SCC: PROJECT REPORT

GENERAL DETAILS

TOPIC: Multi-objective workflow scheduling system

STRATEGY: MOHEFT [1]

WORKFLOW: Custom

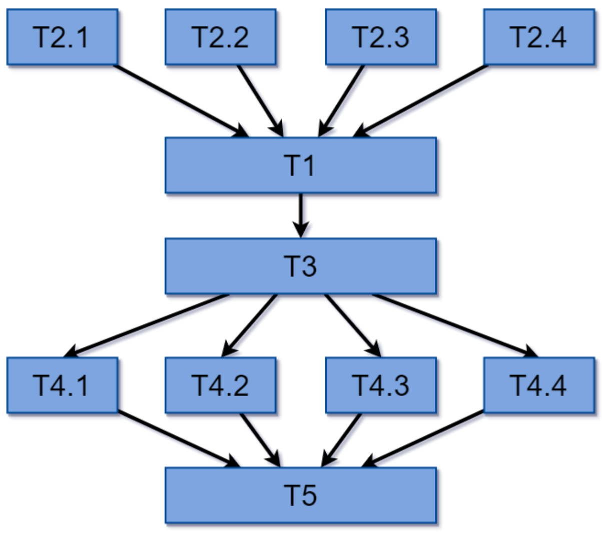
OBJECTIVES: Makespan & Cost

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.

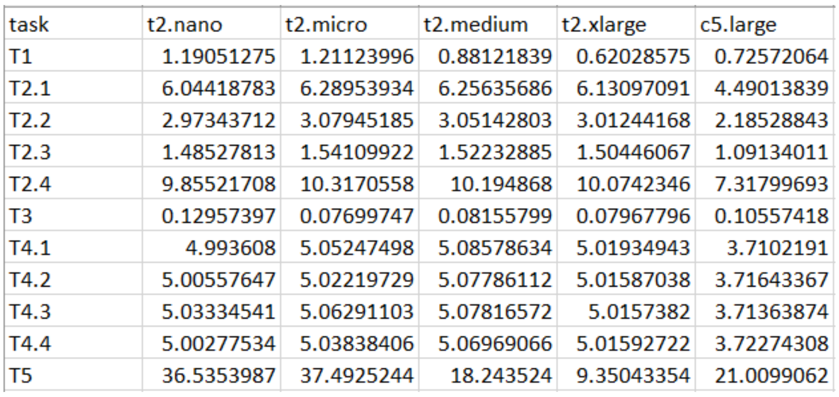
INPUT

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).



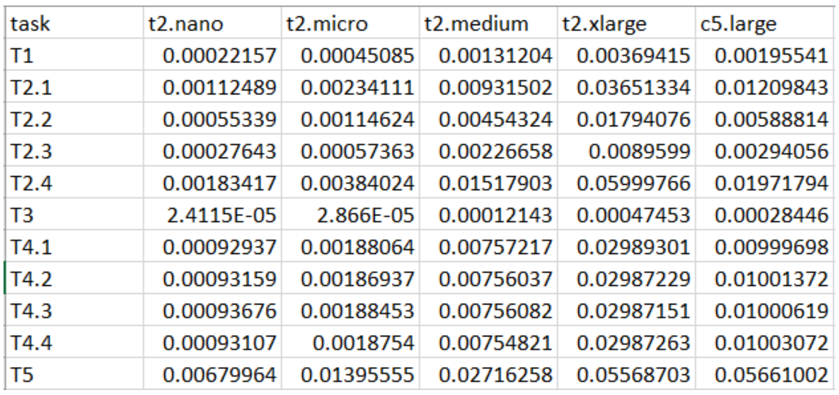
```
task:successorTask1,succe
T1:T3
T2.1:T1
T2.2:T1
T2.3:T1
T2.4:T1
T3:T4.1,T4.2,T4.3,T4.4
T4.1:T5
T4.2:T5
T4.3:T5
T4.4:T5
T5:end
```

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.



