SCC: Project Report

# General details

Topic: Multi-objective workflow scheduling system

Strategy: MOHEFT [1]

Workflow: Custom

Objectives: Makespan & Cost

# Objective

With this project we want to implement a real-world workflow scheduling algorithm and evaluate its performance on a custom workflow which could be executed on Amazon EC2 instances. We use the workflow scheduling algorithm MOHEFT to schedule the execution of our workflow while optimizing multiple objectives, namely makespan and cost.

# Input

We performed benchmarking on EC2 instances of the types t2.nano, t2.micro, t2.medium, t2.xlarge and c5.large in order to establish the makespan of the individual tasks on different instance types. Next, we combined the makespan information (Figure 2 makespan.CSV) with the price per hour for the instances to arrive at the cost of execution for every task on every instance type (Figure 3: costs.CSV).

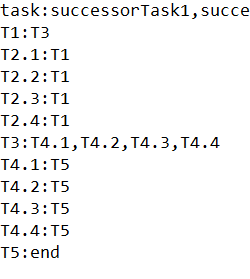
 

Figure 1: A textual representation (adjacency list) of the directed acyclic graph (DAG) representing our workflow, together with makespan.CSV and costs.CSV forms the decision base of our approach.

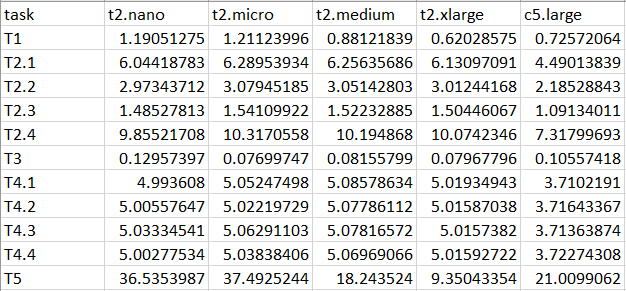


Figure 2 makespan.CSV

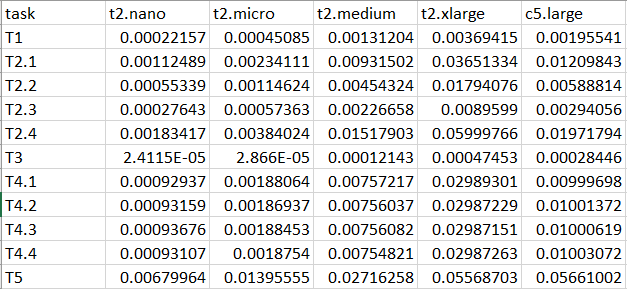
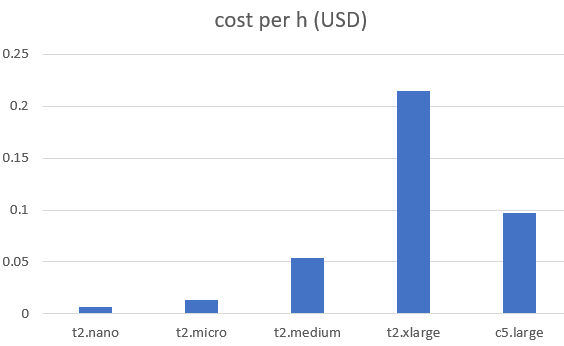
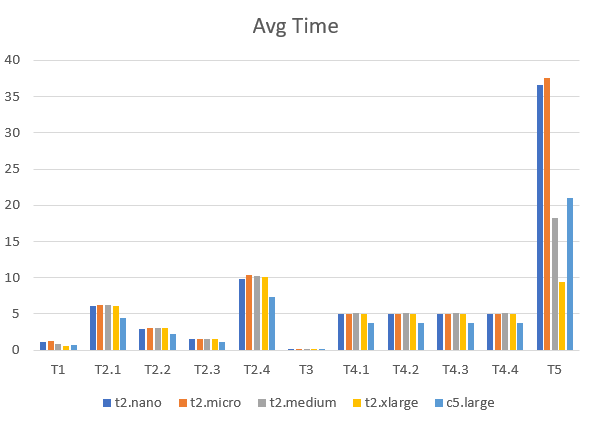


Figure 3: costs.CSV



The Tasks T2.x T3 and T4.x use at maximum one core while T1 and T5 profit from multiple cores, which results in significantly lower makespans if T1 and T5 are executed on multi core instances like t2.medium, t2.xlarge and c5.large. The cost of these instances is multiple times higher as the single-core instances.

