Faster Algorithms for Real Time Data Base Updations using Deferrable Scheduling

1Rajesh Babu. Movva, 2A.P.N.G. Krishna and 3Bomma Manikanta

1Asistant Professor, 2Student, 3Student
E-mail: mrb.csebec@gmail.com, phalgunanagani@gmail.com, manikantabommal@gmail.com

Abstract
The deferrable scheduling algorithms are very impressive for minimizing real-time update transaction workload but suffer from its on-line scheduling overhead. In this paper, we propose two enlarged versions of deferrable scheduling fixed priority algorithms to reduce the on-line scheduling overhead. These algorithms produce a hyper period which can use endless times by satisfying the temporal constraints. The first one is Deferrable Scheduling with hyper period by Schedule Construction. It searches the deferrable scheduling fixed priority schedule for a hyper period. The second one is deferrable Scheduling with hyper period by Schedule Adjustment. It adjusts the deferrable scheduling fixed priority schedule in an interval to form a hyper period. Both deferrable Scheduling with hyper period by Schedule Construction and deferrable Scheduling with hyper period by Schedule Adjustment can overcome the drawbacks in deferrable scheduling fixed priority algorithms, and deferrable Scheduling with hyper period by Schedule Adjustment works better than deferrable Scheduling with hyper period by Schedule Construction by performing more number of update transactions in the system.

Keywords: Real-Time databases, Temporal validity constraint, Fixed priority scheduling, Deferrable scheduling.

Introduction
Real-time embedded systems are important components of many time-critical applications that require timely processing of massive amount of real-time data. The correct functioning of a real-time embedded system depends not only on meeting real-time constraints of application jobs, but also on the accuracy of real-time data values sampled from real-world entities. Examples of real-time data include sensor data in sensor networks, positions of aircrafts in air traffic control systems, and vehicle velocity in adaptive cruise control applications. To provide better management of sampled real-time data and to support effective processing of application jobs, real-time data are typically managed by a real-time database system (RTDBS), passage of time since the status of the corresponding entity in the real-world may change continuously. Sensor update transactions should constantly sample real-world data values and install them into the RTDBS.

One efficient way to determine the correctness of real-time data in a RTDBS is to define a validity constraint or age constraint, which determines a validity interval length for each real-time data object. A real-time data value is only valid within its validity interval. For reliability reasons, real-time embedded systems require continuous generation of update transactions to refresh real-time data objects regardless of how much the status of the corresponding real-world entities have been changed. To meet this constraint, it is important to produce a schedule for all update transactions such that for any consecutive updates of a real-time data object, the next update is completed before the previous validity interval expires. Thus one of the crucial issues in the design of real time embedded systems is to schedule the update transactions efficiently to maintain the validity of real-time data while minimizing the total update transaction workload.

Existing Methods
Most of the previous work in update transaction scheduling assumed a periodic transaction model. The update problem for periodic update transactions consists of two parts.

1. the determination of the sampling periods and deadlines of update transactions; and
2. the scheduling of up-date transactions.

One of the proposed approaches is the Half-Half (HH) scheme in which an update transaction has a fixed period that is half of the validity interval. If the set of transactions is schedulable in HH, the validity constraints of the corresponding real-time data objects can also be guaranteed. Similarly to HH, More-Less (ML) is another periodic approach in which up-date transactions are scheduled based on the deadline monotonic algorithm Compared to HH, ML can guarantee the validity of real-time data objects with less update transaction workload. Recently, the DS-FP algorithm was proposed to further reduce the total update transaction work-load. The main idea of DS-FP is to adopt sporadic task model instead of the periodic model.

Compared to ML, DS-FP increases the separation of two consecutive trans-action jobs by releasing an update transaction job as late as possible based on the sampling time of its previous job. Both theoretical analysis and experimental results have demonstrated that DS-FP outperforms HH and ML significantly in reducing the update transaction workload while still maintaining the real-time data validity. One major problem of DS-FP is its time complexity for on-line schedule computation. The variation in its run-time overhead leads to unpredictability of the system performance.
A real-time data object \( X_i \) at time \( t \) is temporally valid (or absolutely consistent) if, for its \( j \)th update finished latest before \( t \), the sampling time \( r_{i,j} \) plus the validity interval \( V_i \) of the data object is not less than \( t \), i.e., \( r_{i,j} + V_i \geq t \).

