MinMax: A Sampling Interval Control Algorithm for Process Control Systems

Xiuming Zhu, Pei-Chi Huang, Song Han, Aloysius K. Mok
Univ. of Texas at Austin
xmzhu, peggy, shan, mok@cs.utexas.edu

Deji Chen, Mark Nixon
Emerson Process Management
deji.chen, mark.nixon@emerson.com

Abstract—The traditional sampling method in process control systems is based on a periodic task model. This is because controllers are executed in a strictly periodic manner. Sensors sample the process data and send it periodically to the appropriate controllers through a communication system such as the fieldbus. Since the fieldbus is shared by multiple sensors, there is some delay (control loop latency) between the sampling and control actions. In order to minimize the control loop latency, a higher than necessary sampling frequency is typically adopted, which results in unnecessary waste of energy. In this paper, we propose *MinMax*: a sampling interval control algorithm for tackling this problem. In *MinMax*, sampling tasks are not periodic but have both maximum and minimum distance constraints. This sampling model has advantages that are especially important in the domain of wireless control for industrial automation. We shall then discuss the jitter property of sampling schemes under this model and propose algorithms for controlling the sampling intervals of sensors in terms of the *MinMax* problem (*UMinMax*) which we shall introduce. Though this problem is NP-hard in general, even for special case of unit-time tasks, we show how to reduce *MinMax* to well-studied scheduling models such as Liu and Layland-type periodic models and pinwheel models, at the expense of some loss of schedulability. These reductions allow us to derive efficient schedulability tests that can be used to solve the sampling interval control problem in practice. Simulations are used to compare the performance of different *UMinMax* schedulers in two key figures of merit: the acceptance ratio and the jitter ratio. Simulation of a process control system model also shows that *UMinMax* can reduce about 40% of the traffic load on the communication system which is especially important for energy-aware wireless process control applications.

Keywords—minimum separations; maximum distance constraints; real-time scheduling; PID control

I. INTRODUCTION

Fieldbus is widely used to provide real-time guarantee in process control systems. Typically, there are multiple devices (sensors, controllers) connecting to a fieldbus and the communications between sensors and related controllers are scheduled by a centralized module: the fieldbus scheduler, as shown in Figure 1. Traditionally, fieldbus scheduling algorithm is based on a periodic model: a controller updates the actuator in a strictly periodic manner according to the latest process data sent by a related sensor. Since the fieldbus is shared by many sensors, a sensor often may not be able to send its data immediately after sampling. Thus, there is usually a time delay (control loop latency) between sampling and the control action.

In order to minimize the latency, current practice typically employs smaller sampling periods than is necessary and this results in unnecessary waste of energy. This problem is especially serious when the communication is based on wireless transmission.

In this paper, we propose a more flexible sampling interval control model (*MinMax*) that applies to fieldbus scheduling. In *MinMax*, tasks have both maximum distance constraint and

Fig. 1: An example of fieldbus

minimum distance constraint. The maximum distance constraint stipulates that two consecutive sampling jobs of a sensor should not be scheduled too far apart; the minimum separation constraint stipulates that two consecutive sampling jobs should not be scheduled too close. As far as we know, there has not been any systematic study so far on sampling interval control schemes that take into account both types of constraints together, although there are significant advantages to use the *MinMax* scheme.

Figure 2 shows a typical response graph of a controlled process. The upper graph (curve), labeled process output, shows the measured process variable value; the lower graph (a step), labeled (process input) shows the value written to the actuator that affects the process state. For example, the process output could be the fluid speed inside a pipe and the process input could be the position of the valve regulating the pipe. If one opens up the valve by a ΔI amount, the fluid speed will stay the same for T_D (Deadtime), then slowly increase until it settles at a higher speed ΔO and stay at this speed. The slope of the curve determines how long (Time Constant) for it to reach 63% of the final rise. To maintain the speed at a constant rate, the valve has to be constantly adjusted to counter the fluctuations such as the upstream pressure variations. The row of big arrows in Figure 2 denote when the control algorithm is executed; and the row of the small arrows indicate when the process state value, the sensor sample value arrives.