# Diagram Overview of approach



Figure 4: Overview

The core of our approach is to perform multiple runs of MOHEFT using the aforementioned DAG, cost.CSV and makespan.CSV in order to generate trade-off solutions. In order to evaluate the trade-off schedules generated by MOHEFT we calculate the Nadir and Utopia points which we use to calculate the optimal hypervolume of the (not necessarily realizable) optimum. We finally calculate the hypervolume of each generated trade-off solution as to determine the best one as a means to gauge the overall performance of the workflow scheduling algorithm.

# Implementation

All presented artefacts were developed and evaluated on a Virtual Machine running the operating system “Ubuntu 16.04.4 LTS”. The used workflow scheduling algorithm based on MOHEFT, as well as the calculation of the Nadir point and Utopia point were realized using Java 10. For the calculation of the Hypervolume (HV) we reused a Python-based script developed by Simon Wessing (TU Dortmund University) [2]. The “run script” which executes the workflow scheduling algorithm and calculates the hypervolume of the results of a run is realized as a single Python script (run.py), which encompasses the whole evaluation process (execution, HV-calculation, comparison of results) depicted in Figure 4: Overview.

# Task Scheduling Algorithm



We based our task scheduling algorithm on the cloud-aware MOHEFT algorithm presented in [1]. After parsing the DAG and ranking the tasks using B-rank [3], a pool of available resources is set up (max N resources) and a set of K initial workflow schedules (S) is created. Next, the B-ranked individual tasks are scheduled iteratively. In every iteration, the set of intermediate workflow schedules (S\_tmp) is created by scheduling the current task on every available resource for every schedule in the set S. When scheduling tasks, the number of used resources is checked, which results in a cost and time of “infinity” if the maximum allowable number of resources is exceeded. Before advancing to the next task to be scheduled, the set S\_tmp is pruned by only keeping the schedules whose goal-domain representation lies on the pareto-front. Additionally, all remaining schedules in S\_tmp are sorted by their respective (descending) crowding distance as to achieve meaningfully diverse trade-offs. Finally, all but the first K entries are removed from S\_tmp and S is assigned to be S\_tmp. The result is complete if all tasks have been scheduled.

# Evaluation

1. Calculate optimal solution (feasible due to small problem)
2. Perform multiple runs of the task scheduling algorithm to compute approximated solutions
3. Compare solutions from different runs with hypervolume
4. Evaluate distance between best approximation and optimum

# Results

The execution of the workflow scheduling algorithm yields several trade-off solutions, which form a pareto-front and additionally have an adequate crowding distance. Using a maximum number of trade-off solutions (K) of 6 (graph above), the hypervolume of the best solution is 82.9% of the optimum hypervolume (Utopia point to Nadir point). Increasing K to 1000, the hypervolume of the best solution increases to 85.74% of the optimal hypervolume. Even larger values for K only lead to marginal improvements.

Looking at the found trade-off schedules as depicted in section “Example Usage”, we can see that trade-off solutions which result in shorter makespans use more of the expensive, multi core instances in order to execute the tasks T1 and T5, which can profit from multiple available cores.

# Example Usage





# References

|  |  |
| --- | --- |
| [1] | J. Durillo and R. Prodan, “Multi-objective workflow scheduling in Amazon EC2,” *Cluster computing,* vol. 17, no. 2, pp. 169--189, 2014. |
| [2] | S. Wessing , “Pure Python-based Hypervolume calculation script,” TU Dortmund University), obtainable from https://ls11-www.cs.tu-dortmund.de/rudolph/hypervolume/start, [Online]. Available: https://ls11-www.cs.tu-dortmund.de/rudolph/hypervolume/start. [Accessed 1 July 2018]. |
| [3] | H. Topcuoglu, S. Hariri and M. Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel Distrib. Syst.,* vol. 3, no. 13, p. 260–274, 2002. |