A data value for real-time data object \( X_i \) sampled at any time \( t \) will be valid up to \( (t + V_i) \). The actual length of the temporal validity interval of a real-time data object is application dependent. One of the important design goals of RTDBS is to guarantee that real-time data remain fresh, i.e., they are always valid. We assume that the network delay for a sensor update transaction job to be sent from a sensor to the RTDBS (i.e., jitter between sampling time at the sensor and release time at the RTDBS) is zero for convenience of presentation.

**More Les**
ML adopts the periodic task model for sensor update transactions whose derived deadlines are not larger than their periods. Consider synchronous transactions whose first jobs all start at time 0. A time instant after which a transaction job has the worst-case response time is called a critical instant, e.g., time 0 is a critical instant for all the transactions with deadlines no larger than their periods if those transactions are synchronous. Note that we only consider synchronous transactions. In ML, there are three constraints to follow for transactions \( t_i \) (\( 1 \leq i \leq m \)).

- Validity constraint: the sum of the period and relative deadline of transaction \( t_i \) is always less than or equal to \( V_i \), i.e., \( P_i + D_i \leq V_i \).
- Deadline constraint: the period of an update transaction is assigned to be more than or equal to half of the validity length of its updated object, while its corresponding relative deadline is less than or equal to half of the validity length of the same object. For \( t_i \) to be schedulable, \( D_i \) must be greater than or equal to \( C_i \), the worst case execution time of \( t_i \), i.e., \( C_i \leq D_i < P_i \).
- Schedulability constraint: for a given set of update transactions, the Deadline Monotonic scheduling algorithm is used to schedule the transactions.

**DS-FP**
DS-FP ML is pessimistic on the deadline and period assignment. This is because it uses a periodic task model that has a fixed period and relative deadline for each transaction, and the relative deadline \( D_i \) is equal to the worst-case response time of the transaction. According to the validity constraint in ML, the larger the deadline \( D_i \), the smaller the period \( P_i \). In order to increase the separation of two consecutive jobs (and thus reduce the sensor update workload), DS-FP adaptively derives the relative deadline and separation of one job from its previous job and preemptions from higher priority transactions. Given release time \( r_{i,j} \) of job \( J_{i,j} \) and deadline \( d_{i,j} \) of job \( J_{i,j} \) \( (j \geq 0) \),

\[
d_{i,j+1} = r_{i,j} + V_i
\]

(1)
guarantees that the validity constraint can be satisfied, as depicted in Fig. 1. Correspondingly, the following equation follows directly from (1):

\[
(r_{i,j+1} - r_{i,j}) + (d_{i,j+1} - r_{i,j+1}) = V_i
\]

(2)

If \( r_{i,j+1} \) can be shifted onward to \( r'_{i,j+1} \) along the time line in Fig. 1, it does not violate (2). After the shift, temporal validity can still be guaranteed as long as \( J_{i,j+1} \) is completed by its deadline \( d_{i,j+1} \). The idea of DS-FP is to defer the sampling time (i.e., release time), \( r_{i,j+1} \), of \( J_{i,j+1} \)’s next job as late as possible while still guaranteeing the validity constraint. According to the fixed priority scheduling theory, \( r_{i,j+1} \) in DS-FP can be derived backwards from its deadline \( d_{i,j+1} \) as follows:

\[
(r_{i,j+1}) = (d_{i,j+1} - R_{i,j+1})(r_{i,j+1}, d_{i,j+1})
\]

(3)

\[
(R_{i,j+1})(r_{i,j+1}, d_{i,j+1}) = \theta(t_{i,j+1}, d_{i,j+1}) + c_i
\]

(4)

where \( \Theta(a, b) \) denotes the total cumulative processor demands made by all jobs of higher-priority transaction \( t_{k} \) (\( 1 \leq k \leq i - 1 \)) during time interval \( [a, b] \), and \( R_{i,j+1}(r_{i,j+1}, d_{i,j+1}) \) (or \( R_{i,j} \) for simplicity in DS-FP) the response time of \( J_{i,j+1} \) deriving backwards from its deadline \( d_{i,j+1} \). Similarly to ML, DS-FP also assigns priorities to transactions according to SVF. Readers are referred to the Appendix for the details of the DS-FP algorithm. Now we summarize the algorithm as follows. First we set \( r_{i,0} = 0 \), \( 1 \leq i \leq m \). The highest priority job among the outstanding jobs is always scheduled first. It is only preempted when a new job with higher priority is ready.

