CO-SCHEDULING OF HYBRID TRANSACTIONS ON MULTIPROCESSOR REAL-TIME DATABASE SYSTEMS

INTRODUCTION

A ***Real-Time Database System (RTDBS)*** is a system which uses real-time processing to handle workloads of database whose state is constantly changing. They differ from traditional databases containing persistent data, mostly unaffected by time. RTDBS are an extension with additional power of yielding reliable responses. They use timing constraints that represent a certain range of values for which the data remains valid. This range is called ***temporal validity***.

To satisfy the key characteristic of RTDBSs, there are two types of transactions

- ***Control Transactions***: They are used to access and use the data in systems.
- ***Update Transactions***: They are used for monitoring real-time data objects and updating their values.

In this project, we have considered the problem of ***Co-Scheduling of Hybrid Transactions on Multiprocessor Systems*** in such a manner that workload of processors is minimized. Two intuitive partitioning scheduling methods namely: ***Half-Half based Partitioned Scheduling (HH-P)*** and ***Partitioned Scheduling for Hybrid Transactions (P-HT)*** are implemented. Then, we have implemented two improvement strategies namely: ***Efficient Partitioned scheduling for Hybrid Transactions (EP-HT)*** and ***Improved EP-HT scheduling (IEP-HT).*** The EP-HT improves the efficiency of P-HT and IEP-HT improves the acceptance ratio of EP-HT.

LITERATURE SURVEY

There has been a lot of work for maintaining real-time data freshness. Song and Liu studied the performances of the two-phase locking and the optimistic algorithm. Xiong and Ramamritham proposed the MoreLess scheme. Li et al. proposed an EDF-based update transaction scheduling method. Lundberg proposed the first update transaction scheduling for multiprocessor environments.

All existing methods focus on the Uniprocessor Co-Scheduling or the Update Transaction Multiprocessor Scheduling, while the paper taken up by us is the first to propose Co-Scheduling of Hybrid Transactions on Multiprocessor Systems.

Li et al. proposed ***Minimum Deadline Calculation (MDC)*** method. The MDC algorithm computes the minimum deadline for each update transaction in a hybrid real-time transaction set. It ranks the update transactions by non-increasing order of their validity and calculates the minimum deadlines.

```
def MDC(Tu, T):
    Tu_dash = sorted(Tu, key = operator.itemgetter(3))
    for i, elem in enumerate(Tu_dash):
        flag = 0
        t = elem[0]
        temp = -1
        while t <= (elem[3]//2):
            Delta = t - h_func1(t, T) + h_func2(elem, t)
            if Delta <= elem[0] and flag == 0:
                temp = t
            elif Delta > elem[0]:
                flag = 1
            t += 1

        if flag == 1 or temp != -1:
            Tu_dash[i][1] = temp
            Tu_dash[i][2] = Tu_dash[i][3] - temp
        else:
            return ([])
            sys.exit("The Transaction Set cannot be Scheduled: 1")
    return Tu_dash
```

Fig. 1: Code Cell for MDC Algorithm

INTUITIVE PARTITIONING METHODS

HH-P

Half-Half is a deadline and period deriving method under which the deadline and the period of each update transaction are always set to half of valid interval length. By combining HH and EDF, we can get an intuitive partitioned co-scheduling method, HH-P, for hybrid transaction sets in multiprocessor systems.

P-HT

P-HT reduces the total workload of processors of HH-P by relaxing the schedulability condition and reassigning the deadlines and periods of update transactions. HH is used to initialize the deadlines and transactions are sorted by increasing order of their deadlines. Transactions are assigned to processors before executing MDC algorithm to reassign the deadlines for update transactions. Finally, EDF schedulability test is performed.

```
def HH_P(n, m, M, T, Tu, Tc):
    for i in range(len(Tu)):
        Tu[i][1] = Tu[i][3]//2
        Tu[i][2] = Tu[i][3]//2
    T = sorted(T, key = operator.itemgetter(1, 3))
    for i in range(n):
        j = 0
        for j in range(m+1):
            if j == m:
                return (0, 0, [], [], [], [])

            M[j].append(T[i])
            Con1 = (utilization(M[j])) <= 1
            Con2 = Theorem1(M[j])
            if Con1 and Con2:
                break

            M[j].pop()
    return n, m, M, T, Tu, Tc
```