There are several observations to note in this figure that help to explain the technical issue explored in this paper. The relevant technical terms are shown in *italics* for ease of reference.

- The control algorithm executes strictly periodically (the big arrows). In order to keep the controller updated with fresh process data, at least one sensor data sample should come between two consecutive executions of control algorithm. (*maximum distance constraint*)
- The sampled data may not arrive in a strictly periodic manner because the fieldbus is shared by multiple user processes. (*the data arrival variation - jitter*)

Fig. 2: A process control model

- A controller does not require neither a fixed start time nor a fixed period. (*non-rigid periodic models*)
- The sample rate does not need to be faster than is needed because the control algorithm only needs the latest process value. (*minimum separation*)
- The need to avoid overly fast sampling rate is important for energy-aware applications, especially for the adoption of wireless control in the process automation industry. Reduced sampled rate minimizes energy usage and equipment wear. And power usage reduction is critical for battery-powered wireless sensors. (*minimum separation*)

In order to exercise effective control of the sensor sampling intervals, we shall first study the general job scheduling problem with both minimum separation and maximum distance constraints (*MinMax*) and then apply it to process control systems. Since control actions are often scheduled in time slots, we pay special attention to the case when all tasks have the same unit execution time. We call this scheduling problem the unit-time *MinMax* problem (*UMinMax*). The major contributions of this paper are as follows.

- We propose a new *MinMax* sensor scheduling model and investigate the relationship between jitter and *MinMax*. Solutions of the associated scheduling problems gives us effective control of the sampling intervals. Although the scheduling problem is NP-hard, we show that they can be solved in practice by reduction to known scheduling problems at the expense of some loss of schedulability.
- We propose several solutions on how to reduce *MinMax* to periodic task models and pinwheel models. For every reduction, we derive sufficient conditions that yield efficient schedulability tests.
- We apply *MinMax* solution to a realistic process control system typical of fieldbus applications today and show that it outperforms current solutions.

The remainder of the paper is structured as follows. Section II summaries the background and related works. Section III gives the formal definition of *MinMax* and lists some analysis results. Section IV and Section V describe how to reduce *MinMax* to periodic models and pinwheel models respectively. Reduction algorithms are provided. Section VI shows the simulation results comparing different schedulers. Section VII describes the application of *MinMax* in a process control system. Finally, Section VIII concludes the paper.

TABLE I: Symbols and definitions

Symbol	Definition
$\mathcal{T}$	Task set
τ_i	Task i
S	Schedule
min_i	Minimum distance separation for task i
max_i	Maximum distance constraint for task i
J	Job
$s_{i,j}(f_{i,j})$	Starting time (finishing time) of job i, j
n	Task number in the task set $\mathcal{T}$
m	number of tasks in a subgroup of task set $\mathcal{T}$
P_i	Period of task i (periodical model)
D_i	Deadline of task i (periodical model)
C_i	Execution time of task i
α_i	Data Validity interval of task i (<i>More-Less</i>)
X_i	Temporal data i (<i>More-Less</i>)
U_n	Utilization ratio of task set (periodical model)
A	A pinwheel instance
a_i	Distance integer (pinwheel model)
$\rho(\mathcal{T})$	Density of task set $\mathcal{T}$ (pinwheel model)
l	Cycle length for two distinct integer pinwheel schedule

II. BACKGROUND AND RELATED WORK

In this section, we briefly summarize results related to distance constrained job scheduling. Roughly, they can be divided into two groups: about maximum distance constraint and about minimum separation constraint. *Distance Constrained Task System* is first proposed by [1] and refers to job scheduling with only maximum distance constraint. In order to distinguish maximum distance constraint and minimum distance constraint, we follow the definitions in [2]. We use the term distance constraint to refer to maximum distance constraint, and the term separation constraint to refer to minimum distance constraint. Formal definitions of some of the often-used symbols are list in Table I.