![Fig 1. illustration of DS-FP scheduling.](image)

As soon as a job \( J_{i,j} \) is completed, we derive the \( r_{i,j+1} \) of its next job according to above calculations. The algorithm fails when a job misses its deadline. Otherwise it keeps running. It is proved that any task set that is scheduled by ML is also scheduled by DS-FP. The EDL algorithm proposed in Chetto and Chetto processes tasks as late as possible based on the Earliest Deadline algorithm. EDL assumes that all deadlines of tasks are given whereas DS-FP and DETH algorithms derive deadlines dynamically. The validity constrained scheduling, e.g., ML, DS-FP, and DETH algorithms, are different from the distance constrained scheduling, which guarantees an upper bound to the finishing times of two consecutive instances of a task.

Next, we present two deferrable Scheduling with hyper period (DESH) algorithms for constructing periodic schedules off-line from the DS-FP algorithm so that the on-line scheduling time complexity can be reduced. Note that this will undoubtedly increase the space overhead for keeping the DESH schedules. But the space overhead can be kept reasonably low. Our DESH algorithms satisfy the following properties:
• Property 1 A schedule satisfies the validity constraint.
• Property 2 The on-line scheduling time complexity is \(O(1)\).

Proposed Methods
We propose two Deferrable Scheduling with Hyper period (DESH) algorithms, which construct the hyper period schedule off-line and reduce the on-line scheduling time complexity to \(O(1)\). The key contributions of our work include the followings.

1. To reduce on-line scheduling overhead, our first algorithm, named Deferrable Scheduling with Hyper period Construction (DESH-SC), searches the hyper period of the set of update transactions so that the transactions can be scheduled by repeating the hyper period schedule. However, the hyper periods found by DESH-SC are exponentially long, which could incur significant space overhead for maintaining the hyper period information for on-line scheduling.

2. Our second algorithm, named Deferrable Scheduling with Schedule Adjustment (DESH-SA), adjusts the DS-FP schedule in an interval such that the adjusted schedule can be repeated infinitely. DESH-SA improves DESH-SC on producing much shorter hyper periods and accommodating significantly more update transactions.

3. Our experimental results demonstrate that both DESH-SC and DESH-SA can reduce scheduling overhead of DS-FP, and DESH-SA outperforms DESH-SC by accommodating significantly more update transactions in the system.

Deferrable Scheduling with Hyper period: Schedule Construction (DESH-SC)
In this subsection, we present Deferrable Scheduling with Hyper period based on Schedule Construction (DESH-SC), a DS-FP based algorithm that can reduce the online scheduling overhead. The basic idea of DESH-SC is to search for an interval of DS-FP schedule, the hyper period, that could be repeated infinitely without violating the validity constraint. Note that DESH-SC could return without finding a hyper period. The DESH-SC algorithm consists of two parts: an algorithm for finding the hyper period off-line and an algorithm for scheduling transactions on-line. The latter is trivial once a hyper period is found because it only needs to repeat the hyper period schedule.

For a DS-FP schedule and a time period \([a, b]\), we say \([a, b]\) is a hyper period for the transaction set if for all transactions \(i (1 \leq i \leq m)\), the following schedule satisfies \(\tau_i\)'s validity constraint: it is the same as the DS-FP schedule from time 0 to te. From \(t_e\) onward, it repeats the DS-FP schedule in \([a, b]\) to infinity. Please note that \(t_s\) and \(t_e\) do not need to be idle time points in DESH-SA, which is different from the requirements of \(t_s\) and \(t_e\) in DESH-SC.

**Theorem 1.** \([t_s, t_e]\) is a hyper period in DESH-SC if for all \(t_i (1 \leq i \leq m)\) the following conditions hold.

1. \(t_s\) and \(t_e\) are CPU idle time points.
2. \(t_s > V_i\).
3. \(t_e\) is scheduled at least once in \([t_s, t_e]\).
4. \(I(t_s, t_e) \geq I(t_s, t_e)\), where function \(I(t, t)\) is defined as the time distance between \(t\) and \(t_i\)'s latest release time before \(t\).