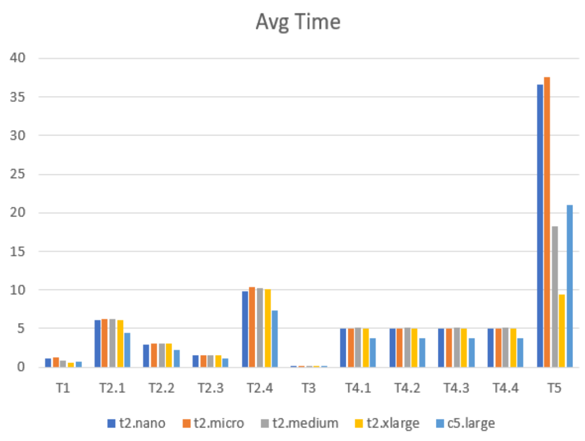
task	t2.nano	t2.micro	t2.medium	t2.xlarge	c5.large
T1	1.19051275	1.21123996	0.88121839	0.62028575	0.72572064
T2.1	6.04418783	6.28953934	6.25635686	6.13097091	4.49013839
T2.2	2.97343712	3.07945185	3.05142803	3.01244168	2.18528843
T2.3	1.48527813	1.54109922	1.52232885	1.50446067	1.09134011
T2.4	9.85521708	10.3170558	10.194868	10.0742346	7.31799693
T3	0.12957397	0.07699747	0.08155799	0.07967796	0.10557418
T4.1	4.993608	5.05247498	5.08578634	5.01934943	3.7102191
T4.2	5.00557647	5.02219729	5.07786112	5.01587038	3.71643367
T4.3	5.03334541	5.06291103	5.07816572	5.0157382	3.71363874
T4.4	5.00277534	5.03838406	5.06969066	5.01592722	3.72274308
T5	36.5353987	37.4925244	18.243524	9.35043354	21.0099062

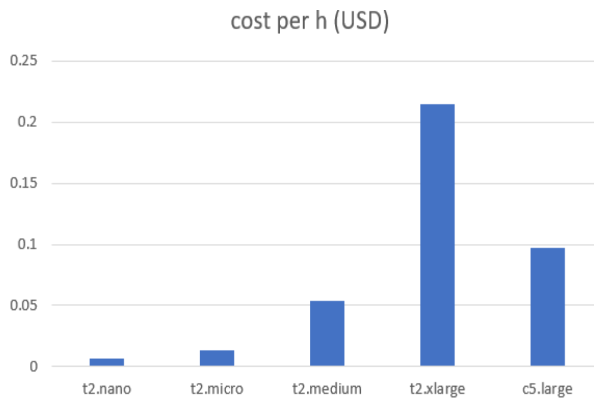
Figure 2 makespan.CSV



task	t2.nano	t2.micro	t2.medium	t2.xlarge	c5.large
T1	0.00022157	0.00045085	0.00131204	0.00369415	0.00195541
T2.1	0.00112489	0.00234111	0.00931502	0.03651334	0.01209843
T2.2	0.00055339	0.00114624	0.00454324	0.01794076	0.00588814
T2.3	0.00027643	0.00057363	0.00226658	0.0089599	0.00294056
T2.4	0.00183417	0.00384024	0.01517903	0.05999766	0.01971794
T3	2.4115E-05	2.866E-05	0.00012143	0.00047453	0.00028446
T4.1	0.00092937	0.00188064	0.00757217	0.02989301	0.00999698
T4.2	0.00093159	0.00186937	0.00756037	0.02987229	0.01001372
T4.3	0.00093676	0.00188453	0.00756082	0.02987151	0.01000619
T4.4	0.00093107	0.0018754	0.00754821	0.02987263	0.01003072
T5	0.00679964	0.01395555	0.02716258	0.05568703	0.05661002

