13th Symposium on Cloud Computing 2022

Cloud-native Workflow Scheduling using a Hybrid Priority Rule and Dynamic Task Parallelism

Jungeun Shin[1], Diana Arroyo[2], Asser Tantawi[2], Chen Wang[2] , Alaa Youssef[2], Rakesh Nagi[1]

[1] University of Illinois at Urbana-Champaign
[2] IBM Thomas J. Watson Research Center

San Francisco, CA
Nov. 8-10, 2022

The Grainger College of Engineering
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

ACM Symposium on Cloud Computing

Outline

1) Introduction

2) Workflow Scheduling Algorithm

- Hybrid task scheduling rules
- Dynamic task splitting rule for parallelism

3) Simulation Results

4) Conclusions and Future work

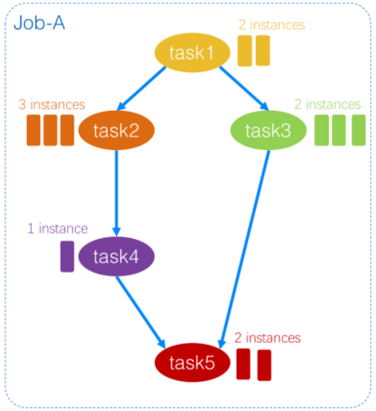
Introduction

- Increasing demand for efficient workflow scheduling as machine learning and data processing projects shift to cloud.
- Many workflow orchestration tools only consider dependencies between tasks and use simple heuristic for scheduling policy.

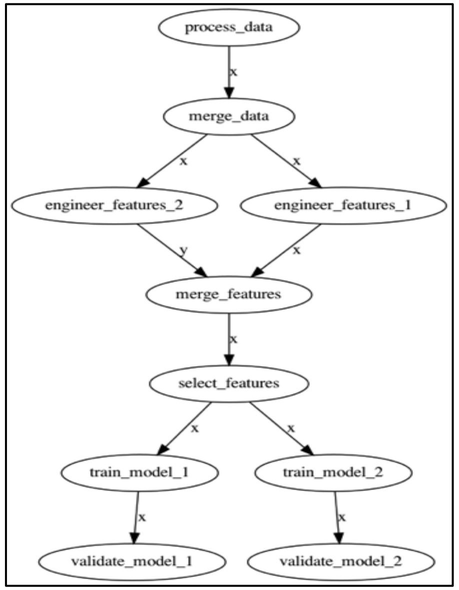
→ Workflow-aware scheduling that considers characteristics of workflows to achieve faster job processing and higher resource utilization.

- Also, they rely on user input for parallelism level.

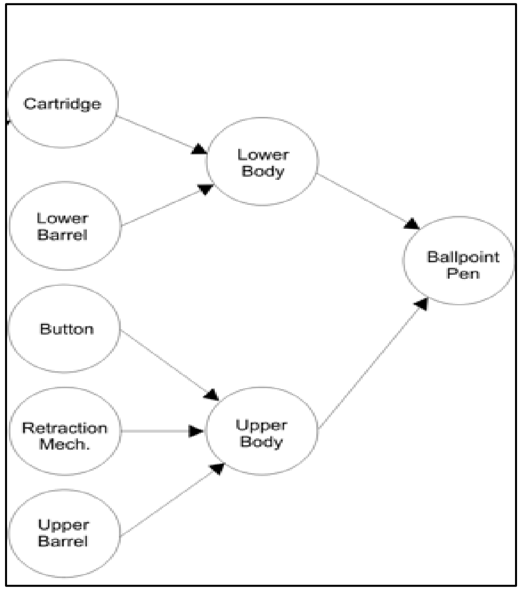
→ Dynamic task splitting rule to determine task parallelism level considering resource availability and task duration



- Cross-pollinating methods from production scheduling and cloud scheduling.
  - ML&RL workflows that comprises a sequence of tasks that have dependency to each other, and it can be represented as *Directed Acyclic Graph (DAG)*.
  - Production scheduling problems modeled with *DAG* have been studied for decades.
  - Task splitting is a less studied problem in both fields.



