SCHEDULING SHARED DATA ACQUISITION FOR REAL-TIME DECISION MAKING

BY

TAI SHENG CHENG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Professor Tarek Abdelzaher

ABSTRACT

This work investigates scheduling policies for the acquisition of possibly overlapping sets of data items required to make multiple decisions by different deadlines. The work is motivated by military IoT applications in which a large number of sensors must collect intelligence data needed to make multiple decisions. For example, data from several cameras in a contested city might be needed to decide where targets of interest are. This work is based on the assumption that network bandwidth is limited, creating a significant resource bottleneck (perhaps between the sensors and the command center where decisions are made). This might be the case, for example, due to active interference by a determined adversary.

A relieved sub-problem is first discussed with a corresponding optimal algorithm. Then, an improved heuristic algorithm based on the insights from the optimal algorithm of the sub-problem is presented. Finally, the new algorithm is evaluated with multiple scheduling parameters and is compared with previous heuristics, demonstrating an improved performance of our solution.

To my family, my friends and my loved ones, for their love and support.

ACKNOWLEDGMENTS

First of all, I would like to thank Professor Tarek Abdelzaher for his guidance over the past 2 years. He provided me with this interesting topic and gave me a lot of advice during my work. Without his knowledge, I wouldn't have achieved what I have accomplished. I really appreciate this guidance. It has truly been a great journey working with him.

Next, I would like to thank all my colleagues in the lab. I would like to thanks Jinyang Li for always being the life of the party. We sure had a lot of memories for playing basketball. I would like to thank Shengzhong Liu for always teasing Jinyang and bringing all the joy to our group. It would be very different without all the laughter and all the wonderful time we had. I would like to thank Dongxin Liu for flattering us a lot, making me always feel good about myself. I'm sure a lot of us feel the same. I would like to thanks Tianshi Wang for chatting with me whenever I am in the lab. It's always nice to have someone working with you (especially when it's the last weekend before the thesis due date). I would like to thank Huajie Shao for providing us with all the information (including sharing all the rumors about people around us). This truly makes the conversation much more interesting. I would like to thank Dr. Shuochao Yao for providing me with all the refreshing conversation we had. We will sure grab a drink once in a while in Seattle with Shengzhong joining us. I would like to thank Yiran Zhao for being the TA of my first class in UIUC. It was really nice to have a senior whom I could talk to about the group and the ongoing projects. Last but not least, I would like to thanks Yifan Hao for bringing me into the lab. It has truly been a great time hanging out with you, skiing, playing video games, basketball, bowling, etc. It's a pity that we didn't take class together since CS424, or else my graduate life would be much easier. Let's be sure to hang out sometime after we start working.

I would like to show appreciation to my roommates and friends from Taiwan. I really appreciate Eli Chien, Hsien Chih Huang and Hsuan Chi Kuo for living together with me. Thanks to Hsuan Chi Kuo for creating a group of all new incomers to UIUC in the beginning. My grad life would be so much more boring without the people we know. I would like to thank Hsien Chih Huang for cooking all the delicious cuisines. In addition, it is really nice to learn all kinds of trivia and terminologies about our life. I would like to thank Eli Chien for carrying me on playing basketball and surviv.io. We sure aced the game with 53.5% of winning rate and an 8 consecutive winning streak. I would like to thank all my college friends who applied to graduate schools with me. Without you guys, I won't be who I am today. I would especially like to show my gratitude to the ones studying together in UIUC.

We sure had a lot of great memories and playing Mahjong together (although Ben eventually decided to live with his girlfriend instead of me).

I would like to thank Oath for having me as an intern during the summer. It was truly a wonderful journey working there. We had a lot of fun playing PS4 during working hours and chatting about the FIFA World Cup. I would also like to thank Microsoft for giving me the opportunity for a full-time job I'm really desperate to have. With such an offer, I was able to have such a laid back final year of my student life.

Last but not least, I would like to thank my parents for all the support they have given me to study abroad. I would like to thank my brother for always bringing me joy during the days in my grad life. Finally, I would like to thank Claire, for being by my side during my grad life. The joys we had were definitely beyond description. I hope we both have a great career after graduation.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Recent directions in military thinking envision a slew of novel IoT services for the battlefield, collectively termed the *Internet of Battlefield Things*.[1] One key service in that context is real-time intelligence gathering for decision making. For example, consider a humanitarian operation in a city, where medical aid and supplies need to be delivered in the face of an active insurgency that impairs security. A set of sensors (e.g., cameras) are deployed in select locations to search for targets of interest. Since power and thus regular networking infrastructure might be lost, the sensors are equipped with a capability to upload data wirelessly via a military satellite. An adversary might attempt to impair the efficacy of this communication by jamming the spectrum used by the satellite, resulting in a severe reduction in available communication bandwidth. In this scenario, the goal is to collect real-time information from the cameras, using what bandwidth remains available, to make decisions on where the different targets are. A big bottleneck exists between the cameras and the collection point (e.g., a command center) due to satellite jamming.

1.2 PREVIOUS WORK

Past work addressed a simple version of this problem where the sets of data objects (e.g., pictures) needed to make differnt decisions are non-overlapping [1, 2, 3]. For example, in order to decide if the target is at location, x, only pictures from location x are needed. Hence, for different locations, the sets of pictures needed are non-overlapping. Unfortunately, in most cases, different decisions will need overlapping subsets of intelligence data.

When there is only one decision task to be scheduled, an *optimal* scheduling policy, the *Least Volatile item First (LVF)* has been derived [1, 2]. It states that the data item with the longest validity interval should be retrieved first. This ensures that retrieved items have the lowest chance of expiration before a decision is made. Previous work also showed that when multiple decision tasks exist but with non-overlapping sets of data items, the optimal retrieval policy is *Earliest Deadline or Expiration First - Least Volatile First (EDEF-LVF)* [3]. The key idea is to assign a higher priority to the task that has the smallest value of the minimum taken across its data item validity expiration times and its deadline. However, the mentioned results lose optimality if sets of data items needed for different decisions overlap. Intuitively,

[1] http://foxillinois.com/news/local/25-million-grant-to-develop-internet-of-battlefield-things

this is because when data are shared across multiple decisions, some items retrieved for the first decision will expire before the second is made, unless their retrieval is artificially delayed, resulting in the need for a new scheduler.

1.3 PROBLEM STATEMENT

This work assumes that each data item has a freshness interval (or validity interval) before the data item becomes invalid. Whenever a data item becomes stale, the source device can re-transmit the data. That is, once a source is activated, the sampling period of the source is the same as the validity interval of the data item. A decision task considers a set of objects (e.g., evidence) needed for the decision. The challenge lies in retrieving these objects so a deicision is made while they are fresh and before the decision deadline. Thus, a decision task is said to be completed successfully if it satisfies two constraints:

- *Schedulability constraint*: The decision is made before the decision deadline.
- *Validity constraint*: At the time when the decision is made, all data items are within their validity interval.

1.4 THESIS OVERVIEW

An scheduling policy was derived for a sub-problem where some of the constraints are relieved, along with proving its *optimality*. Next, a heuristic algorithm for scheduling decision tasks with multiple shared items based on the insights of the previous optimal algorithm of the sub-problem was proposed.

Previous work [3] proposed simple heuristic algorithms for the problem, where scheduling decision tasks requires shared items. However, neither properties of these algorithms nor evaluations of the trade-off of the algorithms were discussed. The contribution of our work lies in the analysis of such problem where data items are shared. Moreover, based on insights gained from solving a simpler special case, our new heuristic outperforms those proposed in the prior work.

The rest of the work is organized as follows. In Chapter 2, the background and describe the task model is described, followed by an example of such problem in Chapter 3. Chapter 4 illustrates the problem of scheduling decision tasks with shared items. It demonstrates the *optimality* properties of a scheduling policy that solves the problem when there is only one shared item. Finally, a heuristic scheduling policy is proposed for the general problem of multiple decision tasks with shared data items. Chapter 5 evaluates the new algorithms. The related work is discussed in Chapter 6 and the work is concluded in Chapter 7.