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

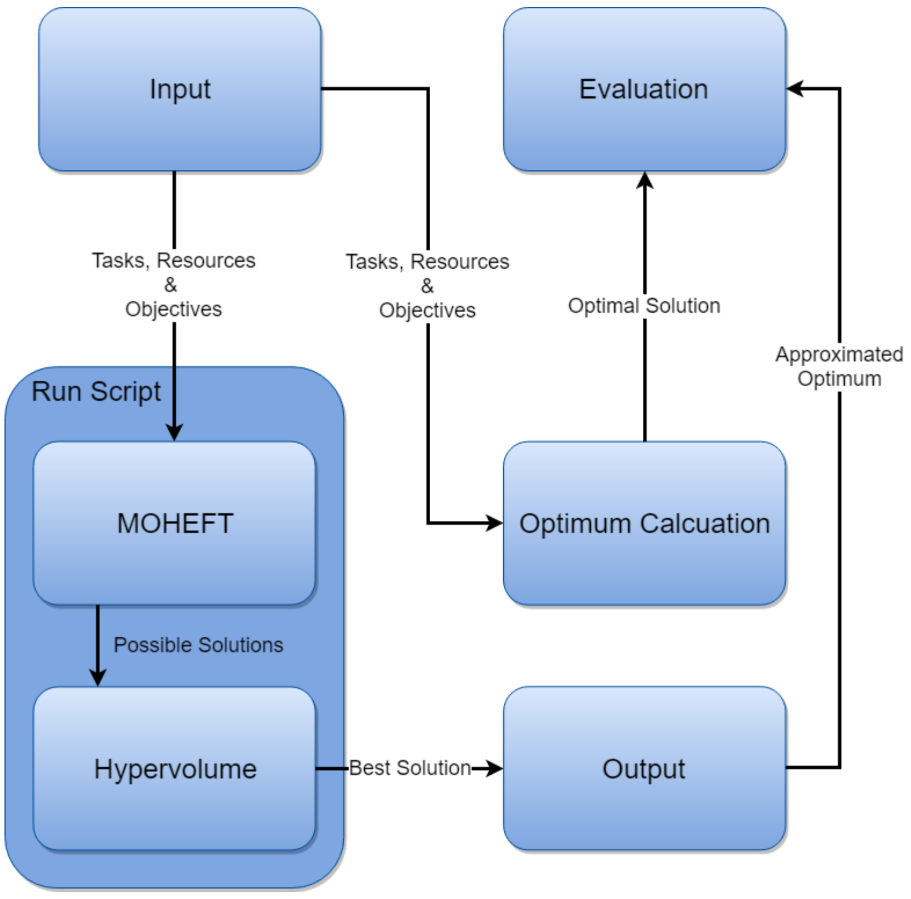


Figure 4: Overview

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

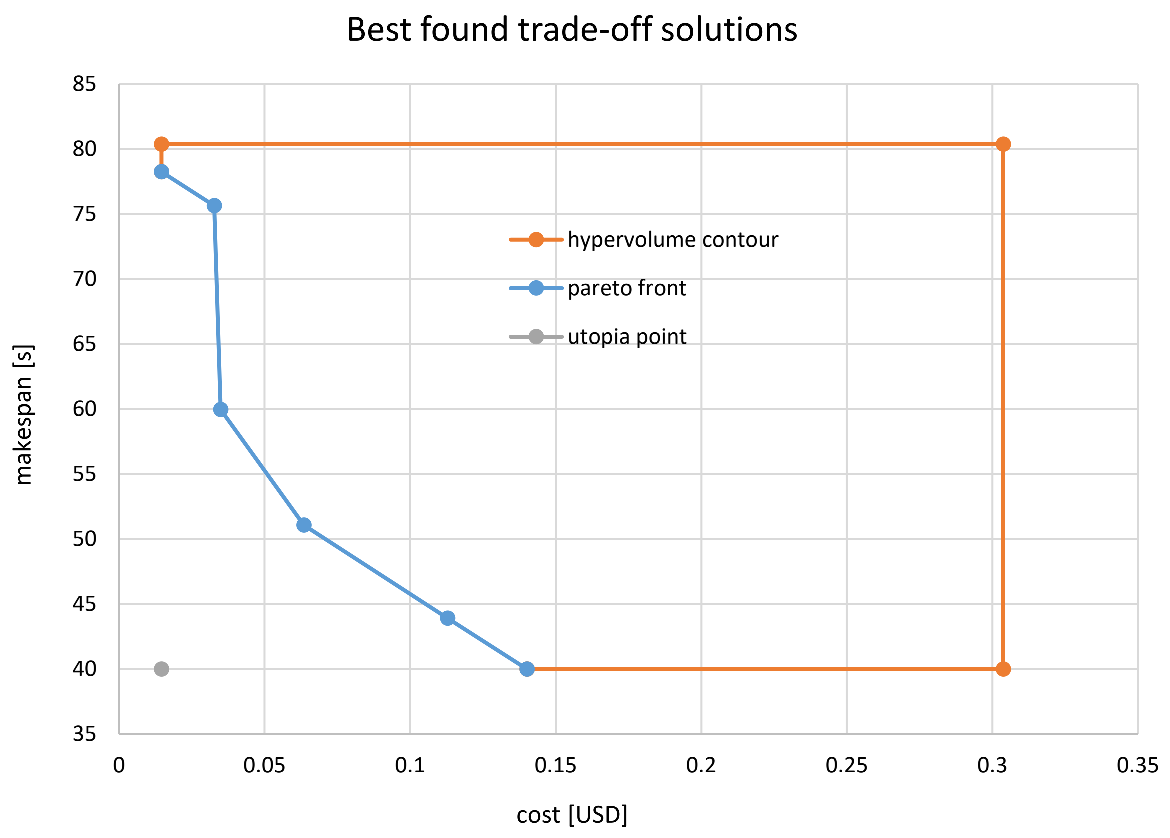
```
// R … set of available resources
// N … max number of resources to be used
// S … set of workflow schedules
// K … max size of S

// use B-rank to order tasks
bRankedTasks = getBRankedTasks();

// init ressources
for(String instanceType : instanceTypes)
  for(int i=0; i < MAX_INSTANCES_PER_TYPE; i++)
    R.add(instanceType+"_"+Integer.toString(i));

// set up initial set of workflow schedules
for(int i = 0; i < K; i++)
  S.add(new WorkflowSchedule(N));


// Iterate over the ranked tasks
for(int i = 0; i < bRankedTasks.length; i++){
  ArrayList<WorkflowSchedule> S_tmp = new ArrayList<>();

  // Iterate over all ressources
  for(int j = 0; j < R.size(); j++){

    // Iterate over all tradeoff schedules
    for(int k = 0; k < Math.min(S.size(),K); k++){
      // Extend all intermediate schedules
      w = new WorkflowSchedule(S.get(k));
      key = bRankedTasks[i] + "@" + R.get(j).split("_")[0];
      // schedule() sets costs/time to “infinity” if the number of used resources exceeds N
      w.schedule(R.get(j),bRankedTasks[i],costMap.get(key),makeSpanMap.get(key));
      S_tmp.add(w);
    }
  }

  // limit schedules to paretoFront
  S_tmp = getParetoFront(S_tmp);

  // sort S_tmp according to crowding distance
  S_tmp = sortByCrowdingDistance(S_tmp);

 // choose K schedules with highest crowding distance (first K schedules in sorted list)
 S_tmp.subList(Math.min(S_tmp.size(),K),S_tmp.size()).clear();
 S = S_tmp;
}


// finished!
```