A. Job Scheduling With Maximum Distance Constraints

For this class of scheduling problems, the distance between two consecutive jobs must not be larger than some maximum allowed value. Pinwheel [3] belongs to this group. In pinwheel, execution time of all jobs is one time unit and the distance between two consecutive jobs of the same task cannot exceed some specified integer value. The general pinwheel scheduling problem is known to be NP-hard [3]. However, by applying several kinds of reductions [4] [5] [6], the complexity of schedulability decision can be reduced to $O(n)$ at the expense of some schedulability loss: schedulability is guaranteed only if the density of the task set is smaller than some bound. In subsequent work, [1] applies pinwheel to scheduling jobs with real number distance constraints in a preemptive environment. It is noteworthy that [1] also enforces fixed separation constraints between two consecutive jobs to avoid over-execution of high priority tasks. The non-preemptive maximum distance constraint job scheduling problem is discussed in [2], where the NP-completeness of the job scheduling problem with arbitrary maximum distance constraint is proved. [7] investigates TDMA frame-based jobs scheduling with both average and maximum distance constraints. It proposes two pseudo-polynomial time scheduling algorithms but it does not provide any sufficient or necessary conditions for schedulability.

B. Job Scheduling With Minimum Separation Constraints

While there are quite a few papers about scheduling jobs with maximum distance constraint, there is relatively little work on distance constraints on the opposite side. [8] is the first one to

propose the dual problem of job scheduling with temporal distance constraints, named *Job Scheduling with Separation Constraints*. In this model, distances of two jobs must be no smaller than certain value. Similarly, the generalized scheduling problem is NP-hard [8].

We note that a number of authors refer to the minimum separation as the distance between the request times of two consecutive jobs, not the actual execution start times. In regard to the latter interpretation, the schedules produced in many of the task models will violate the minimum constraints. For example, in a periodic task model, although the period may be no smaller than the minimum separation time, two jobs of the task may be scheduled with zero separation when the first one runs to the end of a period and the second one runs at the start of the next period. In our case, minimum separation applies to the actual execution and not the request times.

III. PROBLEM FORMULATION

Definition III.1: A *MinMax* task set $\mathcal{T}$ is a set of *MinMax* tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$. Each task τ_i is a 3-tuple $(C_i, \min_i, \max_i)$, $i \in [1, n]$, $\max_i \geq \min_i + 2 * C_i$, $\min_i, \max_i, C_i \in \mathbb{R}^+$. C_i is the execution time. $\min_i$ is the minimum separation time. $\max_i$ is the maximum distance. τ_i consists of an infinite sequence of jobs $J_{i,1}, J_{i,2}, J_{i,3}, \dots$. Let $s_{i,j}, f_{i,j}$ denote the starting time and ending time of executing $J_{i,j}$, $j \geq 1$ respectively. For two consecutive jobs $J_{i,j}, J_{i,j+1}$, it is required that $s_{i,j+1} - f_{i,j} \geq \min_i$ and $f_{i,j+1} - s_{i,j} \leq \max_i$.

An illustration is shown in Figure 3.

We could have defined $s_{i,j+1} - s_{i,j} \leq \max_i$ or $f_{i,j+1} - f_{i,j} \leq \max_i$. However, we think our definition is more germane in practice, i.e., two consecutive jobs are completely within $\max_i$ of each other. Besides, these definitions are readily be convertible to each other. Similar reasoning applies to our definition of $\min_i$.

Note that in our definition, $s_{i,j}$ is not the ready time of a job but the time when $J_{i,j}$ is actually executed. $f_{i,j}$ is the time point at which the last part of $J_{i,j}$ is executed when it completes. This is important when we consider both types of constraints, otherwise $\min$ would be meaningless.

There have been quite a bit of work on jitter bounds. Reference [9] provides a summary. The author also studied the arrival jitter, in which jobs arrive unevenly. The finish jitter is paid less attention partly because one could arbitrarily define a job to be finished at its deadline if the last part of a job is completed arbitrarily close to the deadline, no matter how smaller the last part of the job is; in this way, we could have 0 jitter. However, this interpretation of jitter does not sit well with real world applications in which the actual job finish time usually is expected to be related to the freshness of the data used.