CHAPTER 2: BACKGROUND AND TASK MODEL

The overview of our task model is given in this chapter. First of all, a decision task m is a task that contains a set of data items (or items) $O_1^m, ..., O_{k^m}^m$. In our model, decision task m is said to be completed (or decision is made) when all the items within the decision task m are either retrieved or still valid. The validity of a task will be described later in this section. Moreover, retrieving items of decision tasks is non-preemptible. In other words, once an item is scheduled, the decision task must completed the retrieval of the current item before the task can be preempted and another decision task can be scheduled to retrieve items. For every item $O_i^m, \forall i \in \{1, ..., k^m\}$, there exists a validity interval, I_i^m, which is defined as the *freshness* of the item. When the item is scheduled to be activated, the sensor samples the environmental measurements at a period of I_i^m, which is the same as the validity interval of item O_i^m. Note that an item will only be re-retrieved when the decision task isn't finished and the item has passed the validity interval. Upon retrieving the measurement, it takes the sensor C_i^m cost to transfer the data to the control system. Similarly, the set of items that is shared by task m and other tasks is defined as $S_1^m, ..., S_{l^m}^m$. The validity interval and the retrieval time of item S_i^m is $I_{S_i^m}^m$. After the system verifies that all the items for decision task m are still valid, data retrieving of decision task m is concluded. Let t^m denote the arriving time of the decision task. For each of the item O_i^m, it is scheduled to deliver the result to the system at time t_i^m. F^m and D^m are denoted as the finish time and the deadline of decision task m, respectively. Therefore, the absolute deadline for decision task m can be computed as:

$$AD_m = t^m + D^m \tag{2.1}$$

Similarly, the absolute validity expiration time for item O_i^m is:

$$Exp_{O_i^m} = t_i^m + I_i^m \tag{2.2}$$

Finally, a schedule sequence is said to be *feasible* if the following conditions hold:

i) A decision task has to be completed before the decision deadline, *i.e., the schedulability constraint.*

$$F^m \leq AD_m \tag{2.3}$$

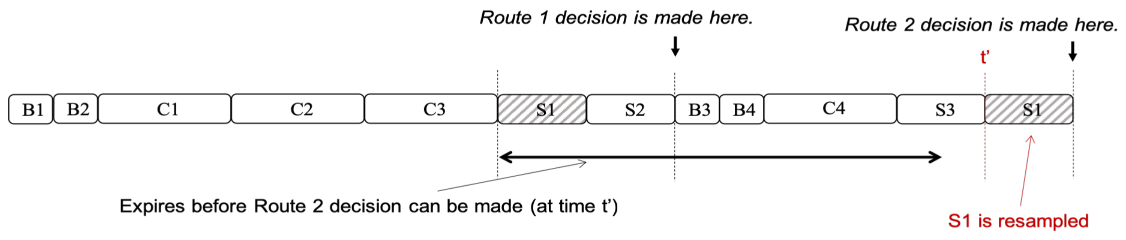
ii) All items within the decision deadline has to be valid when the decision is made, *i.e., the validity constraint.*

$$Exp_{O_i^m} \leq F^m \tag{2.4}$$

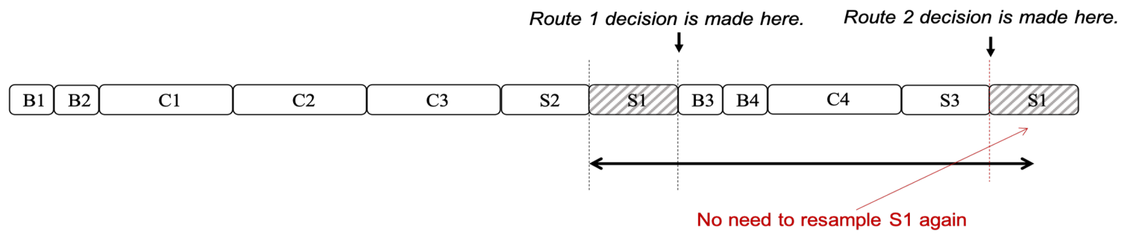
Table 2.1: Example for Items Required to Decide which Route to Take

Notation	Description
K	Number of decision tasks
k^m	Number of data items in decision task m
l^m	Number of shared items in decision task m
D^m	Relative deadline of decision task m
t^m	Arrival time of decision task m
F^m	Finish time of the decision task m
O_i^m	Data item i of decision task m
C_i^m	Retrieval time of data item O_i^m
I_i^m	Validity window of data item O_i^m
t_i^m	Start time of retrieving data item O_i^m
S_i^m	Shared item i between decision task m and all other tasks

CHAPTER 3: AN ILLUSTRATIVE EXAMPLE


Route 1 decision is made here.
Route 2 decision is made here.
t'
B1
B2
C1
C2
C3
S1
S2
B3
B4
C4
S3
S1
Expires before Route 2 decision can be made (at time t')
S1 is resampled

(a) The case where S1 has to be re-retrieved


Route 1 decision is made here.
Route 2 decision is made here.
B1
B2
C1
C2
C3
S2
S1
B3
B4
C4
S3
S1
No need to resample S1 again

(b) All items are sampled exactly once

Figure 3.1: The example of reusing item S1

This chapater presents an illustrative example to further explain why this problem is non-trivial and demonstrate that it cannot be solved by the previously proposed solution. Continuing with our example of collecting intelligence for decision making in a city, imagine a case where civilians should be evacuated. A route suggestion system inspects cameras and other sensors along different potential evacuation routes in order to make a decision on the safest one. However, evacuation route options of different civilians overlap in some segments. Table 3.1 shows the sensors and cameras required for the evacuation system to determine whether *Route 1* or *Route 2* is a better evacuation route, along with the retrieval time and the validity interval for content of each sensor. Note in particular, that item S1 is shared across the two routes.

Let us first use algorithm EDEF-LVF, which is proven to be optimal when scheduling multiple decision tasks without shared items [3]. As shown in Figure 3.1(a), the shared item S1 will become stale when items for the decision task *Route 2* are fetched. Therefore, item S1 has to be re-retrieved. If the scheduling order is changed as shown in Figure 3.1(b), S1 will still be valid when the items in *Route 2* have been retrieved. The problem can be even harder when multiple decision tasks shared multiple data items.

Table 3.1: Example for Items Required to Decide which Route to Take

Route 1 Item	Retrieval Time(sec)	Duration(sec)
Bridge 1 status sensor (B1)	1	600
Bridge 2 status sensor (B2)	1	600
Camera 1 (C1)	3	30
Camera 2 (C2)	3	30
Camera 3 (C3)	3	30
Sensor 1 (S1)	2	10
Sensor 2 (S2)	2	10
Route 2 Item	Retrieval Time(sec)	Duration(sec)
Bridge 3 status sensor (B3)	1	600
Bridge 4 status sensor (B4)	1	600
Camera 4 (C4)	3	30
Sensor 3 (S3)	2	10
Sensor 1 (S1)	2	10

CHAPTER 4: SCHEDULE RETRIEVING DATA SHARED BETWEEN DIFFERENT DECISION TASKS

This chapter discusses the problem of scheduling decision tasks with shared items to pruduce a *feasible* scheduling sequence.

4.1 A RELIEVED SUB-PROBLEM

To solve the problem, this work begins by solving a sub-problem where all decision task shared the same item and arrive at the same time. First, the notation of this problem is provided. Decision task m has arrives at t^m and has a relative deadline at D^m. Note that in the sub-problem, all the arriving time should be the same, denoted as t_{arrive}. That is, the absolute deadline for task m is $t_{arrive} + D^m$. There are k^m items within decision task m, *i.e.*, $O_1^m, ..., O_{k^m}^m$. Each of the items has a retrieval time C_i^m and a validity interval I_i^m. Data item O_i^m is said to be retrieved at time t_i^m. This work assumes that there is only one shared item between decision task m and other tasks, which is denoted as S_1^m; furthermore, decision task m is assumed to finish retrieving at time F^m. This sub-problem only discusses the case where all $S_1^m, \forall m \in [1, K]$ are identical.