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

```
user@ubuntu:~$ cd sccWFsched18/

user@ubuntu:~/sccWFsched18$ ls
benchmark  Documentation  Hypervolume  LICENSE  MOHEFT  README.md  Scripting

user@ubuntu:~/sccWFsched18$ cd Scripting/

user@ubuntu:~/sccWFsched18/Scripting   ls
run.py

user@ubuntu:~/sccWFsched18/Scripting$ python run.py
rm: cannot remove '/home/theuers/sccTasksched18/Scripting/results/*': No such file or directory
rm: cannot remove '/home/theuers/sccTasksched18/Scripting/bin/*': No such file or directory
Note: ../MOHEFT/src/WorkflowSchedule.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
run 0
run 1
using config-file from: /home/theuers/sccTasksched18/Scripting/config.txt
B-ranked task order: [T2.3, T2.2, T2.1, T2.4, T1, T3, T4.2, T4.4, T4.1, T4.3, T5]


Nadir point (cost,time):
0.30369981226152787,80.37198839200002

using config-file from: /home/theuers/sccTasksched18/Scripting/config.txt
B-ranked task order: [T2.3, T2.2, T2.1, T2.4, T1, T3, T4.2, T4.4, T4.1, T4.3, T5]


Utopia point (cost,time):
0.014562991724941455,39.995515203

optimum hypervolume: 11.6743250823

All runs:
('tradeoffs_run_0.CSV', 9.6785855305)
('tradeoffs_run_1.CSV', 9.6785855305)



--------------------------------------------------------------------------------
```

```
Best run:
('tradeoffs_run_0.CSV', 9.6785855305)   82.9048828285% of optimum hypervolume

totalCost,totalTime
0.14010251196649326,39.995515203
0.014562991724941455,78.248910758
0.034925928229623315,59.957036067999994
0.03270003677619825,75.64581517900001
0.06345037450088392,51.06394562799999
0.11284857065902806,43.900655650999994

using config-file from: /home/theuers/sccTasksched18/Scripting/config.txt
B-ranked task order: [T2.3, T2.2, T2.1, T2.4, T1, T3, T4.2, T4.4, T4.1, T4.3, T5]
Ressources: [t2.nano_0, t2.nano_1, t2.nano_2, t2.nano_3, t2.nano_4, t2.micro_0, t2.micro_1,
t2.micro_2, t2.micro_3, t2.micro_4, t2.medium_0, t2.medium_1, t2.medium_2, t2.medium_3,
t2.medium_4, t2.xlarge_0, t2.xlarge_1, t2.xlarge_2, t2.xlarge_3, t2.xlarge_4, c5.large_0, c5.large_1,
c5.large_2, c5.large_3, c5.large_4]


Found tradeoff schedules:
totalCost: 0.14010251196649326          totalTime: 39.995515203
          c5.large_0      [T2.3, T2.2, T2.1, T2.4, T4.2, T4.4, T4.1, T4.3]
          t2.micro_0      [T3]
          t2.xlarge_0     [T1, T5]

totalCost: 0.014562991724941455         totalTime: 78.248910758
          t2.nano_0       [T2.3, T2.2, T2.1, T2.4, T1, T3, T4.2, T4.4, T4.1, T4.3, T5]

totalCost: 0.034925928229623315         totalTime: 59.957036067999994
          t2.medium_0     [T5]
          t2.nano_0       [T2.3, T2.2, T2.1, T2.4, T1, T3, T4.2, T4.4, T4.1, T4.3]

totalCost: 0.03270003677619825          totalTime: 75.64581517900001
          c5.large_0      [T4.1, T4.3]
          t2.nano_0       [T2.3, T2.2, T2.1, T2.4, T1, T3, T4.2, T4.4, T5]

totalCost: 0.06345037450088392          totalTime: 51.06394562799999
          t2.nano_0       [T2.3, T2.2, T2.1, T2.4, T1, T3, T4.2, T4.4, T4.1, T4.3]
          t2.xlarge_0     [T5]

totalCost: 0.11284857065902806          totalTime: 43.900655650999994
          c5.large_0      [T2.3, T2.2, T2.1, T2.4, T4.3]
          t2.nano_0       [T3, T4.2, T4.4, T4.1]
          t2.xlarge_0     [T1, T5]
```

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.