Example III.1: A WirelessHART star network is used to monitor a process. The control algorithm is run in the gateway. The sensors and actuators are one hop from the gateway. Periodically, the sensor sends data to the gateway, which executes the control algorithm, and then sends the output to the actuator. When the control algorithm is run periodically, the best control performance is achieved when the sensor data also arrives punctually, i.e., no jitter, in a period.

Definition III.2: A jitter constraint task set $\mathcal{T}$ is a set of tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$. Each task τ_i is a 3-tuple $(C_i, \delta_i, \max_i)$, $i \in$

Fig. 3: An illustration of MinMax task

$[1, n]$, $C_i, \delta_i, \max_i \in \mathbb{R}^+$. C_i is the execution time. δ_i is the maximum jitter. $\max_i$ is the maximum distance. τ_i consists of an infinite sequence of jobs $J_{i,1}, J_{i,2}, J_{i,3}, \dots$. Let $s_{i,j}, f_{i,j}$ denote the starting time and ending time of executing $J_{i,j}$, $j \geq 1$ respectively. For two consecutive jobs $J_{i,j}, J_{i,j+1}$, it is required that $f_{i,j+1} - s_{i,j} \leq \max_i$ and that the maximum and minimum $f_{i,j+1} - f_{i,j}$ do no differ more than δ_i .

An illustration is shown in Figure 4.

Theorem III.1: If there exists a schedule for an *MinMax* instance $\mathcal{T} = \{(C_i, \min_i, \max_i), i \in [1, n]\}$, then there exists a schedule for a jitter constraint instance $\mathcal{T} = \{(C_i, \delta_i, \max_i), i \in [1, n]\}$, where $\delta_i = \max_i - \min_i - 2 * C_i$.

Proof Sketch: For a valid *MinMax* schedule, the maximum $f_{i,j+1} - f_{i,j}$ happens when $J_{i,j}$ is run immediately at $s_{i,j}$ for C_i to finish and $J_{i,j+1}$ is finished $\max_i$ later, which is $\max_i - C_i$. the minimum $f_{i,j+1} - f_{i,j}$ happens when $J_{i,j}$ and $J_{i,j+1}$ is separated by $\min_i$ and $J_{i,j+1}$ runs immediately at $s_{i,j+1}$ for C_i to finish, which is $\min_i + C_i$. So the jitter is no more than $(\max_i - C_i) - (\min_i + C_i) = \max_i - \min_i - 2 * C_i$. $\square$

Theorem III.1 shows that any valid schedule of an *MinMax* task set also provides a certain degree of jitter guarantee.

Note the converse is not true. A valid schedule for a single jitter constraint task could have this task run on the processor continuously without break. In this case the job separation is 0 and the jitter is 0. No minimum separation could be guaranteed.

In this paper, we focus on the special case of *MinMax* where all execution times are one. We call it unit-time *MinMax* (*UMinMax*). The data transmission jobs in Example III.1 are all scheduled in one time slot regardless of the message size.

In *Discrete Time Model*, the granularity of time measurements is in terms of integral multiples of some basic time quantum and the time intervals are always measured between integral time points and given in integral time units (time slots). No task period can span a time interval that starts in the middle of a time slot and ends in the middle of another one.

Based on Definition III.1 and *Discrete Time Model*, *UMinMax* can be defined as follows.

Definition III.3: In *UMinMax*, the execution time of all tasks $C_i, i \in [1, n]$ is one time unit (time slot) and all time parameters $C_i, \min_i, \max_i, s_{i,j}, f_{i,j}, i, j \in [1, n]$ satisfy the requirements of *Discrete Time Model*.

Since $C_i = 1$, for ease of expression, we can represent the distance definitions between two consecutive jobs as $s_{i,j+1} - s_{i,j}$. For the rest of the paper, we will use $\{(\min_i, \max_i), i \in [1, n]\}$ to represent an instance of *UMinMax*.

Now, we shall prove that the schedulability decision problem of *UMinMax* is NP-hard by presenting that it is NP-Complete.