Sample ML Pipeline
(source: https://medium.com/hashmapinc)



Assembly Chart of Ballpoint Pen
(source: https://www.idef.com/)

Workflow Scheduling Algorithm

Hybrid task scheduling rules

Objective : improving <u>resource utilization</u> and minimizing <u>weighted workflow completion time</u>

* Workflow duration multiplied by user specific workflow priority.

- *Maximum Children (MC)* rule releases the dependencies faster by prioritizing a task with the largest number of successors.
- *Weighted Shortest Critical Path Time first (WSCPT)* rule prioritizes a task that has the largest ratio of workflow weight over the remaining critical path time.

→ Our hybrid task scheduling rules combine *MC* and *WSCPT* rules and use them alternatively.
With a switching threshold parameter k, apply *MC* if task queue length < k and *WSCPT* o.w

How to determine k ?

We analyze the cluster status using a Continuous-Time Markov Chain and exploit it to find the smallest k, which keeps cluster busy enough.

<u>Notations</u>

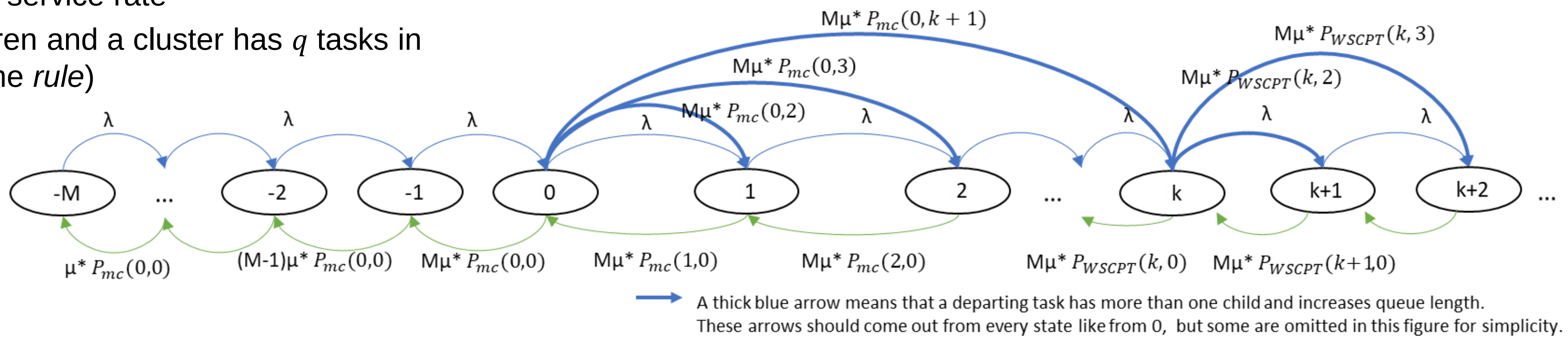
State space *s*: Queue length (if *s* > *0*) or Number of idle servers (if *s* ≤ *0*)

λ: Workflow arrival rate / μ: Single server service rate

$P_{rule}(q,d)$: P(a departing task has c children and a cluster has q tasks in queue with scheduling regime *rule*)

$-P_{mc}(q,d)$: $\sum_{y=1}^{q}\binom{q}{y}\frac{1}{D+1}^{y}\frac{d}{D+1}^{(q-y)}$

$-P_{wscpt}(q,d)$: $\frac{1}{D+1}$



Workflow Scheduling Algorithm

Dynamic task splitting rule for parallelism

Task Completion Time (TCT) change with different parallelism levels.

- TCT initially decreases with parallelism as the durations of subtasks reduce
- TCT then increases with high parallelism because some subtasks can be pending if cluster doesn't have enough available executors to run them simultaneously.

