulations are not suited to finding feasible solutions fast, but could be used to find optimal solutions in a later, separate step. Future work will implement the other building blocks, such as deriving local constraints from end-to-end deadlines and evaluate the concept in simulation as well as in real world testbeds.

References

[1] L. De Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio, "SOCRADES: A Web Service Based Shop Floor Integration Infrastructure," in *The Internet of Things*, ser. LNCS. Springer Berlin Heidelberg, 2008, vol. 4952.

[2] P. Chretienne, "A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints," *European Journal of Operational Research*, vol. 43, 1989.

[3] Z. Liu, "Single machine scheduling to minimize maximum lateness subject to release dates and precedence constraints," *Computers & Operations Research*, vol. 37, no. 9, 2010.

[4] L. A. Hall and D. B. Shmoys, "Jackson's rule for single-machine scheduling: making a good heuristic better," *Mathematics of Operations Research*, vol. 17, no. 1, 1992.

[5] J. W. F. Ahmet B. Keha, Ketan Khowala, "Mixed integer programming formulations for single machine scheduling problems," *Computers & Industrial Engineering*, vol. 56, 2009.

[6] A. K. Mok and M. L. Dertouzos, "Multiprocessor scheduling in a hard real-time environment," in *Seventh Texas Conf. Comput. Syst.*, 1978.