4.2 OPTIMAL ONLINE SCHEDULING ALGORITHM FOR THE SUB-PROBLEM

To minimize the cost, whether the shared item could be reused after retrieved in the first decision task to avoid resampling in the second decision task should be investigated. The example in Chapter 3 provided an insight that in order to reuse a task, the scheduler should shcedule the item as close to the deadline as possible. That is, it will have the largest remaining valid window for the next decision task. This work begins by discussing some correlated properties and then present and evaluate the scheduling algorithm. Although this part is covered in detail in previous works, only the discussions of the concepts are presented, while the proofs to calculate the priority of decision tasks are neglected. To begin with, the following describes the process to prioritize multiple decision tasks. The optimal algorithm EDEF-LVF states that *the highest priority should be assigned to the task with the smallest value of the minimum of its item validity expiration times and its decision deadline.* The earliest expiration time for decision task m id defined as:

$$minExp(m, t) \tag{4.1}$$

where t is the current time. That is, the highest priority is given to the task with the smallest minimum of validity expiration times and deadline, which can be represented as:

$$\min_{m=1,\dots,K}(minExp(m,t), t^m + D^m) \tag{4.2}$$

The above concept is summarized in Theorem 4.1.

Theorem 4.1 *If a feasible order exists for a decision task set, the scheduling scheme that assigns the highest priority to a task with a smaller minimum of item validity expiration times and deadline can always schedule the task set.*

Proof. The theorem is proved as Theorem 2 in our previous work [3].

Next, the algorithm is presented following by the discussion of the optimality of the algorithm.

Algorithm 4.1

```
t: the current time
RQ: the ready queue of decision tasks
S: the shared item
I_S: the validity interval of S
1: $V_{shared} \leftarrow 0$
2: if the set of decision task arrives then
3:     Decision tasks are added to ready queue $RQ$ in ascending order with respect to their deadline $D^m$
4:     while $RQ$ is not empty do
5:         Task $m$: the first task in $RQ$
6:         Dequeue Task $m$ from $RQ$
7:         $V_{shared} \leftarrow Retrieve(Task\, m, V_{shared})$
```

Procedure $Retrieve(Task\, m, V_{shared})$

```
1: $new\_V_{shared} \leftarrow V_{shared}$
2: $F_{predicted} \leftarrow t + \sum C_i^m - C_S$
3: if $F_{predicted} \leq V_{shared}$ then
4:     Remove $S$ from $O^m$
5: $seq(m) \leftarrow LVF\_SEQ(Task\, m)$
6: Retrieve data item in $seq(m)$
7: if $S$ is retrieved then
8:     $new\_V_{shared} \leftarrow t + I_S$
9: return $new\_V_{shared}$
```

Algorithm 4.1 implements the *optimal online* scheduling policy for the relieved sub-problem. The variable V_{shared} is used to store the absolute validity expiration time of item S. The decision task are first sorted in ascending order with respect to their deadline D^m. That is, the task with the smallest deadline will be executeed first. Note that the task with the smallest deadline D is the task with the smallest absolute deadline since

```
Procedure $LVF\_SEQ(Task\, m)$
1: $seq \leftarrow$ items $O_1^m, ..., O_{k^m}^m$ sorted in decending order with respect to $I_i^m$
2: if $S \in seq$ then
3:     $S$ is the $k^{th}$ item in $seq$
4:     $F_{predicted} \leftarrow t + \sum C_i^m$
5:     while $F_{predicted} \leq t_{k+1}^m I_{k+1}^m - I_S$ do
6:         switch $S$ and $O_{k+1}^m$
7:         $k \leftarrow k + 1$
8: return $seq$
```

all tasks have the same arrival time. Once the task is dequeued from RQ, the procedure $Retrieve(Task\, m, V_{shared})$ is called to retrieve the items for task m.

The procedure $Retrieve(Task\ m, V_{shared})$ retrieves the item of task m. The variable new_V_{shared} is used as the updated of the absolute validity expiration time for shared item S. The total computation time required to retrieved all items except for S for task m is computed first. If the predicted finish time to retrieve all the items $t + F_{predicted}$ is less than V_{shared}, the shared item is removed from the task set (Line 2-4). The detail will be discussed in Lemma 4.1. Next, the scheduling sequence with procedure $LVF\_SEQ(Task\, m)$ is computed. The procedure will return the sequence of algorithm LVF [1, 2]. Once the sequence is computed, the scheduler will start retrieving the items according to the sequence. (Line 5-6) On the other hand, if the shared item appears in the retrieving sequence and is retrieved, new_V_{shared} is calculated and updated.

The procedure $LVF\_SEQ(Task\ m)$ computes and return the sequence of un-retrieved items for task m. The sequence is produced by the algorithm LVF [1, 2] and the knowledge of the shared task. Therefore, the seq is ordered and sorted by the valid window. Consider the two following scenarios:

- If the sequence seq does not contain the shared item S, the algorithm will return seq.
- If seq contains S, the item has to be scheduled as close to the deadline D^m as possible. The details will later be presented in Lemma 4.2. This process is done by iteratively checking if switching with the next item in the schedule violates the validity constraints (Equation 2.4). Once the shared item cannot be switched with the next item, the schedule is complete (Line 2-7).

Lemma 4.1 *If the predicted finish time for task m is earlier than the absolute validity expiration time of the shared item S, task m can be made without retrieving S.*

Proof. A decision task m is predicted to finish when all items are retrieved. That is, the predicted finish time is the current time plus the retrieved time of all items. The expected retrieval without retrieving the shared items, *i.e.* $F_{predicted} = t + \sum C_i^m - C_S$, is first computed in order to check if the shared item will still be valid until the decision is made. If the shared item is still valid at $F_{predicted}$, it can safely be assumed that the decision can be made at time $F_{predicted}$.

Lemma 4.2 *If switching the schedule of item O_i with a subsequent item, in terms of scheduling sequence, O_j and does not violate the validity constraint, the absolute validity expiration time for object O_i becomes larger.*

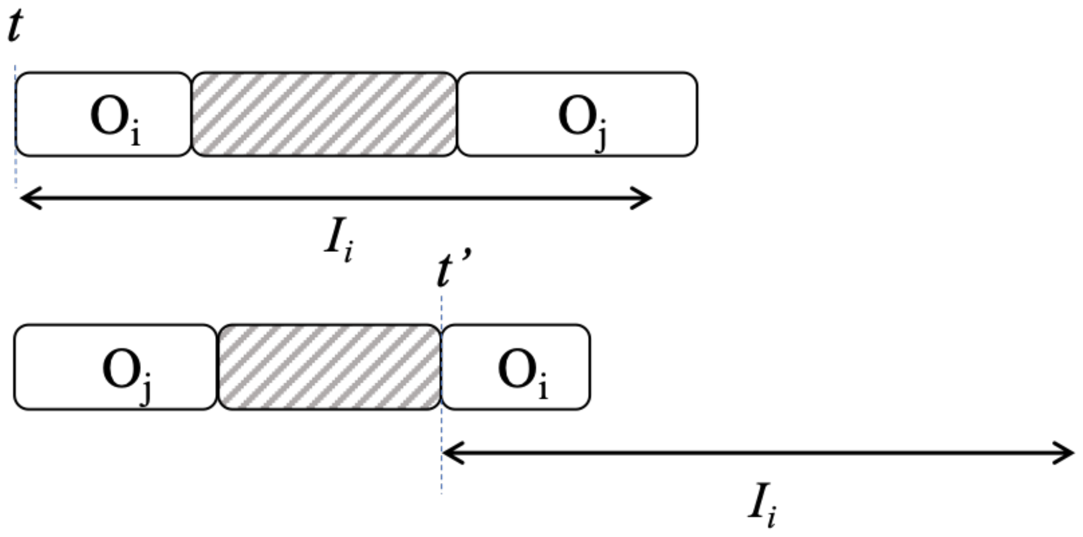


Figure 4.1: The example of switching O_i and O_j

Proof. Let's consider this example from Figure 4.1. The two cases result in different absolute validity expiration time for object O_i. In the first case, where O_i is scheduled before O_j, the absolute validity expiration time is the schedule time t plus the validity window I_i. On the other hand, if the two items are switched, the absolute validity expiration time of O_i now becomes $t' + I_i$, which is larger than $t + I_i$. Thus, the lemma is proved.

