OPERATING SYSTEMS PROJECT

A NOVEL TASK-DUPLICATION BASED DAG SCHEDULING ALGORITHM FOR HETEROGENEOUS ENVIRONMENTS

Kun He, Member, IEEE, Xiaozhu Meng, Zhizhou Pan, Ling Yuan, Pan Zhou Member, IEEE

BY:
JAI GARG (DTU/2K18/MC/044)
JATIN PAPREJA (DTU/2K18/MC/049)

Introduction:

- ***Heterogeneous systems*** consists of multiple cores having different architecture resulting in different computational power for each core. As a crucial task in heterogeneous distributed systems, DAG-scheduling models a scheduling application with a set of distributed tasks by a ***Direct Acyclic Graph (DAG).***
- In this project, we have implemented a novel algorithm, called the ***Task Duplication based Clustering Algorithm (TDCA),*** to effectively perform task duplication and improve the scheduling quality. ***TDCA*** improves the ***schedule performance*** by utilizing duplication task more thoroughly. It improves parameter calculation, task duplication, and task merging. ***TDCA*** aims to improve upon the ***schedule makespan for heterogeneous systems*** for various communication-computing cost ratios.

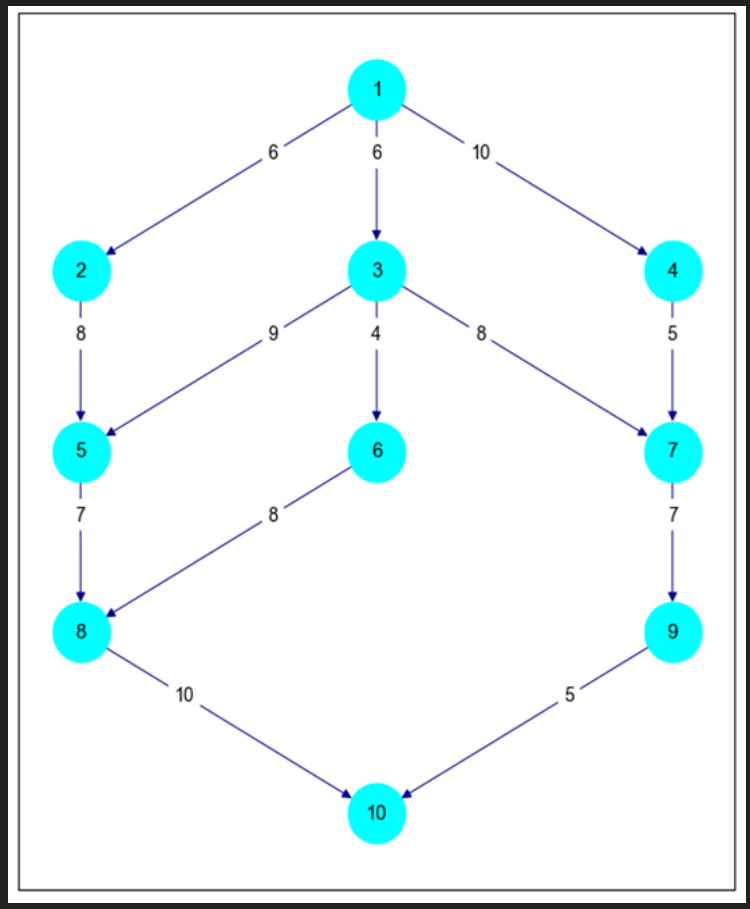


Fig. 1. A DAG Example

Literature Survey:

There are three main types of the DAG scheduling algorithms: ***List Scheduling***, ***Cluster-Based Scheduling***, and ***Task Duplication-Based Scheduling***. ***Hybrid algorithms*** have also been developed that combines algorithms given above. Various Scheduling Algorithms have been developed such as:

- ***Heterogeneous Earliest Finish Time (HEFT)***
- ***Dynamic Critical Path Duplication (DCPD)***
- ***Task duplication-based scheduling Algorithm for Network of Heterogeneous systems (TANH)***

The proposed algorithm ***TDCA*** is based on the framework introduced in ***TANH*** scheduling. ***TANH*** algorithm allows other algorithms to take a huge leap. But, it has some limitations as well such as:

- Definitions of several parameters are rather loose.
- In the initial cluster generation step, it always keeps adding nodes to a cluster until the entry-node is added.
- In the task duplication step, it does not show the exact order of the positions of performing task duplication.

One key difference between ***TDCA*** and ***TANH*** is that ***TDCA*** uses both task duplication and cluster merging to reduce the makespan, while ***TANH*** only applies one depending upon the initial clustering.

