Qualification Task Description

Designed by: Huawei Research in Stockholm

Task Description:

1. Let us assume a 5G RAN real-time application including a set of Directed Acyclic Graphs (DAGs), each of which implies an end-to-end functionality of the system.
2. Each DAG:
   - Includes a set of dependent tasks, and
   - Has a certain relative deadline.
3. Each DAG has multiple instances released periodically over time. For example, D_1^i and D_1^{i+1}denote the *ith* and *i+1th* instances of the DAG 1, respectively.

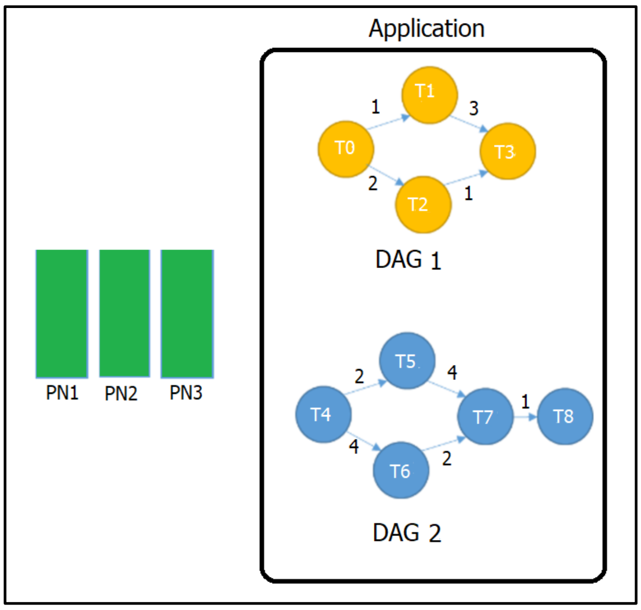


Fig1: Two DAGs, DAG 1 and DAG 2 which need to be run on three processing nodes PN1, PN2 and PN3. The numbers on edges show inter-task communication times in micro second (µs).

Important factors:

1. Data dependency between tasks of a DAG: Each task can start its execution only once all its parents are completed and the required data of the task, generated by its parents, has arrived.

2. Communication time: The time which takes to send the parents' data to their children, if they are located on different processing nodes; otherwise it can be assumed negligible. As a result, we prefer to assign communicating tasks on the same processing node.

3. Affinity: It refers to the affinity of a task to its previous instances running on the same processing node that can reduce overhead to initialize the task, such as a lower Instruction Cache Miss. Ideally the task is better to run on the same processing node where its previous instance was recently run.

4. Load Balancing of processing nodes: The CPU utilization of processing nodes should be balanced and uniformed.

Assumptions:

1. If communicating tasks assigned to the same processing node, the communication time between them is negligible, i.e., equal to 0.
2. If the previous instance of the same task is *recently assigned* to the same processing node, the estimated execution time of the current instance of the task reduces by 10%. For example, if T0 is assigned to PN1, the execution time of the second instance of T0 (denoted by T0') on PN1 is 9µs, rather than 10µs.
3. "Recently assigned" can be translated to:
    - If the previous instance of the current task is among the last X tasks run on the PN.
    - For this purpose we need to keep, a history of the X recent tasks which run on each PN.
4. A DAG's deadline is relative to its release time which denoted by d_i. For example, if the deadline of a DAG is 3 and the release time of its *ith* instance is 12, it should be completed before 15.
5. All time units are in micro second.
6. The execution of tasks are non-preemptive, in the sense that a task that starts executing on a processor will not be interrupted by other tasks until its execution is completed.

Problem Formulation:

1. Let us consider a real-time application including *n* DAGs including DAG1, DAG2, … , DAGn, each of which are periodically released with period P_k. $Inst_i$ instances of each DAG is released over the course of running the application. The *i*th instance of the *k*th DAG is denoted by $D_k^{(i)}$.
2. The application is run on *m* homogenous processing nodes PN1, PN2 to PNm. The proposed algorithm should find a solution on how to assign the tasks of DAGs to the processing nodes such that not only all the DAGs deadlines are respected but also the makespan of the given application is minimized.
3. Makespan is the time where all instances of all DAGs of the application are completed.