Theorem 4.2 *If a feasible schedule sequence exists for a decision task set, iteratively switching the shared item S before violating the validity constraint can produce the largest absolute validity expiration time for S.*

Proof. From Lemma 4.2, it is clear that switching with a subsequent item can produce a larger absolute validity expiration time. The absolute validity expiration time for S gradually increases as interatively switching S with the direct subsequent item. The process stops when switching with the direct subsequent item, denoted as O_a, violates the validity constraint.

The constraint is violated because switching results in an earlier absolute validity expiration time for O_a. It can also be described as $(t_a - C_S) + I_a < D$, where the new schedule time $(t_a - C_S)$ plus the validity window I_a is earlier than the deadline D. If switching with the direct subsequent item will result in the violation above, switching with any subsequent will also result in the violation above. This is because the validity window is less than or equal to item O_a, which is denoted as $I_a - \epsilon$. Therefore, $(t_a - C_S) + (I_a - \epsilon)$ will still be less than the deadline D. It is proved that switching until S reaches O_a will increase the absolute validity expiration time of S. Also, no more subsequent items can further be switched. Therefore, the theorem is proved.

Theorem 4.3 *Algorithm 4.1 is optimal in terms of solving the sub-problem described in Section 4.1.*

Proof. Before proving the theorem, consider the previous assumption that items are non-preemptive once it is scheduled to retrieve. In order to prove the optimality, the optimality task-wise and item-wise is discussed in the following. When a scheduler is invoked, the task that has the smallest minimum of validity expiration time is chosen. According to Theorem 4.1, this method is optimal when there are multiple decision tasks. That is, if a *feasible* sequence exists, the method of choosing the smallest minimum validity expiration time can also produce the *feasible* sequence. On the other hand, in Line 2-7 in Procedure $LVF\_SEQ(Task\, m)$, if the shared item S exists in the scheduling sequence seq, S is iteratively switched towards the deadline of the decision task. From Theorem 4.2, this switching process will produce the largest absolute validity expiration time for S. In other words, to prove the procedure is optimal is identical to prove that it produces a sequence where the total cost of computation is minimized. This is equivalent to finding the sequence where the times of retrieving S is minimized. Suppose that an optimal sequence seq_{opt} have a smaller absolute validity expiration time when S is first retrieved comparing to our produced sequence seq_{comp}. Whenever S is retrieved in seq_{opt}, S is also retrieved in seq_{comp}. This construction of seq_{comp} can be done because the absolute validity expiration time of S is larger in seq_{comp} than in seq_{opt}. Every decision in seq_{comp} can be made at the same time in seq_{opt}. That is, this procedure can result in an optimal when scheduling items within decision tasks. The optimality of both task-wise and item-wise of Algorithm 4.1 is shown. As a result, this concludes that Algorithm 4.1 is optimal for the problem described in Section 4.1.

4.3 A HEURISTIC ALGORITHM MULTIPLE SHARED ITEMS SCHEDULING

The relieved sub-probelm is discussed and a corresponding *optimal* algorithm is presented. However, in the real world, different tasks may share different sets of items and tasks may arrive in different time. The decision task scheduling problem becomes very hard when preemption happens at item level, where tasks may be preempted when a task with higher priority arrives. Therefore, a heuristic algorithm that takes advantages of the properties from Algorithm 4.1 is presented. This work begins by presenting the algorithm and followed by the description and the design philosophy of the algorithm.

Algorithm 4.2 RVI-LVF

```
$t$: the current time
$RQ$: the ready queue of decision tasks
$S$: the shared item
$I_S$: the validity interval of $S$
 1: if a new decision task arrives or a task finishes then
 2:     if a new task $n$ is arriving then
 3:         $t^n \leftarrow t$
 4:         $AD_n \leftarrow t^n + D^n$
 5:         if no task is running then
 6:             $Retrieve(Task\, n)$
 7:         else
 8:             task $m$: the task currently retrieving items
 9:             if $AD_m \leq AD_n$ then
10:                 Enqueue $n$ to $RQ$
11:             else
12:                 Enqueue $m$ to $RQ$
13:                 $V_{shared} \leftarrow Retrieve(Task\, n, V_{shared})$
14:     else
15:         $Task\, m \leftarrow argmin_{Task\, m \in RQ}(AD_m)$
16:         Dequeue $Task\, m$ from $RQ$
17:         $Retrieve(Task\, m)$
```

Procedure $Retrieve(Task\, m)$

```
1: $F_{predicted} \leftarrow t + \sum C_i^m$
2: $S^m$ : the $l^m$ items shared by m and other decision tasks sorted in decending order with respect
   to $I^m_{S_i^m}$
3: for $i \leftarrow 1$ to $l^m$ do
4:     if $S_i^m$ is still valid at $F_{predicted}$ then
5:         Remove $S_I^m$ from $O^m$ and $S^m$
6:     else if $S_i^m$ is not shared with any task in $RQ$ then
7:         Remove $S_i^m$ from $S^m$
8: $seq(m) \leftarrow LVF\_SEQ(Task\, m)$
9: Retrieve data item in $seq(m)$ and remove from $O_i^m$
```

This algorithm is called *Reuse Valid Item - Lease Volatile First(RVI-LVF).* RVI-LVF implements the heuristic *online* scheduling policy for multiple shared data item acquisition. The scheduler is invoked whenever a previously scheduled decision task finishes executing or

Procedure $LVF\_SEQ(Task\, m)$

```
1: $seq \leftarrow$ items $O_1^m, ..., O_{k^m}^m$ sorted in decending order with respect to $I_i^m$
2: for $i \leftarrow 1$ to $l^m$ do
3:     $S_i^m$ is the $k^{th}$ item in $seq$
4:     while Switch $S_i^m$ and $O_k^m$ does not violate validity constraint do
5:         Switch $S_i^m$ and $O_k^m$
6:         $k \leftarrow k + 1$
7: return $seq$
```

when a new decision task arrives. The variable AD_n is used to denote the *smallest minimum of item validity expiration times and deadline* of task n. It is calculated by Equation 4.2. When the new decision task n arrives, the three following scenarios are considered:

- If there is no current executing task, AD_n will store the absolute deadline of task n. The procedure $Retrieve(Task\, n, V_{shared})$ will then be executed (Line 5-6).

- If the currently executing task m has a smaller *minimum of item validity expiration times and deadline*, the scheduler will mark the absolute deadline of task n and add task n to the ready queue RQ. Task m will continue to be executed (Line 8-10).

- If the new arriving task n has a smaller *minimum of item validity expiration times and deadline*, the scheduler will preempt task m and add task m to the ready queue RQ. Task n will start retrieving data items (Line 11-13).

If the scheduler is invoked by a finished task, the scheduler will select the task that has the earliest absolute deadline from the ready queue RQ (Line 14-17).

The procedure $Retrieve(Taskm)$ is called to retrieve the items in order to execute decision task m. The expected finish time $F_{predicted}$ is first calculated to determine whether each of the shared task S_i^m should be re-retrieved. Note that is is assumed that there are l^m items within O^m that is shared with other task sets. It checks whether each of the shared item S_i^m is still valid at $F_{predicted}$; if so, it will be removed from item sets O^m and S^m. On the other hand, the scheduler check if the item S_i^m is shared with any task in RQ. If not, this means that the task will not be reused for further scheduling and thus should be removed from the shared task set S_i^m (Line 3-7). Once the item sequence from procedure $LVF\_SEQ(Task\, m)$ is acquired, the scheduler retrieves the item according to the sequence and remove the item from the item list O_i^m.

The procedure $LVF\_SEQ(Task\, m)$ is where the item sequence for task m is arranged. In the first step, the items $\{O_1^m, ..., O_{k^m}^m\}$ are sorted in descending order with respect to the validity window I_I^m to construct the *least volatile first* sequence. Next, the scheduler begins by iteratively switching each shared task with the direct subsequent task. According to

Lemma 4.2, this can produce a larger absolute validity expiration time for the shared item (Line 2-6). Finally, the procedure is done, and the order sequence is returned.

Theorem 4.4 *When solving the sub-problem described in secion 4.1, RVI-LVF produces the optimal scheduling sequence.*

Proof. Proving this theorem is equivalent to proving RVI-LVF yields to Algorithm 4.1 when solving the sub-problem described in secion 4.1. To prove they produce the same scheduling sequence is the same to prove that they produce the same order in terms of scheduling data items and decision tasks.