Approach:

The ***Task Duplication-Based Clustering Algorithm (TDCA)*** has been broadly categorized into five phases:

1. ***Definitions of Key Parameters***
2. ***Phase I - Initial Task Clustering***
3. ***Phase II - Task Duplication***
4. ***Phase III - Task Merging***
5. ***Phase IV - Task Insertion***

Key Parameters:

- ***Earliest Starting Time (est)***
- ***Earliest Completion Time (ect)***
- ***Favorite Processor (fproc)***
- ***Critical Predecessor (cpred)***
- ***B-level Task Priority (level)***

```
def Initial_Entry_Table(n, m, Dag_succ, Dag_pred, T, C):
    est = [[0 for j in range(m)] for i in range(n)]

    ect = [[[0, j] for j in range(m)] for i in range(n)]
    for j in range(m):
        ect[0][j][0] = T[0][j]

    sort_ect = sorted(ect[0])
    fproc = [[0 for j in range(m)] for i in range(n)]
    for j in range(m):
        fproc[0][j] = sort_ect[j][1]

    for i in range(1, n):
        for j in range(m):
            est[i][j] = est_max(i, j, Dag_pred, ect, C, fproc)
            ect[i][j][0] = est[i][j] + T[i][j]
        sort_ect = sorted(ect[i])
        for j in range(m):
            fproc[i][j] = sort_ect[j][1]

    cpred = [0 for i in range(n)]
    for i in range(n):
        cpred[i] = cpred_max(i, Dag_pred, ect, fproc, C)

    level = [0 for i in range(n)]
    for i in range(n-1, -1, -1):
        level[i] = level_max(i, T, Dag_succ, level, C)

    print_table(n, m, est, ect, fproc, cpred, level)
    return est, ect, fproc, cpred, level
```

Fig. 2. Code Cell for calculating Key Parameters

Four Phases Of TDCA:

- ***Initial Task Clustering Phase:***

  After the calculation of the key parameters, tasks are sorted by their levels in non-decreasing order. ***TDCA*** starts to construct the initial clusters.

- ***Task Duplication Phase:***

  A task duplication method is used to modify the initial clusters in order to shorten the makespan.

- ***Task Merging Phase:***

  In this phase, we try to merge clusters to see if the makespan could be further reduced.

- ***Task Insertion Phase:***

  In this phase, for each predecessor-successor pair, the predecessor is inserted in the cluster containing the successor, if this improves scheduling performance.

```
def calc_makespan(n, m, T, C, Dag_succ, Dag_pred, cluster_mat):
    makespan = [[0 for j in range(m)] for i in range(n)]

    for p in range(m):
        if 0 in cluster_mat[p]:
            makespan[0][p] = T[0][p]

    for i in range(1, n):
        for p in range(m):
            if i not in cluster_mat[p]:
                continue
            cand_list = []
            for j in Dag_pred[i]:
                if j in cluster_mat[p]:
                    a = makespan[j][p]
                    cand_list.append(a)
                    continue
                else:
                    for k in range(m):
                        if j in cluster_mat[k]:
                            a = makespan[j][k] + C[j][i]
                            cand_list.append(a)
            makespan[i][p] = max(cand_list) + T[i][p]

    return max(makespan[n-1])
```

Fig. 3. Function for calculating Schedule Makespan

Results:

In order to analyze the results and performance, ***TDCA*** is compared to the well known scheduling algorithms namely: ***DCPD, HEFT, TANH.*** The ***Time Complexity*** of the ***TDCA*** comes out to be ***O(mne + m²e)*** whereas the ***Space Complexity*** comes out to be ***O(mn)***. The algorithms are evaluated on various parameters in which ***CCR*** has the ***greatest impact on the performance*** of the algorithms.

```
P1 -> 1 2
P2 ->
P3 -> 1 4
P4 -> 1 3 7 9
P5 -> 1 3 5 6 8 10
Makespan : 27
```