Questions:

1. Propose an algorithm to solve the considered problem to maximize the utility function including both the total application Makespan and the standard deviation of the PN utilizations (i.e., how well-uniform is the assignment) such that both task dependency constraints and DAGs deadlines are met.

   $\text{Utility Function} = 1 \,/\, (10 * \text{Normalized(Makespan)} + \text{Standard Deviation of PN utilizations})$
   Where

$$\text{Normalized(Makespan)} = \frac{Makespan}{\text{Application\_worst\_case\_compelition\_time}}$$

Application_worst_case_completition_time is the sum of execution times and communication times of all DAG instances of the given test case. Indeed, it is an upper-bound for the makespan of an application. Therefore, both Normalized makespan and the standard deviation are real numbers between zero and one.

The algorithm should specify:

- Assignment of tasks to processing nodes to maximize the utility function. In the sense that your algorithm should find the best processing nodes to run the tasks.
- Task priorities. The order the tasks are scheduled and the execution order of the tasks on each processing node.

Input and output:

1. The input of the scheduler is a set of 12 test cases each of which implies a real-time application. Test cases are all given to you as a starting package at the beginning of the qualification phase in Slack. Each test case is shown by a JSON file that includes a set of independent DAGs. The JSON file also includes the deadline of the DAGs, number of instances of each DAG, period of the DAGs, list of their tasks, execution time and inter-task communication times.
2. Let's assume the considered real-time applications need to run on an embedded processor with a given number of homogenous cores. The number of cores is mentioned in each test case. Some test cases runs on a 6- core processor while other runs on an 8-core processor.
3. The output should be a CSV file including:
   - The PN_id to which each task is assigned (0,1,...,7),
   - Order of execution of the tasks in the assigned PN,
   - Start and finish time of each task,
   - Application makespan,
   - The standard deviation of the clusters' utilization,
   - Value of the utility function.
   - The execution time of the scheduler on your machine.

The CSV file example:

	Tuple(Task ID, Start time, Finish time)			
Processor 1	0,0,10	3,43,53	1000,60,69	1003,99,108
Processor 2	1,11,31	1001,70,88		
Processor 3	2,12,42	1002,71,98		
Makespan	108			
Standard deviation of processors loads	0.082932			
utility function	0.141			
Execution time of the scheduler (in milliseconds)	3			

Note: commas are added only for readability. They don't need to be included in the answer files. Please look at the csv file in the starting package.

Note for C coders: If you code in C, we already developed a JSON reader function parsing the DAGs information of each test case, a printer function creating such a standard csv file for each test case, and a simple scheduler to give you an idea how it looks like. All the items are included in the starting package. Although, using the printer function and the JSON reader function are recommended, they are still optional.

Note for Python coders: If you code in Python, you need to write your own printer function to create the csv files in the specified format.

Evaluation Criteria:

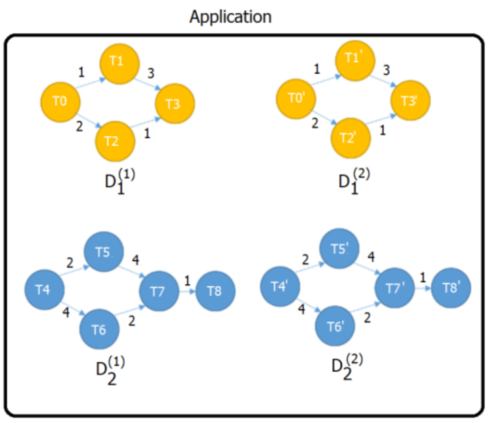
- The score of the proposed scheduler is evaluated based on:

  1. All the above mentioned constraints (including perquisites and deadlines) are met. Otherwise, the score is 0.
  2. The execution time of the scheduler should be short enough to make it feasible for real-time scheduling, i.e., in scale of a few milliseconds on a VM with an Intel processor 3.0 GHz eight-cores and 16GB RAM. If the execution time of your scheduler on such a VM is much greater, more than 1 second, the score is 0.