Fig. 2: Code Cell for HH-P Algorithm

IMPROVED PARTITIONING METHODS

EP-HT

EP-HT improves the efficiency of P-HT. It works similarly to P-HT till sorting phase. Then, if a control transaction is encountered , it is assigned to the processor such that the set of transactions on that processor satisfy some conditions. Similarly, update transactions are assigned based on some different conditions. Then, MDC and EDF-Test are performed on schedules.

IEP-HT

IEP-HT improves acceptance ratio of EP-HT. It calculates final deadlines and periods for update transactions once they are assigned to processors. It treats update transactions as control transactions when their deadlines and periods have be assigned. Whenever an update transaction is dispatched to a processor successfully, the MDC is used to calculate the deadline.

```
def IEP_HT1(n, m, M, T, Tu, Tc):
    for i in range(n):
        j = 0
        for j in range(m+1):
            if j == m:
                return (0, 0, [], [], [], [])
            T_dash = M[j].copy()
            M[j].append(T[i])
            M_Tu = []
            M_Tc = []
            for elem in M[j]:
                if elem[4][0] == 'U':
                    M_Tu.append(elem)
                else:
                    M_Tc.append(elem)
            Con1 = (utilization(M[j])) <= 1
            Con2 = Theorem5(T_dash, T[i])
            if Con1 and Con2:
                if T[i][4][0] == 'U':
                    M_Tu = MDC(M_Tu, M[j])
                    M[j] = M_Tu + M_Tc
                    if Theorem1(M[j]):
                        break
                    else:
                        return (0, 0, [], [], [], [])
                else:
                    break
            M[j].pop()
    return n, m, M, T, Tu, Tc
```