Proof: Given any \(t_i (1 \leq i \leq m)\), we first prove in the hyper period schedule that the distance of the finish time of the first \(t_i\) job after \(t_e\) and the release time of its latest job before \(t_e\) satisfies the validity constraint. Note that the first job after \(t_e\) repeats the first job in \([t_s, t_e]\). Because \(t_s > V_i\), the first job of \(t_i\) in \([t_s, t_e]\) must finish by \((t_i - 1 (t_i, t_i) + V_i\), in other words, by \(V_i - I(t_i, t_i)\) after \(t_e\). Since \([t_s, t_e]\) is repeated after \(t_e\), the first job of \(t_i\) after \(t_e\) also finishes by \(V_i - I(t_s, t_i)\) after \(t_e\). The distance between the finish time of its first job in the second hyper period \([2t_s - t_e]\) and the release time of its latest job in the first hyper period \([t_s, t_e]\) is no longer than:

\[
I(t_s, t_e) + (V_i - I(t_s, t_i)) = V_i + I(t_s, t_i) 
\]

Similarly, we can prove that the distance between the finish time of its first job in the \((k + 1)th\) (\(k = 1, 2, \ldots\)) hyper period and the release time of its latest job in the \(k\)th hyper period is no longer than \(V_i\). Thus, the jobs of \(t_i\) satisfy the validity constraint.

The idea is presented in Algorithm 1. In the algorithm, we continuously push \(t_e\) of idle periods into a queue \(Q\) as possible candidates for \(t_s\) of a hyper period. For each subsequent idle period, we check its \(t_s\) against each \(t_e\) saved in \(Q\) to see if they form a hyper period. If the hyper period is found, we could then further increase \(t_e\) as long as \([t_s, t_e]\) still satisfies the conditions in theorem 1. The increase cannot exceed \(I(t_s, t_e) - I(t_s, t_i)\) for any transaction \(t_i\), \(1 \leq i \leq m\). We define \(w\) to be the minimum of \(I(t_s, t_e) - I(t_s, t_i)\) in the algorithm.

**Proof:**
Given any \(t_i (1 \leq i \leq m)\), we first prove in the hyper period schedule that the distance of the finish time of the first \(t_i\) job after \(t_e\) and the release time of its latest job before \(t_e\) satisfies the validity constraint. Note that the first job after \(t_e\) repeats the first job in \([t_s, t_e]\). Because \(t_s > V_i\), the first job of \(t_i\) in \([t_s, t_e]\) must finish by \((t_i - 1 (t_i, t_i) + V_i\), in other words, by \(V_i - I(t_i, t_i)\) after \(t_e\). Since \([t_s, t_e]\) is repeated after \(t_e\), the first job of \(t_i\) after \(t_e\) also finishes by \(V_i - I(t_s, t_i)\) after \(t_e\). The distance between the finish time of its first job in the second hyper period \([2t_s - t_e]\) and the release time of its latest job in the first hyper period \([t_s, t_e]\) is no longer than:

\[
I(t_s, t_e) + (V_i - I(t_s, t_i)) = V_i + I(t_s, t_i) 
\]

Similarly, we can prove that the distance between the finish time of its first job in the \((k + 1)th\) (\(k = 1, 2, \ldots\)) hyper period and the release time of its latest job in the \(k\)th hyper period is no longer than \(V_i\). Thus, the jobs of \(t_i\) satisfy the validity constraint.

The idea is presented in Algorithm 1. In the algorithm, we continuously push \(t_e\) of idle periods into a queue \(Q\) as possible candidates for \(t_s\) of a hyper period. For each subsequent idle period, we check its \(t_s\) against each \(t_e\) saved in \(Q\) to see if they form a hyper period. If the hyper period is found, we could then further increase \(t_e\) as long as \([t_s, t_e]\) still satisfies the conditions in theorem 1. The increase cannot exceed \(I(t_s, t_e) - I(t_s, t_i)\) for any transaction \(t_i\), \(1 \leq i \leq m\). We define \(w\) to be the minimum of \(I(t_s, t_e) - I(t_s, t_i)\) in the algorithm.

**Algorithm 1.** Search Hyper period:

Input: A DS-FP schedule, a utilization limit \(U_{\text{Max}}\), and a time limit \(T_{\text{Max}}\).

Output: The hyper period with utilization \(U_{\text{Max}}\).