     Note: As most of today's laptops are not more powerful than the considered VM, if the execution time of your scheduler is in the scale of a few milliseconds on your laptop, you can ensure that it doesn't take longer on the considered VM.
  3. The scheduler can achieve as high value as possible for the utility function for the given set of test cases.
  4. The ranking is done primarily based on the number of test cases for which the scheduler can fulfill all the requirements, which are called *passed test cases*. If two teams achieve the same count of passed test cases, sum of the scores of all the passed test cases are considered as the preference metric. If two teams exactly achieve the same number of passed test cases and the same total score, the faster code in terms of the execution time of the scheduler is preferred to advance to the final stage.

Solution Example:

- We have an application which consists of 2 DAGs and 2 instance for each DAG.
- The goal is to maximize the utility function.
- D_1period is 100µs which means that its first instance $D_1^{(1)}$ arrives in 0µs and the second instance $D_1^{(2)}$ arrives in 100µs. D_2 period is 120µs which means that its first instance arrives in 0µs and its next instance arrives in 120µs.
- The deadline of each DAG is equal to its period, i.e., $P_i = d_i$
- The board that runs the application has 3 homogenous processing nodes PN1, PN2 and PN3.
- The size of history of the recent tasks running on each PN is set to 4, i.e., X=4.
- Let us assume the proposed scheduler does the assignment as follow:

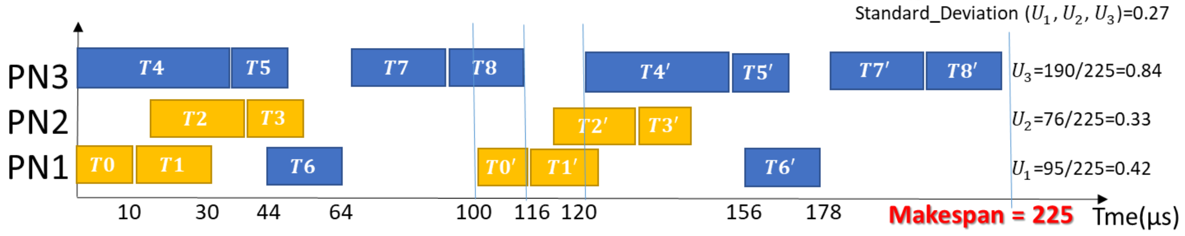


DAG2 Tasks	Estimated Execution Time (μs)	Assigned PN
T4	40	PN3
T5	10	PN3
T6	20	PN1
T7	30	PN3
T8	20	PN3
T4´	40-0.1*40=36	PN3
T5´	10-0.1*10=9	PN3
T6´	20-0.1*20=18	PN1
T7´	30-0.1*30=27	PN3
T8´	20-0.1*20=18	PN3

DAG1 Tasks	Estimated Execution Time (μs)	Assigned PN
T0	10	PN1
T1	20	PN1
T2	30	PN2
T3	10	PN2
T0´	10-0.1*10=9	PN1
T1´	20-0.1*20=18	PN1
T2´	30-0.1*30=27	PN2
T3´	10-0.1*10=9	PN2

Utility Function= $\frac{1}{10 * \frac{225}{420} + 0.27} = \frac{1}{5.627} = 0.177$

Standard_Deviation (U_1, U_2, U_3)=0.27


PN3
PN2
PN1
$T4$
$T5$
$T7$
$T8$
$T4'$
$T5'$
$T7'$
$T8'$
$T2$
$T3$
$T2'$
$T3'$
$T0$
$T1$
$T6$
$T0'$
$T1'$
$T6'$
U_3=190/225=0.84
U_2=76/225=0.33
U_1=95/225=0.42
Makespan = 225
Tme(μs)