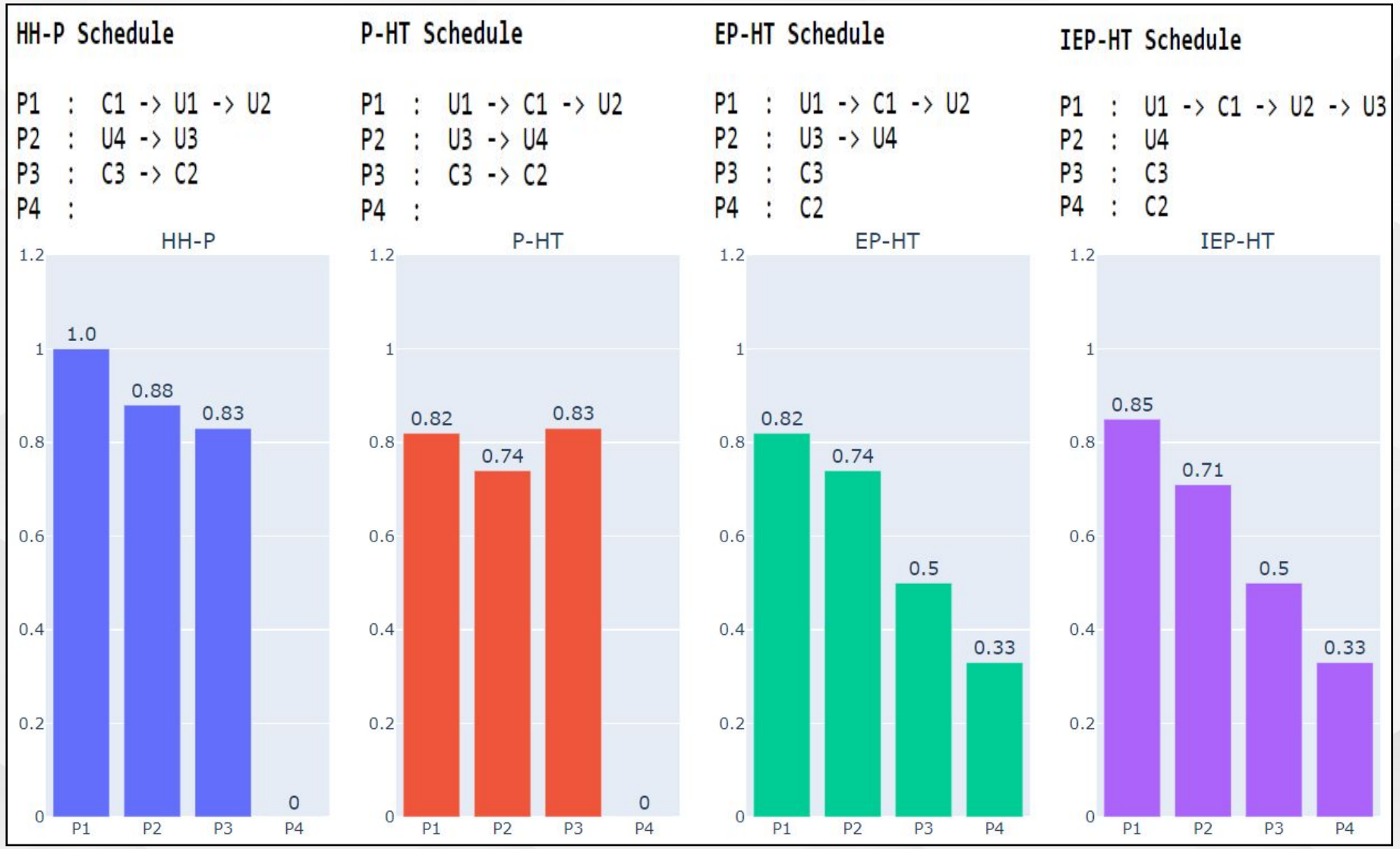
Fig. 3: Code Cell for IEP-HT Algorithm

RESULTS

In order to obtain the schedules, we have taken the input as follows:

- **Number of Control Transactions:** 3
- **Number of Update Transactions:** 4
- **Number of Processors:** 4