Theorem III.2: *UMinMax* is NP-complete in the strong sense.

Proof: It is easy to see that the problem is in NP. In order to prove its NP-Completeness, we reduce general multi-level unit-

Fig. 4: An illustration of jitter constraint task

time job scheduling problem with temporal distance constraints (general MUJSD) [2] to it. The general MUJSD can be described as follows: Given a set of Jobs $J_set = \{J_1, J_2, \dots, J_n\}$, in which each job J_i has the execution time C_i , ready time R_i and deadline D_i . In general MUJSD, $C_i = 1, R_i = 0, i \in [1, n]$ and the job set J_set is divided into chains of jobs. There are two types of jobs: head job set H_set and tail job set T_set . Each job H_i in $H_set = \{H_1, H_2, \dots, H_m\}$ has a deadline D_i . Jobs in T_set are grouped into m subsets $T_set_1, T_set_2, \dots, T_set_m$, where $T_set_i = \{J_{i,1}, J_{i,2}, \dots, J_{i,k_i}\}, k_i \geq 0$. T_set_i can be viewed as the set of tail jobs after the head job H_i , while $J_{i,j}$ is the j th tail of H_i and $J_{i,0} = H_i$. The objective is to find a schedule that meets the temporal distance constraint requirement which job $J_{i,j}$ must be started within max_i time units after its immediate predecessor $J_{i,j-1}$ is started and $max_i < m$. It is straightforward to see that the general MUJSD is only a special case of *UMinMax* with $min_i = 0$. Since general MUJSD has been proved NP-complete in the strong sense, *UMinMax* is also NP-complete in the strong sense. $\square$

By definition, pinwheel problem is a special case of *UMinMax* problem. By following the existence proof of cyclic pinwheel schedule, we can also claim the existence of cyclic schedule for *UMinMax*:

Corollary III.1: If there exists a schedule for a *UMinMax* instance $\mathcal{T} = \{(min_i, max_i), i \in [1, n]\}$, then $\mathcal{T}$ has a cyclic schedule whose cycle length is no greater than $\prod_{i=1}^n max_i$.

In the next two sections we shall look at how to schedule *MinMax* tasks by reducing the *MinMax* model to other well studied task models: periodic task model and pinwheel scheduling model. The idea is that once we have done the conversion, we can use all the tools at our disposal for those models to solve the problems of the *MinMax* model. The trick is to make sure that the schedule produced with the traditional models will satisfy not only the max requirement but also the min requirement. To accomplish this, we give six reductions for *MinMax*. Section IV looks at the periodic models. Section V applies pinwheel results to *UMinMax* task sets. A comparison of the reductions is presented in Section VI.

IV. SOLVING *UMinMax* PROBLEM WITH PERIODIC SCHEDULING MODEL

It has been noted that if the deadline equals the period, a scheduling policy such as Earliest Deadline First (EDF) or Rate Monotonic (RMA) could schedule two consecutive jobs side by side, which will violate any minimum separation requirement. Fortunately, we can take advantage of task models in which the deadline is less than the period.

We can first reduce the *MinMax* problem to periodic model as in Figure 5. The relationship can be expressed as follows.

$$P_i + D_i \leq max_i \quad (1)$$

Fig. 5: Periodical model of MinMax

$$P_i - D_i \geq min_i \quad (2)$$

$$D_i \geq C_i \quad (3)$$

It is straightforward to arrive at the following conclusion.

Theorem IV.1: For an *MinMax* task set and any periodic task set whose parameters satisfy equations 1, 2, and 3. Any valid schedule for the periodic task set is also a valid schedule for the *MinMax* set.

Note the converse is not true as *MinMax* allows jobs to drift away from the multiples of a period.

The scheduling problem of periodic model with $D_i < P_i$ has been proved NP-hard [10]. We list the relevant scheduling results in the following subsection. In the next subsection we show how to apply them to our model.