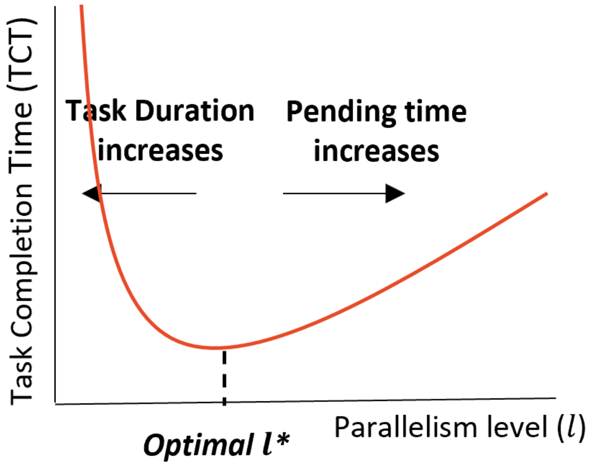


Figure 1 (a) Task Completion Time(TCT) curve

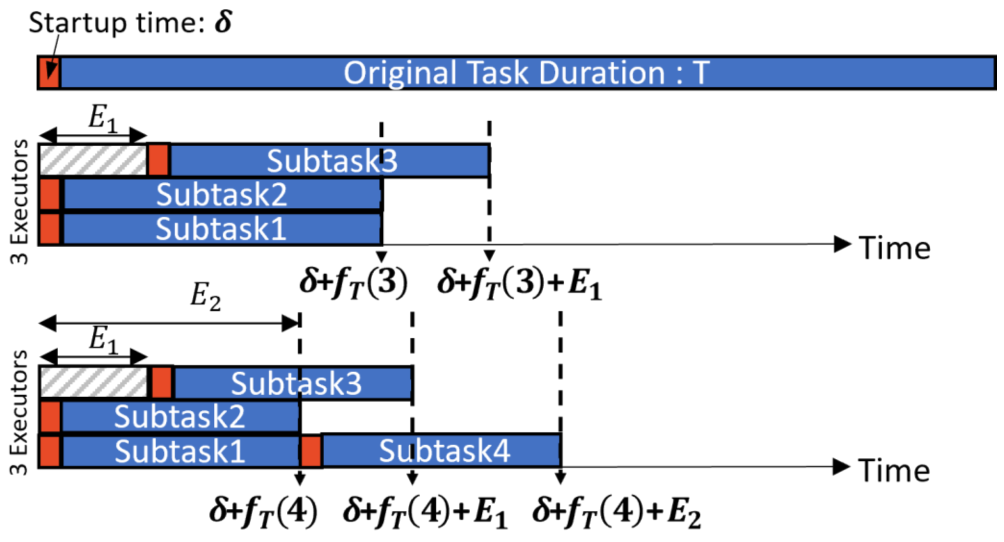


Figure 1 (b) TCT with parallelism level l=4 can be larger than l=3

δ : Task startup time

$f_T(l)$: subtask duration with an original task duration T and parallelism level l

$P_s(x)$: P(Cluster has x idle slots when a new task arrives)

E_i: The expected time until i-th next executor becomes available

Assuming that the time until each executor becomes idle is independent and identically distributed with mean w.