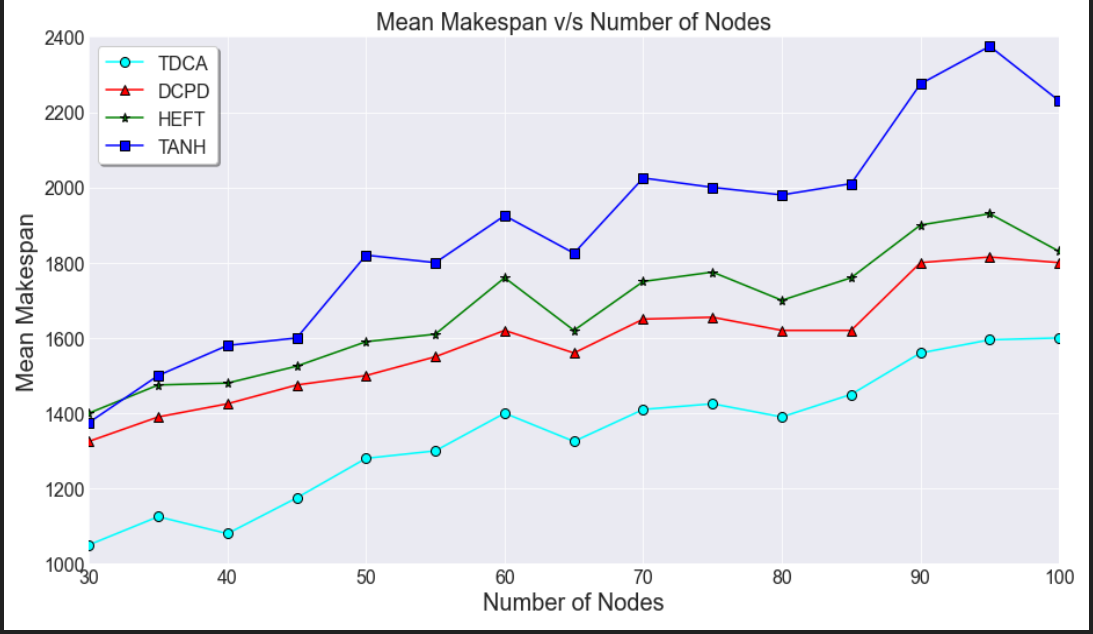
Fig. 4. Final Schedule

Task Number	Earliest Starting Time	Earliest Completion Time	Favourite Processor	Critical Predecessor	Level
1	[0, 0, 0, 0, 0]	[3, 6, 5, 5, 4]	[1, 5, 3, 4, 2]	0	65
2	[3, 6, 5, 5, 4]	[9, 9, 10, 12, 7]	[5, 1, 2, 3, 4]	1	52
3	[3, 6, 5, 5, 4]	[9, 11, 12, 12, 10]	[1, 5, 2, 3, 4]	1	53
4	[3, 6, 5, 5, 4]	[8, 12, 9, 9, 8]	[1, 5, 3, 4, 2]	1	41
5	[9, 11, 12, 12, 10]	[16, 13, 16, 14, 12]	[5, 2, 4, 1, 3]	3	37
6	[9, 11, 12, 12, 10]	[15, 18, 16, 18, 15]	[1, 5, 3, 2, 4]	3	38
7	[9, 12, 12, 12, 10]	[15, 16, 17, 16, 16]	[1, 2, 4, 5, 3]	3	30
8	[16, 18, 16, 18, 15]	[22, 24, 21, 23, 21]	[3, 5, 1, 4, 2]	6	23
9	[15, 16, 17, 16, 16]	[19, 21, 20, 18, 19]	[4, 1, 5, 3, 2]	7	17
10	[22, 24, 21, 23, 21]	[25, 25, 28, 24, 23]	[5, 4, 1, 2, 3]	8	7