A. Background

Theorem IV.2: [11] Consider N tasks where C_i, D_i are the execution time and deadline of tasks $\tau_i, i = 1, \dots, N$, respectively. The task set is schedulable by the EDF if

$$\sum_{i=1}^N \frac{C_i}{D_i} \leq 1 \quad (4)$$

Theorem IV.3: [12] Consider N tasks where C_i, D_i, P_i are the execution time, deadline, period of tasks $\tau_i, i = 1, \dots, N$, respectively. Suppose $i \leq j \Rightarrow D_i \leq D_j$. The task set is schedulable if

$$L^* \geq \sum_{i=1}^N \left(\frac{L^* - D_i}{P_i} + 1 \right) C_i \quad (5)$$

$$\sum_{i=1}^j C_i \leq D_j, j = 1, \dots, n. \quad (6)$$

where

$$L^* = \begin{cases} D_2 & : D_1 + P_1 \leq D_2 \\ \min_{i=1}^N (P_i + D_i) & : \text{otherwise} \end{cases}$$

Theorem IV.2 and IV.3 are for EDF scheduling. [13] introduced *More-Less* model for fixed priority scheduling. Given a task set $\mathcal{T} = \{\tau_i\}_{i=1}^n$, each task is associated with a distance constraint max_i and an execution time C_i , if certain pre-conditions are satisfied, *More-Less* is able to find the maximum P_i for each task and use deadline monotonic algorithm to schedule it. Please refer to [13] for details.

B. Reduction

In this subsection, we shall show how to deduct $\{(P_i, D_i)\}$ from $\{(min_i, max_i)\}$ to make an *UMinMax* problem most schedulable.

For Theorem IV.2, the answer is straightforward. We are trying to enlarge D_i , and the largest D_i is $\lceil (max_i - min_i)/2 \rceil$. Then, as long as $\sum_{i=1}^N 1/\lceil (max_i - min_i)/2 \rceil \leq 1$ and $D_i \leq P_i$, it will be schedulable by EDF. This reduction process takes $O(N)$.

TABLE II: Reducing *MinMax* to periodical model

Reduction	P_i, D_i in terms of $\max_i, \min_i$
EDF [11]	$D_i = \lceil \max_i - \min_i / 2 \rceil$ $P_i = \max_i - D_i \wedge P_i \geq D_i$
EDF [12] (<i>UMinMax</i>)	Pre-condition: $\sum_{i=1}^j C_i \leq D_j, j = 1, \dots, n$
	$D_1 = 1, P_1 = \max_1$ $D_2 \geq D_1 + P_1 : D_i = \lceil (\max_i + 2 - \min_i) / 2 \rceil$ $P_i = \max_i + 1 - D_i \wedge P_i \geq D_i$
	$D_2 < D_1 + P_1 : D_i = \lceil (\max_i + 2 - \min_i) / 2 \rceil$ $P_i = \max_i + 1 - D_i \wedge P_i \geq D_i$
<i>More-Less</i>	1. Check whether <i>More-Less</i> pre-conditions are satisfied 2. P_i, D_i will be computed by <i>More-Less</i> 3. Check whether $P_i - D_i \geq \min_i$ holds

We are only able to apply Theorem IV.3 to *UMinMax* reduction. We can first check whether equation 6 is satisfied or not. If yes, P_i, D_i can be computed, otherwise, no need to continue. For equation 5, it can be firstly assumed that L^* is fixed. Since $C_i = 1$, we need to minimize $\frac{L^* - D_i}{P_i}$ to make it more promising to be schedulable. By intuition, both D_i, P_i should be maximized. However, since $D_i + P_i \leq \max_i$, we need to consider whether P_i should be bigger or otherwise. This involves comparing $\frac{L^* - (D_i + 1)}{P_i - 1}$ and $\frac{L^* - (D_i)}{P_i}$.