$$TCT(l) = \delta + f_T(l) + P_s(l-1)E_1 + P_s(l-2)E_2 + \cdots$$

$$= \delta + f_T(l) + \sum_{i=1}^{l-1} P_s(l-i)\, E_i$$

$$= \delta + f_T(l) + \sum_{i=1}^{l-1} P_s(l-i)\, * w * i$$

Simulation Results

Hybrid task scheduling rules

First, find the switching threshold parameter (k) to use

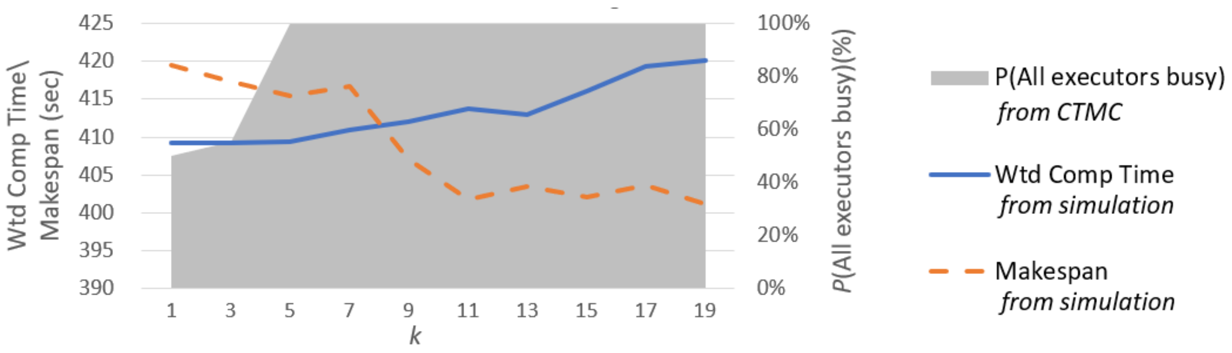


Figure 2: A hybrid rule switching threshold (k) analysis by $CTMC$ and simulation.

Hybrid rule with k =7, (no parallelism)

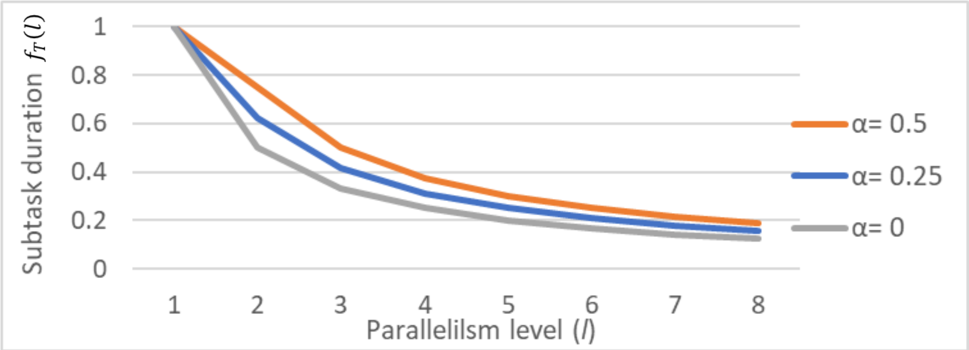
Scheduling rule	Wtd Comp Time(sec) Value	Wtd Comp Time(sec) Diff(%)	Makespan(sec) Value	Makespan(sec) Diff(%)
Hybrid rule	**412**	-	**409**	-
FCFS	425	+13 (+3.0%)	419	+10 (+2.4%)
SJF	448	+36 (+8.7%)	398	-11 (-2.7%)
WSCPT	409	-3 (-0.7%)	420	+10 (+2.5%)
MC	445	+33 (+7.9%)	389	-20 (-4.8%)

Table 1: Performance comparison of different task scheduling rules

Dynamic task splitting rule for parallelism

The rate of change in subtask duration depending on parallelism level can vary.

* Subtask duration function : $f_T(l) = \frac{T}{l}(1 + \alpha)$, α = proportionality constant



Splitting Rule	Wtd Comp Time(sec) $\alpha = 0$	0.25	0.5	Makespan(sec) $\alpha = 0$	0.25	0.5
Dynamic parallelism	**242**	**312**	**388**	**337**	**401**	**466**
1 (No parallelism)	412	515	616	409	497	593
3	269	338	406	364	434	501
5	263	327	391	392	457	522
7	288	346	403	435	498	561

Table 2: Performance comparison of different task splitting rules

Conclusions and Future work

- Our cloud-native scheduling heuristic leverages workflow information and cluster status for scheduling workflow.
- The heuristic produces an efficient balance of weighted workflow completion time and resource utilization of the cluster.
- Our approach can be easily applicable to any workflow scheduling system without the need of analyzing historical data or learning policies in advance.

We plan

- to consider preemptible workflows and handle them efficiently using scheduler.
- to apply our algorithm to existing established simulators and use real-world traces for experiments.

Thank you
Q & A

Jungeun Shin
jungeun4@illinois.edu