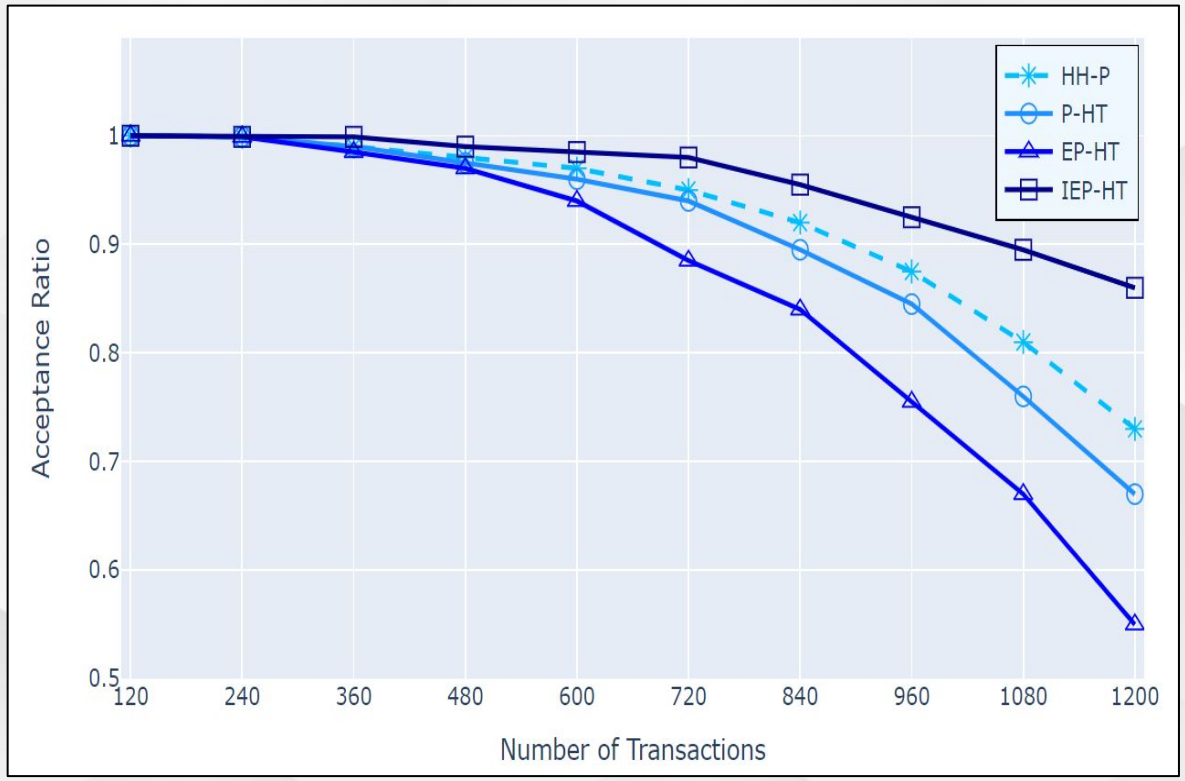
The figure on the right shows the Schedule and Workload of each processor obtained using all four methods. We can clearly see that ***HH-P performs the worst*** since it gives highest workload for the three processors and does not even use P4. ***P-HT performs slightly better than HH-P*** but still does not take advantage of P4. ***EP-HT performs the best and is closely followed by IEP-HT.*** Both take advantage of all 4 processors.