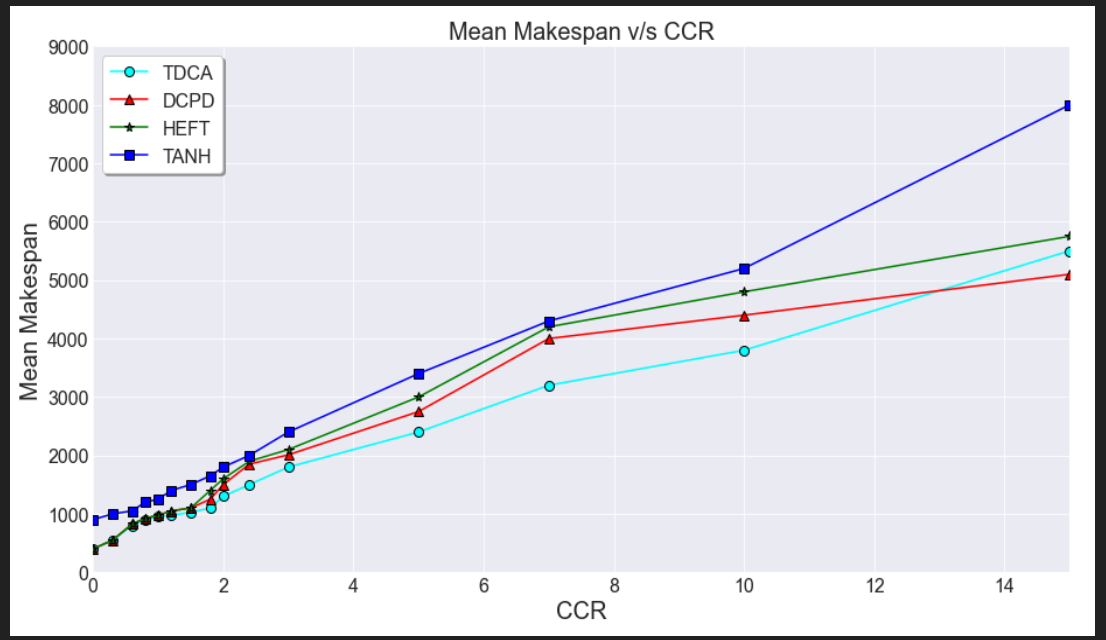
Fig. 5. Key Parameters

Comparative Results:

First, the performance of ***TDCA*** is compared against the existing scheduling algorithms by varying the value of ***number of nodes (tasks)*** in the DAG. The ***number of nodes*** are varied from 30 to 100. All algorithms scale linearly with the number of nodes. ***TDCA*** produces the smallest mean makespan.

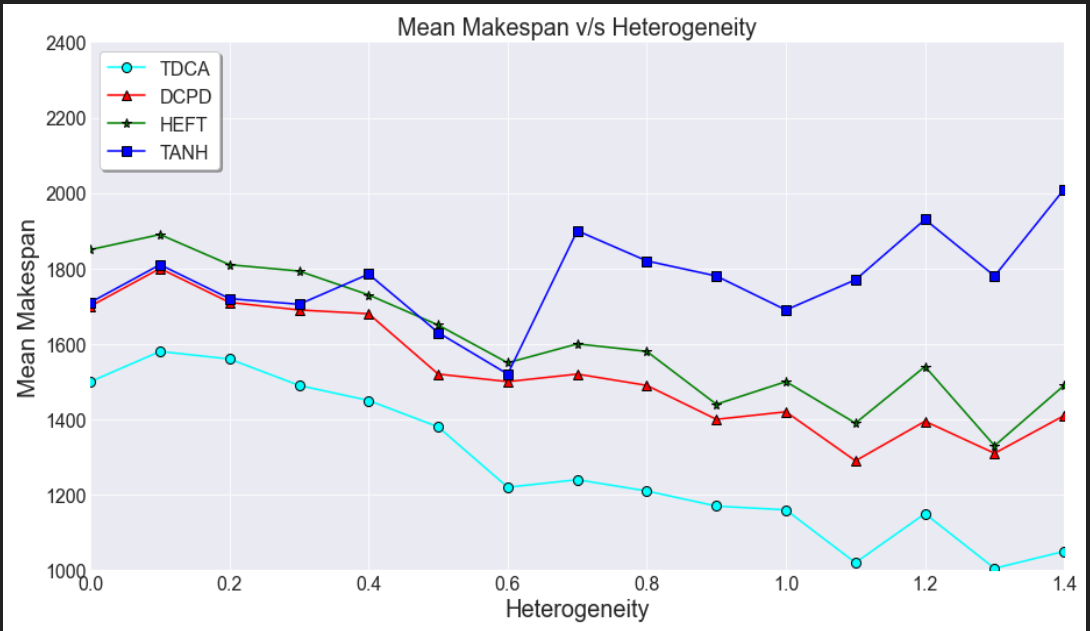


Secondly, the impact ***of Communication-Computation Ratio (CCR)*** is studied on all four algorithms. When ***CCR*** is small, the communication cost between tasks can be negligible and vice-versa. ***TDCA*** outperforms other three in range 0.3 to 13. In general, ***TDCA*** is the best performing algorithm.