- When scheduling data items, both Algorithm 4.1 and RVI-LVF adopts the idea of LVF, which is placing the item according to their validity window. The main difference is that RVI-LVF is designed to deal with multiple shared items. In other words, if there is only one shared item, *e.g.,* S_1 between all decision task, it will also iteratively switch S_1 towards the deadline to provide a larger absolute validity window. Therefore, it will produce the same item sequence as Algorithm 4.1.
- When a task arrives, both algorithms assign the highest priority to the task with the smallest minimum item validity expiration time and deadline. According to the statement above, both algorithms produce the same set of item sequence; therefore, it is safe to infer that the finish time of the first decision task is the same. By induction hypothesis, it is assumed that Algorithm 4.1 and RVI-LVF produces the same finish time for the k^{th} decision task. The ready queue for both algorithms is identical as they have the exact same absolute time and has finished the same set of decision task. They both select the task with the highest priority, where these two tasks should be identical. As shown previously, they will have the identical scheduling sequence and therefore finish at the same time. Therefore, by showing that all decision task finishes at the same time, the proof is concluded that RVI-LVF produces the same result as Algorithm 4.1 when solving the sub-problem.

According to Theorem 4.3, since Algorithm 4.1 is *optimal*, it is a proved fact that RVI-LVF is also *optimal* when solving the sub-problem described in secion 4.1.

CHAPTER 5: PERFORMANCE EVALUATION

In this chapter, the algorithm is evaluated by comparing the schedulability of the task sets to the previously proposed algorithms, i.e. EDEF-LVF and SPECULATION. Since is is proved that RVI-LVF yileds to Algorithm 4.1 when solving the relieved sub-problem in Theorem 4.4, only the experiment of Algorithm RVI-LVF is conducted. The chapter begins by introducing the algorithms that are compare with:

- EDEF-LVF: This is the optimal algorithm for scheduling multiple data item acquisition tasks without shared item [3]. It assigns the highest priority to the task that has the smallest minimum validity expiration time and deadline. The data item are retrieved with respect to their validity window.
- Speculation: This is the heuristic algorithm from EDEF-LVF. Whenever an item is retrieved, it checks if it was previously retrieved and its expiration time is not earlier than the deadline for the current task. If so, the item does not have to be retrieved.

In the experiment, the results from multiple dimensions is compared, including the arrival rate of decision tasks, the toal utilization of a given task set (hereinafter referred to as task utilization), and the percentage of shared item among all tasks. The task utilization is defined as:

$$\sum_m \frac{\sum_{1\leq m\leq k^m} C_i^m}{D^m}, \tag{5.1}$$

and the load ratio of shared items is defined as:

$$\frac{\sum\limits_m S^m}{\sum\limits_m O^m}. \tag{5.2}$$

5.1 DIFFERENT TASK UTILIZATION COMPARISON

First of all, RVI-LVF is compared with EDEF-LVF and speculation based on different task utilization. To simulate and evaluate the algorithms, 1,000 synthetic task sets are generated for every set of task utilization, i.e. (0,0.1], (0.1,0.2],...,(0.9,1], which is shown as percentage in the x-axis of Figure 5.1. Each task set consists of 10 to 12 tasks and each task consists of 5 to 10 data items to be acquired. The arrival time of each task is randomly drawn between [0,400] and [0,800] in order to show the affect of different arrival rate. The relative deadline of each task is randomly assigned between 50 and 100. The cost to retrieve an item is randomly assigned from 1 to 20. Each data item can be categorized into one of the three following types:

- Small validity window item: the validity winodw of such items is the deadline of the task, with some minor deviation incurred
- Medium validity window items: the validity window of such items is randomly generarted from 3 to 5 times of the deadline of the task
- Large validity window items: the validity window of such items is randomly generated from 10 to 50 times of the deadline of the task

Last but not least, in this experiment, there are 80% of the items are shared with at least one other task.

Figure 5.1 shows the percentage of schedulable tasks with respect to different scheduling algorithms. A task is said to be *schedulable* if a *feasible* scheduling sequence could be found and if the task meets its deadline. That is, algorithm RVI-LVF outperforms both SPECULATION and EDEF-LVF. First of all, the algorithm EDEF-LVF does not take into account of the fact that items could be shared among different tasks. That is, it schedules the item to be retrieved regardless of the validity of the item. As a result, there will be some redundant item retrieved, which leads to a lower schedulability. As for the heuristic algorithm SPECULATION, it inspects the validity of the item when scheduling to retrieve the item. However, RVI-LVF has a better schedulability due to the fact that it schedules the shared items later in the scheduling sequence to provide a larger validity window for subsequent tasks. In addition, SPECULATION checks the extension of the validity comparing to the deadline of the task. However, using the deadline as checkpoint overestimates the required extension of the validity window. In RVI-LVF, a *predicted finsih time* for the task is computed. It does not retrieve an item if it has been retrieved and its validity extends until the *predicted finish time*.

On the other hand, Figure 5.1 also discussed about the impact of different arrival rate. To begin with, the execution window of a task is defined as the arrival time to the deadline, and the remaining execution window as the execution window minus the time interval where higher priority tasks are executing. In Figure 5.1(a), the arrival time of the tasks are more dense (higher arrival rate). That is, there will be a lot of everlaps in the execution time of each tasks. As a result, the scheduability of tasks decreases more drastically comparing to Figure 5.1(b), where tasks have lower arrival rate. Furthermore, RVI-LVF has a better improvement comparing to SPECULATION in lower arrival rate. When the tasks have high arrival rate, there will be too much overlaps in the execution window of each task. Even if RVI-LVF tends to be more aggresive on reusing shared items, the remaining execution window for subsequent tasks is still too low to finish a task. Therefore, a lower arrival rate will lead to a larger ramining execution window, therefore giving the advantage of sharing items.

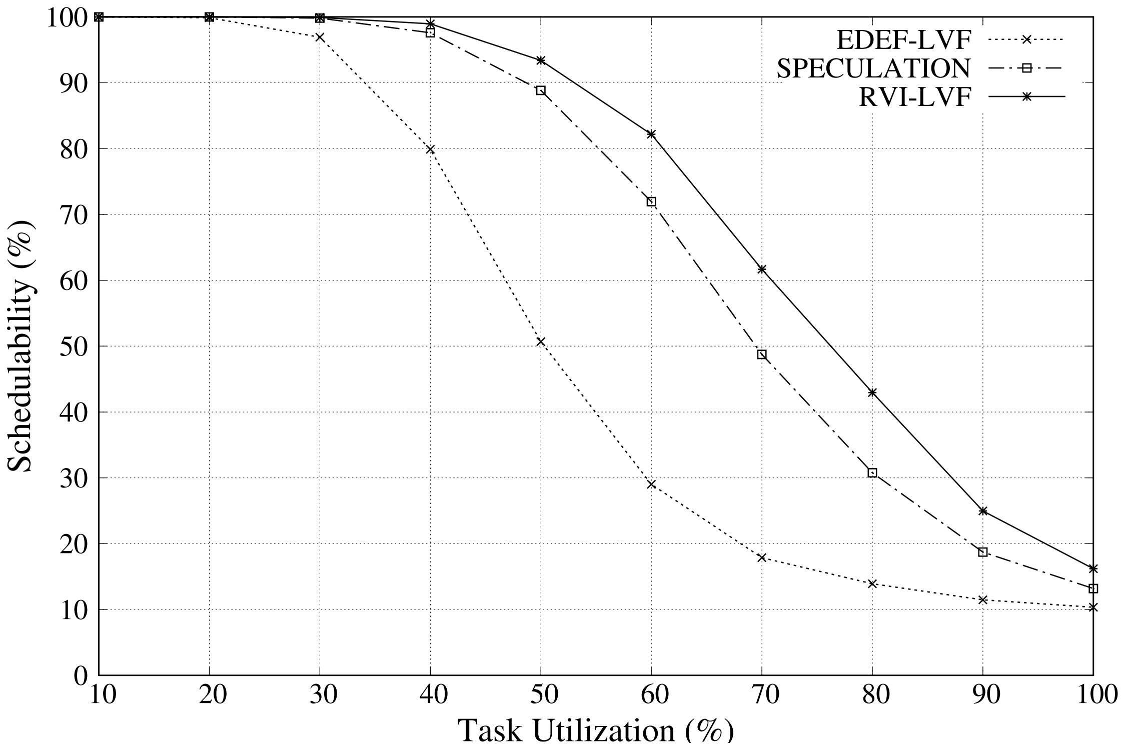
Furthermore, the CPU utilization of different scheduling algorithms with respect to task utilization is also compared. In Figure 5.2, it is clear that RVI-LVF has lower CPU utilization comparing to both EDEF-LVF and SPECULATION. First of all, EDEF-LVF does not take into account of the shared items, therefore wastes a lot of CPU cost on retrieving items that are still valid. SPECULATION tends to make passive prediction on the validity of shared items (uses deadline of the task instead of the predicted finsih time). On the other hand, in the experiment where tasks have higher arrival rate, as the task utilization increases, the CPU utilization of SPECULATION and EVI-LVF gradually converge to the CPU utilization of EDEF-LVF. This is because the remaining execution window of tasks are smaller and therefore the system could easily be overloaded as the task utilization increases. In contrast, when the tasks are arriving in a lower arrival rate, the remaining execution window is larger. That is, the saving of costs on RVI-LVF and SPECULATION is more evident.