1. Initialize the hyper period: \(U_{\text{Max}} = 1.001\); // Initialization of hyper period utilization.
2. \(t_s = \max \{V_i | 1 \leq i \leq m\};\)
3. \(t_e = \text{first CPU idle period after } t_s;\)
4. Append \(t_e\) to \(Q;\) // \(Q\) is a FIFO queue of \(t_e\).
5. While \((U > U_{\text{Max}})\) do
6. \(t_e = \text{next CPU idle period after } t_e;\)
7. // \(T_{\text{Max}}\) is the maximum time to search.
8. if \((t_e > T_{\text{Max}})\) then return failure; endif
9. \(t_s = t_e;\)
10. for \(t_e = \text{first in } Q\) to \(t_s\) in \(Q\) do
11. if \((t_e, t_s)\) satisfies Condition 4) then
12. Signal that a hyper period exists.
13. \}
Deferrable Scheduling With Hyper Period: Schedule Adjustment (DESH-SA)

In this subsection, we present Deferrable Scheduling with Hyper Period (DESH-SA), a DS-FP based algorithm that can reduce the online scheduling overhead while achieving processor utilization close to that of DS-FP. By schedule adjustment, we mean changing release times and deadlines of jobs. The basic idea of DESH-SA is to construct a hyper period schedule SH off-line for T, a set of valid transactions. Suppose the first hyper period of SH has length $\Vert S_H \Vert$. If the first hyper period of SH can be constructed by adjusting the DS-FP schedule in the time interval $[0, \Vert S_H \Vert]$ the complete SH schedule is constructed by repeating the first hyper period of SH infinitely every $\Vert S_H \Vert$ time units.

Thus, similarly to DESH-SC, the DESH-SA algorithm consists of two parts: an algorithm for constructing the hyper period off-line and an algorithm for scheduling transactions on-line. We next describe how the first hyper period schedule of SH in the interval $[0, \Vert S_H \Vert]$ is derived from the schedule of DS-FP. Given time $t_e > 0$, note that a DS-FP schedule in the interval $[0, t_e]$ can be constructed off-line. Assume that jobs $J_{i,j}$ and $J_{m,j}$ of $t_i (k_i \geq 1 \& 1 \leq i \leq m)$ satisfy the following condition for $t_e$:

$$g(n_1t_e, t) = g(n_2t_e, t)$$

where $g(n_t, t)$ is a function returning a pair of integers $i, k$, which indicates that the $k^{th}$ job of $t_i$ in the $n^{th}$ hyper period is active at time $t$ (i.e., at time $n_t + t$). Equation (7) implies that any two hyper periods have the exactly same schedule.

$$g(n_t, t) = \begin{cases} \langle i, j - n(k_i + 1) \rangle, & \text{the CPU is allocated to } J_{i,j} \\ at \text{ time } n_t + t; \end{cases}$$

$$= \begin{cases} 0, & \text{the CPU is idle at time } n_t + t. \end{cases}$$

Note that $n, j$ are integers, and $n \geq 0 \& j \geq 0$ hold for (8). Equation (7) ensures that all transactions are released synchronously at time $0, t_e, 2t_e, \ldots, \text{ etc.}$ If the processor is allocated to job $J_{i,j}$ at time $n_t + t$, then it is the $(j - n(k_i + 1))^{th}$ job of $t_i$ from time $n_t$ (Note that there are $(k_i + 1) t_i$ jobs during the interval $[0, t_e]$). Equations (7) and (8) ensure that the complete SH schedule is constructed periodically by repeating the schedule of the interval $[0, t_e]$ every $t_e$ units.

**Theorem 2.** Given a DS-FP schedule for a validity constrained transaction set T, suppose $t_{idle}$ is an idle time in the schedule and the schedule before $t_{idle}$ is feasible. Let $r_i, k_i \geq 1 (k_i \geq 1)$ be the latest release time of jobs of $t_i (1 \leq i \leq m)$ before $t_{idle}$. If $\forall i (1 \leq i \leq m)$, holds, then the interval $[t_{idle}, 2 \ast t_{idle}]$ can be used as the first hyper period of the DESH-SA schedule without any adjustment. Proof Note that once a job is released under DS-FP, the processor cannot be idle until the job completes. Thus, if all jobs $J_{i,k}$ are released at time $t_{idle}$ i.e., $r_i, k_i = t_{idle}$ then the schedule of the interval $[t_{idle}, 2 \ast t_{idle}]$ is the same as that of the interval $[0, t_{idle}]$. Moreover, if (9) holds,

$$\begin{align*}
(d_{i,k_i} - t_{idle}) + t_{idle} - r_{i,k_i} - 1) & = d_{i,0} - 0 + t_{idle} - r_{i,k_i} - 1 \\
& \leq V_i
\end{align*}$$