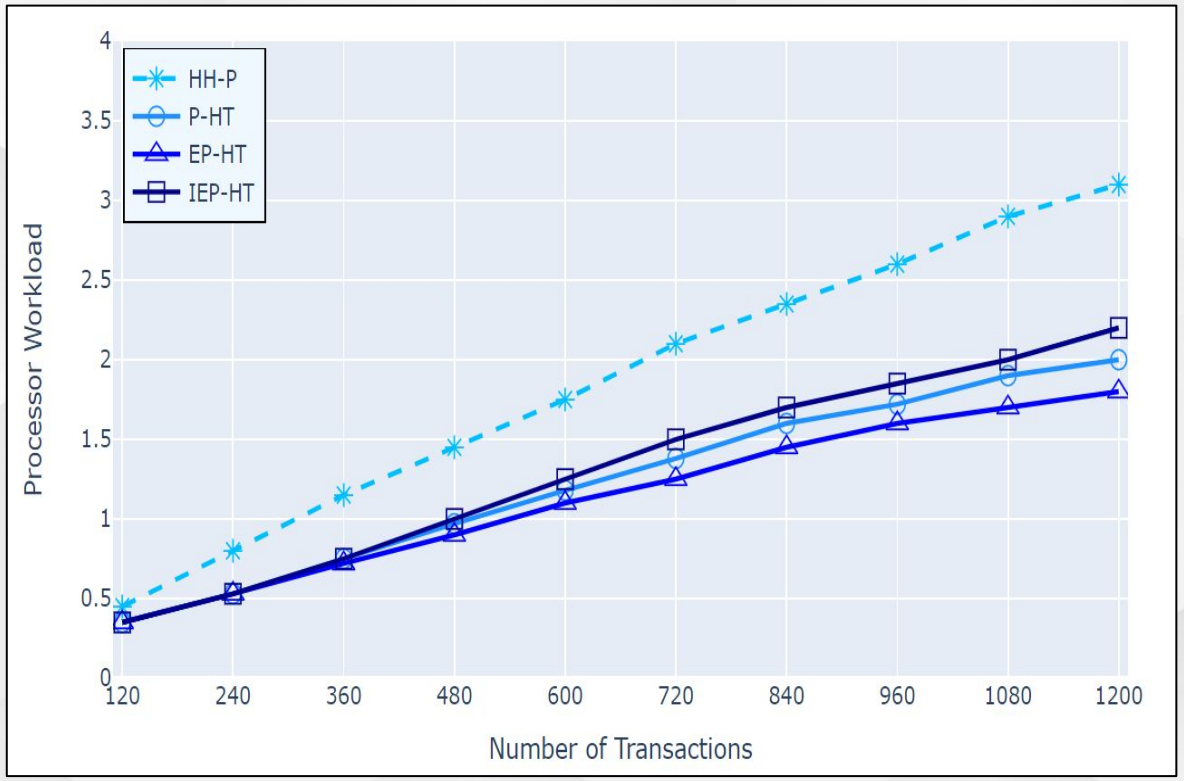


COMPARATIVE RESULTS

The ***Acceptance Ratios*** of all methods decrease with the growth of ***Number of Transactions (N). IEP-HT always has a better performance than the others.*** For N = 1200, IEP-HT improves the acceptance ratios of HH-P, P-HT and EP-HT for about 17%, 28% and 56%, respectively. ***EP-HT has the worst performance on acceptance ratio.*** Acceptance Ratio of P-HT is lower than that of HH-P.

EP-HT has the best performance of all other methods. The performances of P-HT, EP-HT and IEP-HT are always better than that of HH-P. At N = 1200, the ***Processor Workload*** under EP-HT is only about 58%, 91% and 84% of those under HH-P, P-HT and IEP-HT, respectively. EP-HT assigns transactions to processors more averagely than P-HT and IEP-HT.