5.2 DIFFERENT SHARED ITEM LOAD COMPARISON

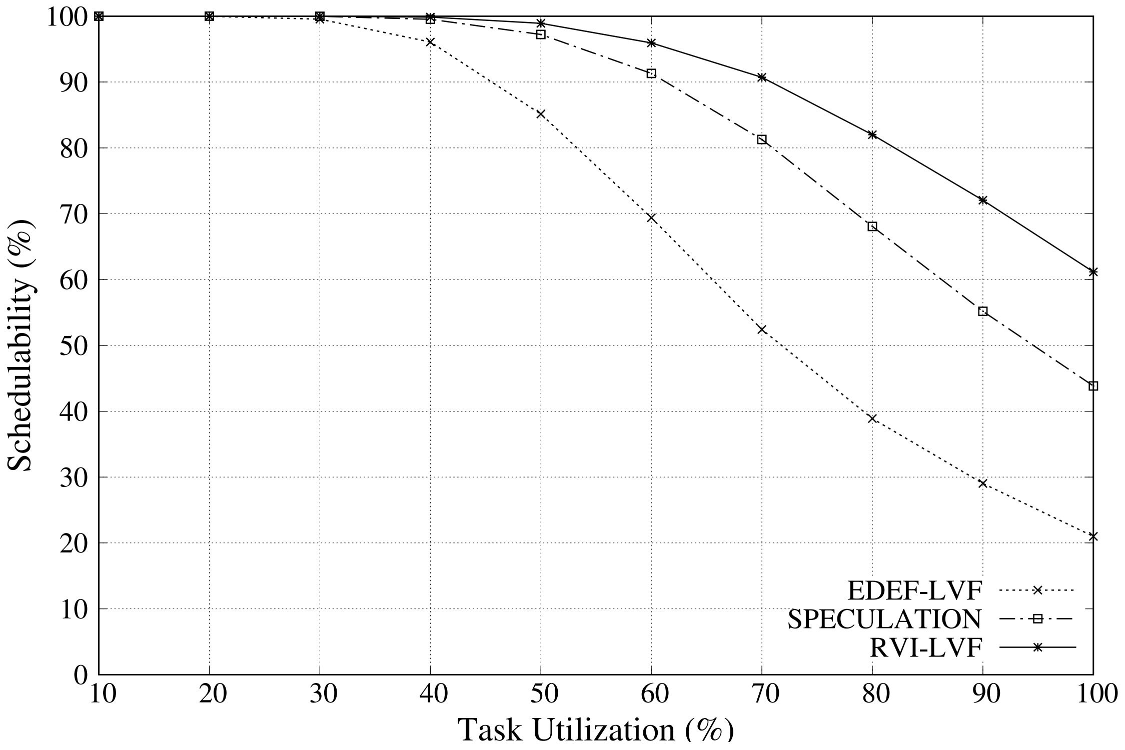
In this section, the algorithms performance with respect to different shared item percentage is compared. Recall that the load of shared item is calculated by Equation (5.2). The setup to generate such task sets is identical to the procedure described in Section 5.1, only that the task sets are grouped by the load of shared items, i.e. [0],(0, 0.1],(0.1,0.2]...,(0.9,1], which is shown in the x-axis of Figure 5.3. The synthetic task sets have a 80% of task utilization.

As shown in Figure 5.3, all three algorithms have the exact same percentage of schedulable tasks when the load of shared items is 0%. Note that EDEF-LVF is the optimal scheduling algorithm for scheduling multiple decision tasks without shared items[3]. SPECULATION and RVI-LVF are 2 heuristic algorithms based on EDEF-LVF. When there are no shared item, the speculation for not retrieving items are no longer in use. Therefore, when there is no shared item, all three algorithms have the same schedulability. As the load of shared items increases, the ratio of schedulable tasks increases in both RVI-LVF and SPECULATION. However, since EDEF-LVF is ignorant to shared items and the total task utilization remains the same, the schedulability remains the same for EDEF-LVF in both cases.

On the other hand, Figure 5.3 also displays the percentage difference of schedulable tasks as shared item load increases in different arrival rate. When the arrival rate is lower, the percentage difference remains approximately the same when the load of shared items is in the range of [50%,100%]. This is due to the fact that when the arrival rate is lower, the execution window (discribed in Section 5.1) is larger. As previously discussed, RVI-LVF speculates the validity of items more aggressive than SPECULATION. When the arrival rate is lower, the remaining execution window is larger. When the load of shared item is high enough, the schedulable tasks will start to remain the same. On the contrary, when the arrival rate is higher, the remaining execution window for each task is smaller. That is, as the load of shared items increases, the shcedulable tasks will increases faster in RVI-LVF than SPECULATION.