That is, two consecutive jobs $J_{i,k_i-1}, J_{i,k_i}$ ($\forall i, 1 \leq i \leq m$) across two neighboring hyper periods satisfy the validity constraint. Thus a feasible DESH-SA schedule can be constructed by having the schedule of the interval $[0, t_{idle}]$ as the first hyper period schedule.

Note that if $t_e$ is set to be $t_{idle}$, then it is not necessary to adjust the schedule of any transactions in the interval $[0, t_{idle}]$ for making the first hyper period of DESH-SA. However, it is not always possible to find such a time tidle for all transactions satisfying (9), in which case the DS-FP schedule in the interval $[0, t_e]$ corresponding to a subset of the transactions needs to be adjusted. Specifically, if transaction $t_i$ ($1 \leq h \leq m$) is the highest priority transaction whose schedule needs to be adjusted due to violation of (9), then the schedule of all lower-priority transactions $t_h (h < i \leq m)$ also
endif
if (r′_{i,j} < d_{i,j} - 1) // Ripple impact.
then d′_{i,j} - 1 ← r′_{i,j}; // Change d_{i,j} - 1.
else d′_{i,j} - 1 ← d_{i,j} - 1; // No change.
endif

j ← j - 1;
else // No adjustment for this job.
if (t_\tau_i ≥ d′_{i,j}) // No more adjustment for τ_i.
then
    t_\tau_i = d′_{i,j};
    break; // Jump out of while loop
else
    // Examine the previous job of τ_i.
    d′_{i,j} - 1 ← d_{i,j} - 1;
    // No change.
    j ← j - 1;
endif
endif
end

if ((j = 0) ∧ (\forall i, k_i))
then report failure;
else t_\tau_i ← 0;
endif
end

RTN adjusted SH in [0, t_e] and ∀ i, k_i;

Examples

Fig 2. Desh-Schedule Construction

Fig 3. DESH-Schedule Adjustment

Performance Evaluation

This section presents important results from our simulation studies of the DESH algorithms. Our goal is to find out whether the DESH algorithms are effective for reducing the DS-FP overhead. The primary performance metrics used in our experimental studies are the CPU workload and the number of transactions supported in the system. In the experiments, we investigate whether DESH-SA and DESH-SC can find a hyper period, and if so, how much excess CPU workloads they may incur compared to DS-FP. We also compare the hyper period length of DESH-SA and DESH-SC to study the space efficiency of the approaches, and demonstrate the percentage of transactions to be adjusted when we calculate the hyper period for DESH-SA. For simplicity, only one version of a real-time data object is maintained. Upon refreshing a real-time data object, the older version is discarded.

We ignore the on-line scheduling overhead in our experiments, and consider it to be O(1) for all algorithms (which is true for DESH algorithms). This is in favor of DS-FP as its scheduling overhead is ignored for the CPU workload in our experiments. We define N_adjust to be the average number of jobs whose release times or deadlines are adjusted in [0, t_e] under DESH-SA.

Conclusion

This paper presents new approaches originate from DS-FP, named Deferrable Scheduling with Hyper period by Schedule Construction and Deferrable Scheduling with Hyper period by Schedule Adjustment, that decreases the on-line scheduling overhead to O(1). Deferrable Scheduling with Hyper period by Schedule Construction searches the deferrable scheduling fixed priority schedule for a hyper period. Where Deferrable Scheduling with Hyper period by Schedule Adjustment adjusts the deferrable scheduling fixed priority schedule in an interval to form a hyper period. Deferrable Scheduling with Hyper period by Schedule Adjustment works better than Deferrable Scheduling with Hyper period by Schedule Construction by performing more number of update transactions in the system and it will also assures the age constraint. It is both space and time efficient. But there are some un answered questions like what is the necessary and sufficient condition is for the Deferrable Scheduling with Hyper period by Schedule Adjustment to produce a hyper period. We are going to address these problems in our future work.

References