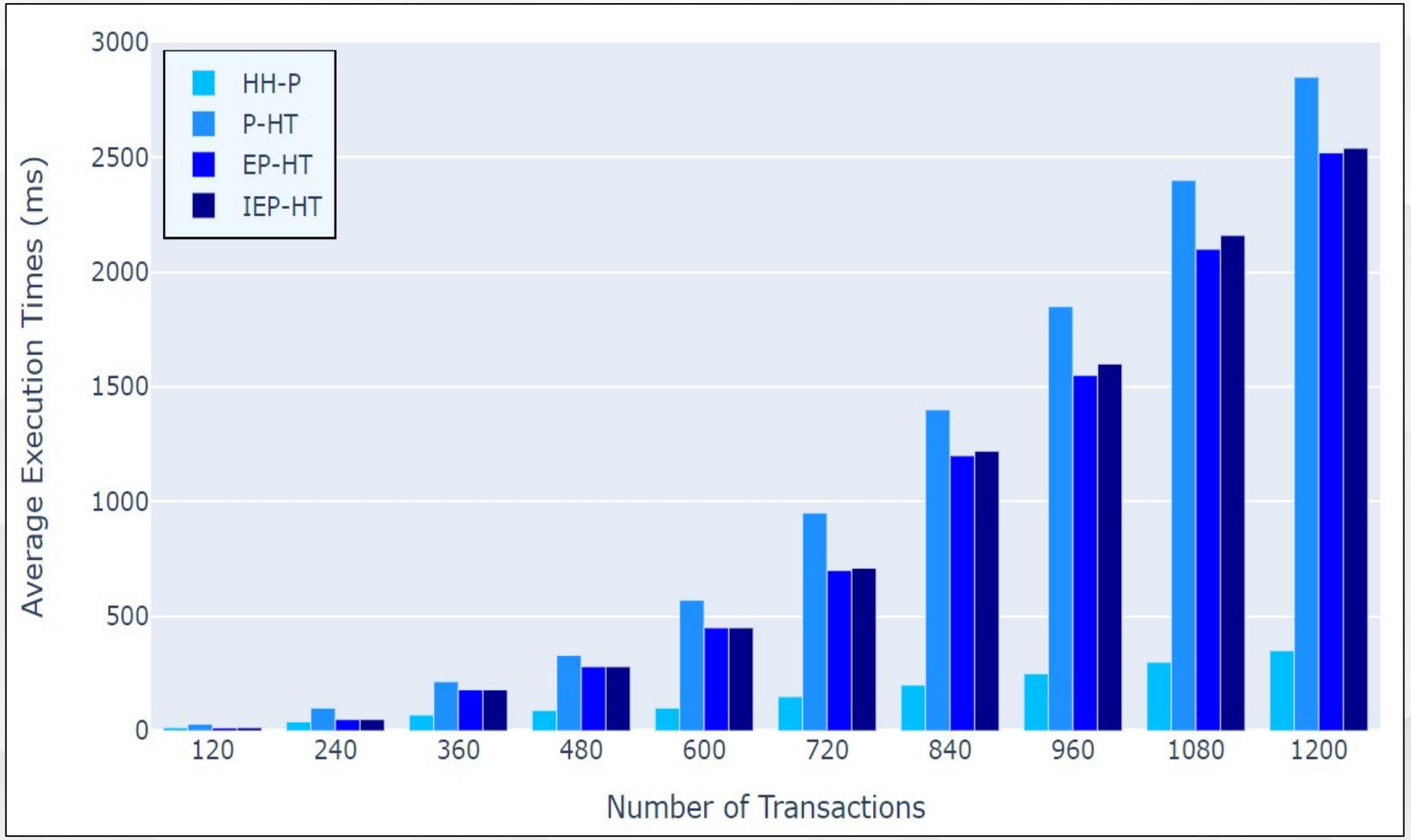


CONT'D

The figure on the right shows that the Average Execution Times of all methods increase with the growth of N.

HH-P always has the best Average Execution Times. It is because MDC is executed in P-HT, EP-HT and IEP-HT, while HH-P assigns the deadline and the period of each update transaction to half of valid interval length directly. The average execution time of HH-P is only about 10%, 12% and 12% of those of P-HT, EP-HT and IEP-HT, respectively, at N = 1200.

EP-HT has almost the same Average Execution Times as IEP-HT.



CONCLUSION

In this project, we have implemented Four Hybrid Transaction Scheduling Algorithms for Multiprocessor Systems namely:

- Half-Half based Partitioned Scheduling (HH-P)
- Partitioned Scheduling for Hybrid Transactions (P-HT)
- Efficient Partitioned Scheduling for Hybrid Transactions (EP-HT)
- Improved EP-HT Scheduling (IEP-HT)

The proposed algorithms are the first attempt for the Multiprocessor Co-Scheduling Methods of Hybrid Transactions. HH-P and P-HT are two intuitive partitioned scheduling methods proposed by developing some existing scheduling methods to multiprocessor environment. For improving the efficiency of these two methods, EP-HT and IEP-HT are two improved methods which reduce the time for testing the schedulability of transaction sets. Theoretical analysis proves that the resource augmentations of the two improvement strategies can reach $3-1/m$. The results obtained upon implementing the four algorithms are used to evaluate the performance of the proposed methods in terms of workload of processors.

IEP-HT has the best overall performance among all the scheduling algorithms on the basis of Acceptance Ratio, Processor Workload and Average Execution Times.

LEARNING

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 ***Real-Time Database Systems***. We learned the intricacies and complexities of scheduling on Multiprocessor Systems in which it is very important to distribute the workload uniformly among all processors. We came to know about the ***EDF-Schedulability Test*** which is used to validate a schedule. During the implementation, we were required to study other research papers based on ***Minimum Deadline Calculation (MDC) Algorithm***. We also learnt about various conditions involved with different types of Transactions namely: ***Control Transactions*** and ***Update Transactions***. This project also gave us an insight of how small tweaks can improve performance in certain parameters drastically. For example, applying MDC algorithm after assigning each transaction to a processor improved the acceptance ratio by a very large margin as compared to applying it after assigning all transactions to processors.

- ***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***. 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