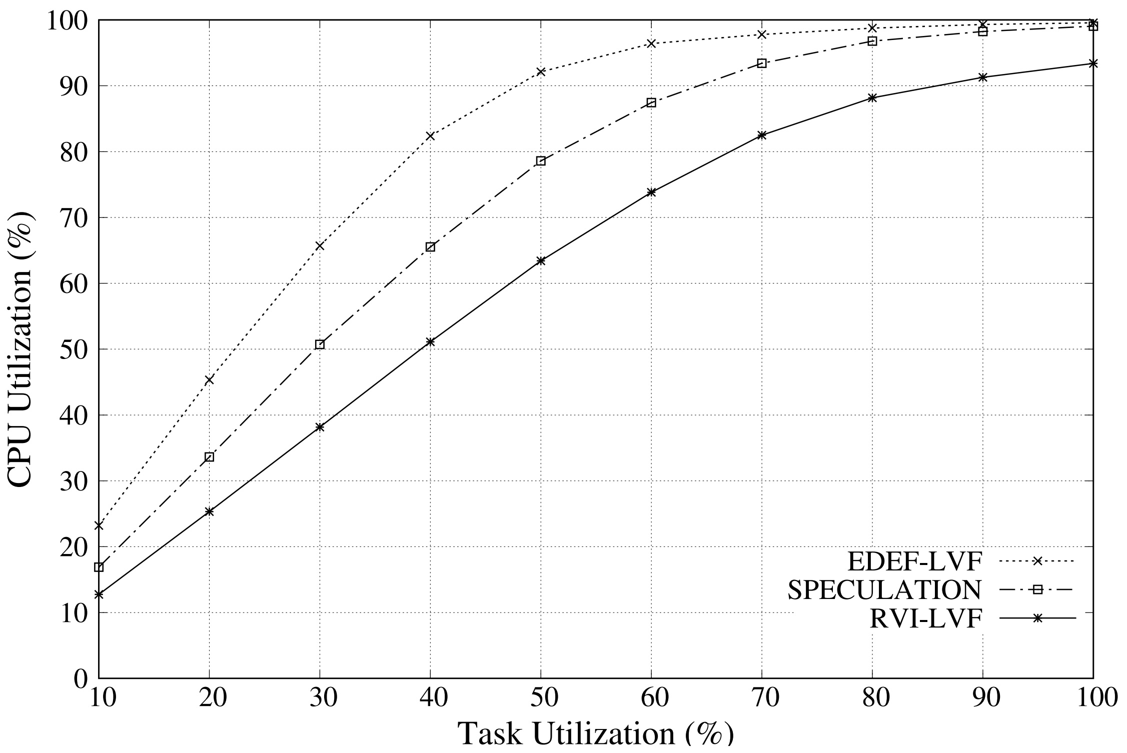


(a) The probability of schedulable tasks with high arrival rate

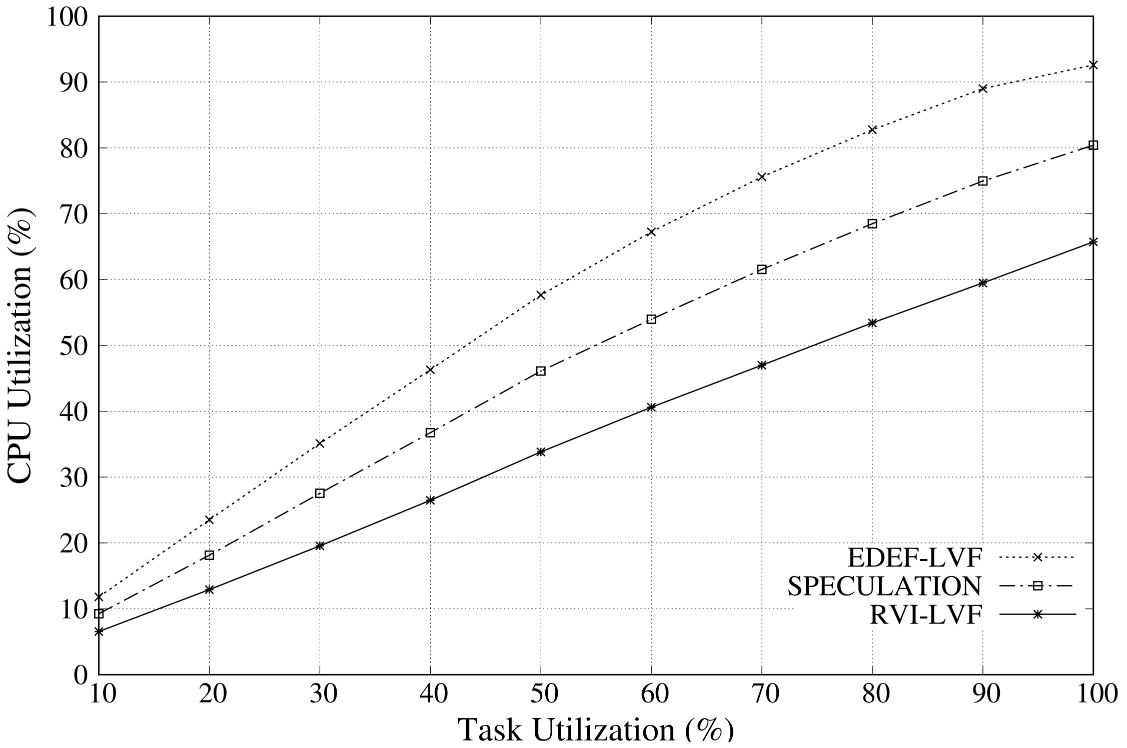


(b) The probability of schedulable tasks with low arrival rate

Figure 5.1: The probability of schedulable tasks with respect to different task utilization and arrival rate

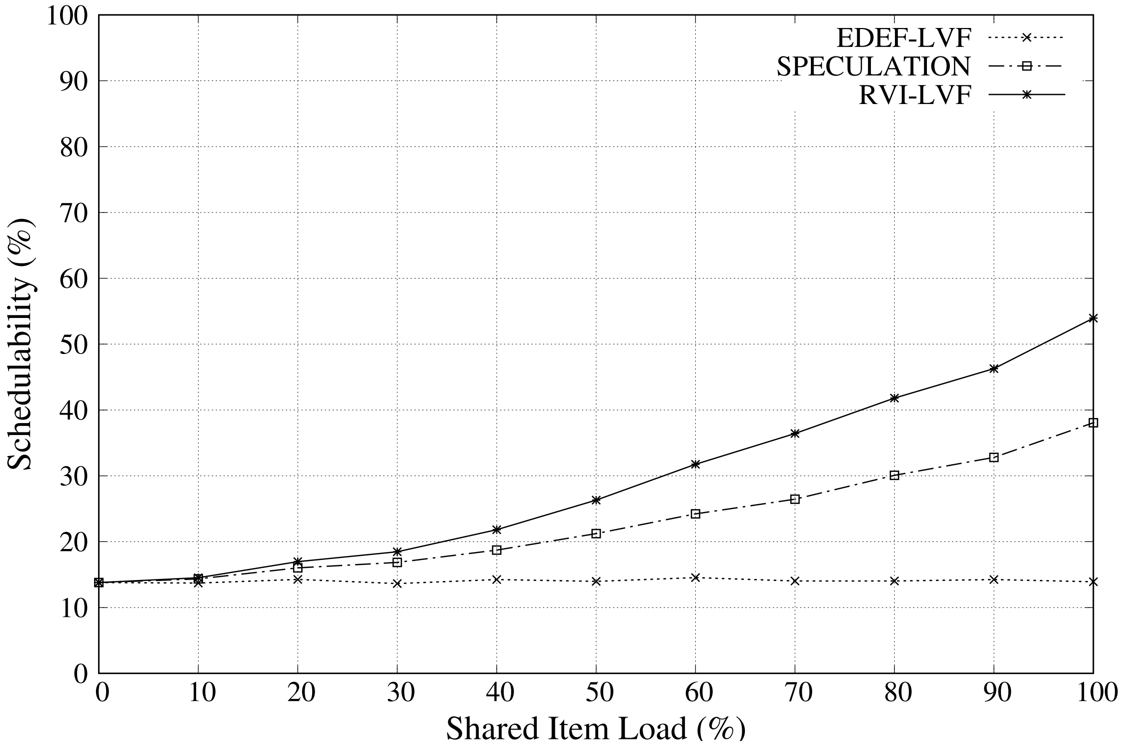


(a) The CPU Utilization of different algorithms with high arrival rate

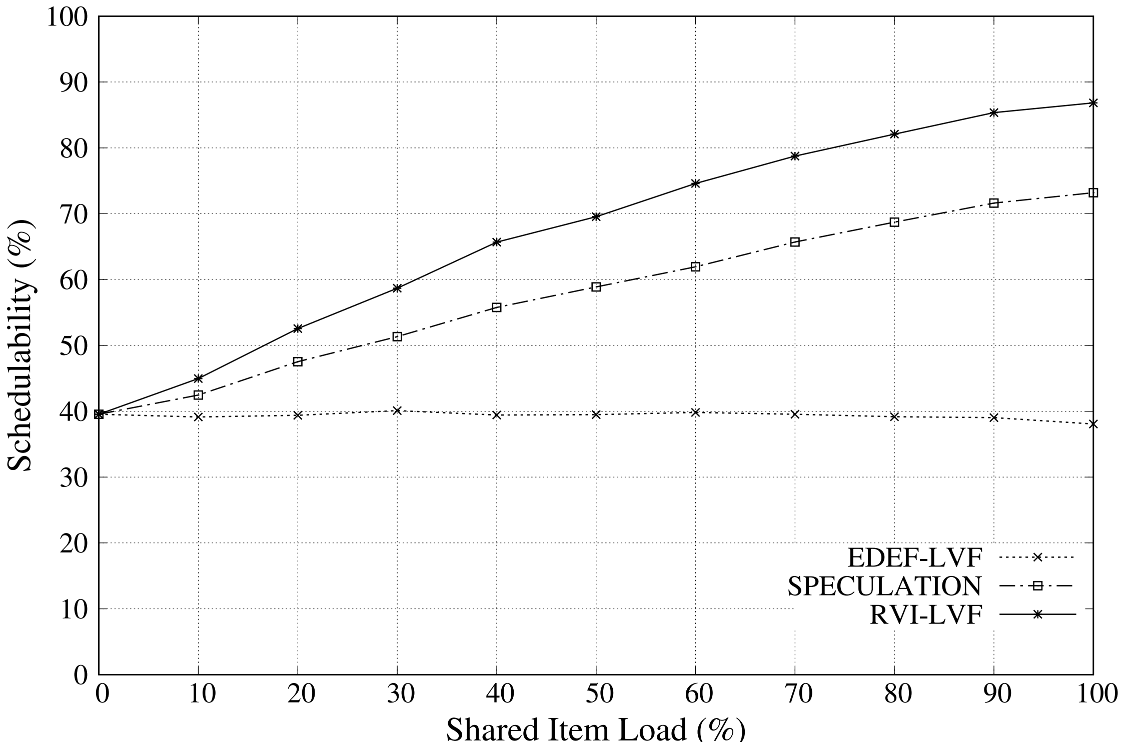


(b) The CPU Utilization of different algorithms with low arrival rate

Figure 5.2: The probability of schedulable tasks with respect to different task utilization and arrival rate