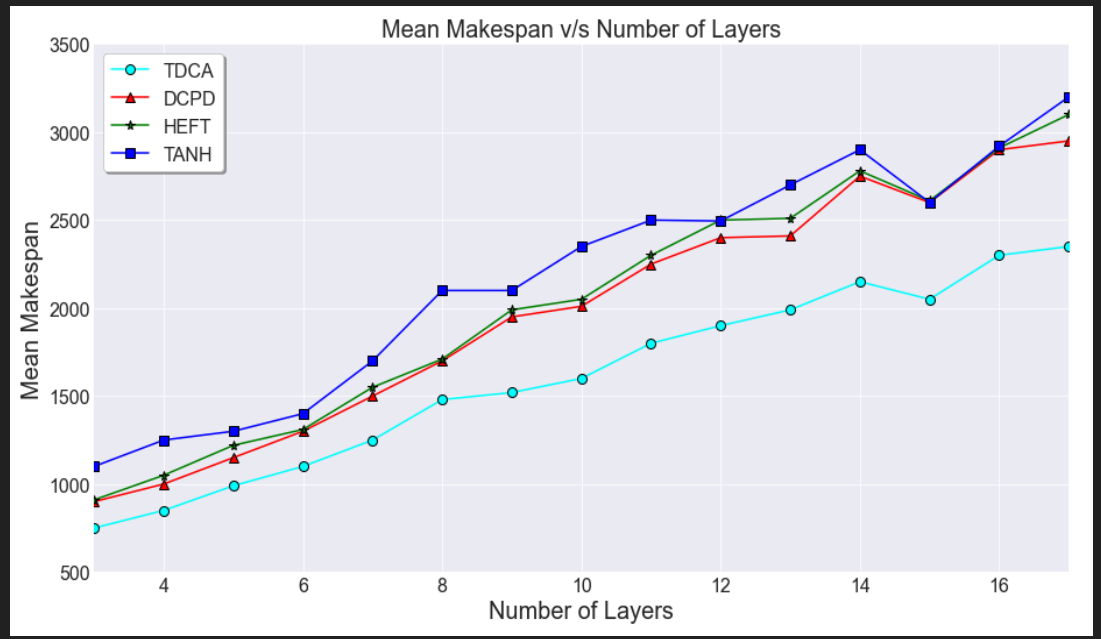


Cont'd:

Next, the impact of ***heterogeneity parameter (h)*** is studied on all four algorithms. The larger the value of ***h*** is, the more heterogeneous the system is. The results show the mean makespan of different algorithms on different values of parameter ***h***. In all ranges, ***TDCA*** produces shortest mean makespan.



Lastly, the impact of ***number of layers (L)*** in the DAG is studied. Makespan increases linearly with the number of layers, which is according to the expectations since the large ***number of layers*** indicate low parallelism. ***TDCA*** performs better than any other algorithm.



Conclusion:

- In this project, we have implemented ***TDCA*** for the DAG task scheduling problem in the ***heterogeneous distributed environment***. The proposed algorithm utilizes the duplication task more thoroughly involving ***parameter calculation, task duplication,*** and ***task merging***. Several ***key parameters*** such as ***Earliest Completion Time (EST)*** and ***Critical Predecessor*** are redefined.
- ***TDCA*** aims at improving the ***initial clustering***. Existing algorithms typically duplicate the predecessors of a task when producing initial task clusters . On the other hand, ***TDCA*** will also consider waiting for transferring the results of a predecessor task from other clusters, which may improve the quality of the initial clusters.
- The ***Duplication*** and ***Merging phases*** are considered to reduce the makespan. During the duplication phase, the order of finding the candidate duplication positions is determined. By analyzing the effect of task duplication, it is concluded that ***chain reactions*** exists. After one round of scanning, all ***duplication candidates*** are classified into ***effective*** and ***ineffective candidates***. In the next round, these ineffective candidates can become effective.

TDCA explicitly outperforms the baseline algorithms without increasing the computation complexity.

Learnings:

In this project, we learned many things. We can break down our learnings into two broad categories:

- ***Technical Learnings:*** Firstly, we gained a large amount of knowledge regarding ***Heterogeneous Systems***. In our course, we are mainly concerned with the ***Homogeneous Systems*** for the convenience. We learned the ***intricacies*** and ***complexities*** of ***Heterogeneous Systems*** in which every processor has its own ***architecture*** and ***computation power***. We also understood the basis of ***Classification of Scheduling Algorithms***. Next, we studied about the ***Types of Priority Levels such as T-Level, S-Level and B-Level.*** Also, during the study of ***Priority Levels***, we were required to study about the ***Critical Path Method (CPM)*** as well. Writing the ***function for Schedule Makespan*** pushed us to ***derive our own method of calculating the makespan*** as it was not mentioned anywhere else. This project also taught us the working of proposed ***TDCA Scheduling Algorithm*** thoroughly. We also learned the impact of ***various key parameters*** in terms of performance improvement.
- ***General Learnings:*** Firstly, we learned how to ***properly read research papers*** and ***how to correlate various research papers*** written in very different timeframes. It also taught us how to ***break down very large and complex algorithms*** into ***smaller as well as simpler algorithms*** that can be understood and implemented by people from different countries and domains. We learned how to ***write modular codes*** in which ***debugging*** can be done easily and its application can be presented in a more sophisticated manner. Above all, we learned a lot about ***team work, time management*** and got a gist of how it feels to ***work on a project that is distributed over months***.

THANK YOU