(a) The probability of schedulable tasks with high arrival rate



(b) The probability of schedulable tasks with low arrival rate

Figure 5.3: The probability of schedulable tasks with respect to different load of shared items and arrival rate

CHAPTER 6: RELATED WORK

The idea of data freshness can be dated back in a few decades in the work [4]. Since then, there has been a series of work discussing the topic of real-time database and data acquisition [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. However, not many of the previous works consider the model of noramlly-off sensors. In these work, the concept of data freshness is defined in terms of the whole system instead of one specific data item.

Some of the works focused on working with maintaining freshness for periodic sensors [8, 5, 6, 16], while others focused on the scheduling of profiling sensors or data item in real-time database [7, 11]. A similar concept of our task model was proposed in [12]. MARTE is a model-based data freshness management approach for expressing data freshness requirements. However, this task model does not take into consideration of scheduling items with different deadlines.

Search-based switch and adjustment-based switch was proposed [7] to improve the data freshness and schedulability, respectively. Nevertheless, our work is more similar to the work of [5], where both of the technique is to maintain better data freshness. The main difference is that the objective of [5] is reducing processor workload, while our goal is to improve the schedulability of given task sets. In terms of scheduling problems, different approaches have been proposed. Some of the works focused on revising the update period [14, 8, 16]. Other works focused on the improving the utilization with different scheduling policies [13, 5]. In the work [17], the author examined in a restricted case wheres Shortest Validity First (SVF), which is a similar idea to LVF, is the optimal solution.

Recently, a similar task model was proposed [1], along with a similar problem that was addressed. The decision task in their model collects sensor data items to determine the best route among multiple options. Each of the retrieved sensor data has a validity interval, while the decision task has a deadline constraint to satisfy. After this, a series of work [2, 3] has been published to address the problem further. In [2], it considers the model where there may be multiple decision tasks, but no analysis or properties provided. Later than, EDEF-LVF was proposed [3]. It proves the optimality of multiple decision tasks that do not contain shared data item. Furthermore, it proposed a heuristic algorithm for shared data item based on EDEF-LVF. However, no analysis on the problem of scheduling multiple decision tasks with shared items was conducted. Therefore, the main contribution of this work lies in the analysis of the problem and the proposed heuristic algorithm.

CHAPTER 7: CONCLUSIONS

This work focuses on on addressing the problem of data acquisition to meet information needs of multiple decisions that need overlapping sets of data items. The challenge lies in the analysis of the problem and providing a non-trivial heuristic algorithm that improves schedulability while meeting deadlines and data validity constraints. An optimal scheduling policy, Algorithm 4.1, was proposed to solve a simplified sub-problem. A heuristic scheduling policy, RVI-LVF was then composed to solve the general problem based on properties and observations from Algorithm 4.1. Finally, the scheduling policy was evaluated compared to prior heuristic algorithms. The new algorithm outperforms prior work. The performance evaluation shows a schedulability improvement of up to 40.3% and 17.2% comparing to EDEF-LVF and SPECULATION, respectively. Future work will focus on extensions of the proposed algorithm to accommodate more nuanced network models (such as cases where multiple different channels exist that can be leveraged in parallel). Extensions may also includle approximation bounds with respect to optimal performance. Specifically, to date, no equivalents to utilization bounds, resource augmentation bounds, or capacity augmentation bounds were derived for the problem addressed in this work. Such bounds would be interesting to derive in future work for different models of the bottlenecked communication channel (or channels) over which data are collected.

REFERENCES

[1] S. Hu, S. Yao, H. Jin, Y. Zhao, Y. Hu, X. Liu, N. Naghibolhosseini, S. Li, A. Kapoor, W. Dron, L. Su, A. Bar-Noy, P. Szekely, R. Govindan, R. Hobbs, and T. F. Abdelzaher, "Data acquisition for real-time decision-making under freshness constraints," in *2015 IEEE Real-Time Systems Symposium*, 2015, pp. 185–194.

[2] J. E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, R. Hobbs, and W. Dron, "On maximizing quality of information for the internet of things: A real-time scheduling perspective (invited paper)," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016, pp. 202–211.

[3] J. E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, and R. Hobbs, "Sporadic decision-centric data scheduling with normally-off sensors," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 135–145.

[4] K. Ramamritham, "Real-time databases," in *Distrib Parallel Databases*, 1993, pp. 199–226.

[5] M. Xiong, S. Han, and K.-Y. Lam, "A deferrable scheduling algorithm for real-time transactions maintaining data freshness," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, Dec 2005, pp. 11 pp.–37.

[6] M. Xiong, S. Han, K. Y. Lam, and D. Chen, "Deferrable scheduling for maintaining real-time data freshness: Algorithms, analysis, and results," *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 952–964, July 2008.

[7] S. Han, D. Chen, M. Xiong, and A. K. Mok, "Online scheduling switch for maintaining data freshness in flexible real-time systems," in *2009 30th IEEE Real-Time Systems Symposium*, Dec 2009, pp. 115–124.

[8] K. D. Kang, S. H. Son, and J. A. Stankovic, "Managing deadline miss ratio and sensor data freshness in real-time databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 10, pp. 1200–1216, Oct 2004.

[9] K.-D. Kang, S. H. Son, J. A. Stankovic, and T. F. Abdelzaher, "A qos-sensitive approach for timeliness and freshness guarantees in real-time databases," in *Proceedings 14th Euromicro Conference on Real-Time Systems. Euromicro RTS 2002*, 2002, pp. 203–212.

[10] K.-D. Kang, S. H. Son, and J. A. Stankovic, "Star: secure real-time transaction processing with timeliness guarantees," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, 2002, pp. 303–314.

[11] B. Kao, K.-Y. Lam, B. Adelberg, R. Cheng, and T. Lee, "Maintaining temporal consistency of discrete objects in soft real-time database systems," *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 373–389, March 2003.

[12] N. Louati, R. Bouaziz, C. Duvallet, and B. Sadeg, "Managing data freshness with marte in real-time databases," in *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, June 2013, pp. 1–6.

[13] T. Gustafsson and J. Hansson, "Data management in real-time systems: a case of on-demand updates in vehicle control systems," in *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.*, May 2004, pp. 182–191.

[14] T. Gustafsson and J. Hansson, "Dynamic on-demand updating of data in real-time database systems," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04. New York, NY, USA: ACM, 2004. [Online]. Available: http://doi.acm.org/10.1145/967900.968074 pp. 846–853.

[15] S.-J. Ho, T.-W. Kuo, and A. K. Mok, "Similarity-based load adjustment for real-time data-intensive applications," in *Proceedings Real-Time Systems Symposium*, Dec 1997, pp. 144–153.

[16] M. Xiong, Q. Wang, and K. Ramamritham, "On earliest deadline first scheduling for temporal consistency maintenance," *Real-Time Systems*, vol. 40, no. 2, pp. 208–237, Nov 2008. [Online]. Available: https://doi.org/10.1007/s11241-008-9055-4

[17] M. Xiong and K. Ramamritham, "Deriving deadlines and periods for real-time update transactions," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 567–583, May 2004.