Let us consider the first case $L^* = D_2$. Since $\mathcal{T}$ is sorted in a nondecreasing order of deadline, it is easy to prove that if $i \geq 2$, $\frac{L^* - (D_i + 1)}{P_i - 1} \leq \frac{L^* - (D_i)}{P_i}$ and if $i = 1$, $\frac{L^* - (D_i + 1)}{P_i - 1} \geq \frac{L^* - (D_i)}{P_i}$. The former can be proved as follows.

$$\begin{aligned} L^* P_i - (D_i + 1) P_i &\leq L^* P_i - L^* - D_i P_i + D_i \\ \Leftrightarrow -P_i &\leq -L^* + D_i \\ \Leftrightarrow L^* &\leq P_i + D_i \quad (i \geq 2) \end{aligned}$$

Similarly, we can prove the latter.

Based on above deduction, if $i = 1$, D_i should be smallest if possible while $i \geq 2$, D_i should be largest if possible. In this case, according to equation 6, $D_1 = 1, P_1 = \max_1$, $D_i = \lceil \frac{\max_i + 2 - \min_i}{2} \rceil, P_i = (\max_i + 1 - D_i) \wedge P_i \geq D_i, i \geq 2$.

Now, we can think about $L^* = \min_{i=1}^N (P_i + D_i)$. Then, $L^* \leq P_i + D_i$ for all i . Thus, $D_i = \lceil \frac{\max_i + 2 - \min_i}{2} \rceil, P_i = (\max_i + 1 - D_i) \wedge P_i \geq D_i$. Same as above, the reduction process also takes $O(N)$.

We also can apply *More-Less* to solve the MinMax problem. First we need to verify whether *MinMax* problem satisfies *More-Less* pre-conditions. Then, we can apply MinMax principle to compute P_i, D_i for τ_i . Finally, we have to make sure whether $P_i - D_i \geq \min_i$ holds. If yes, we can find a solution, otherwise, it fails. The reduction only adds a check procedure in the tail of *More-Less*, which takes $O(n)$ complexity.

The reduction to periodic model is summarized in Table II.

V. SOLVING *UMinMax* PROBLEM WITH PINWHEEL SCHEDULING MODEL

In this section, we will discuss how to apply pinwheel model to solve *UMinMax* problem. Since pinwheel does not consider minimum separation constraints, it appears difficult to satisfy *UMinMax* with a pinwheel schedule. Fortunately, an observation on pinwheel algorithms yields nice properties that we can exploit. In particular, we are able to derive the lower separation bound of a pinwheel schedule. So we take a different approach from those in the previous section. We first use this separation bound as a sufficient condition for making schedulability decision for *UMinMax*. Once it is met, we proceed with the deduction, the

success of which guarantees a valid schedule for the *UMinMax* set.

A. Background

We shall briefly describe the pinwheel problem. Detailed definition of pinwheel can be found in [3]. Given a set of integers $A = \{a_1, a_2, \dots, a_n\}$, a successful schedule S is an infinite sequence over $\{1, 2, \dots, n\}$ such that any subsequence of $a_i, i \in [1, n]$ consecutive entries (slots) contains at least one i . The density of the instance A is defined to be $\rho(A) = \sum_{i=1}^n 1/a_i$. The pinwheel problem is claimed to be NP-hard [3]. However, with integer reductions [4] [5] [6], a subset of pinwheel decision problems can be easily solved. There are four classes of solutions for pinwheel problems, which we will describe in the following subsections.

1) *Single Integer Reduction*: Before introducing one integer reduction, we list several important lemmas and definitions.

Definition V.1: A is special with respect to x means $\forall a_i \in A, \exists j \Rightarrow a_i = x * 2^j$

Lemma V.1: [5] Let pinwheel instances $A = \{a_1, \dots, a_n\}$, $A' = \{a'_1, \dots, a'_n\}$, and $a_i \geq a'_i, i \in [1, n]$. Then, if A' is schedulable, so is A .

Lemma V.2: [3] Given a pinwheel instance $A = \{a_1, \dots, a_n\}$, if $a_i | a_j, i \leq j$, A is schedulable.

Based on above, S_x [5] is proposed to reduce an arbitrary pinwheel instance A to A' , which $a'_i \in A' \wedge a'_i = x * 2^j$, then A can be scheduled easily as long as $\rho(A') \leq 1$.

2) *Two Integers Reduction*: If there are only two distinct numbers in A , as long as the density is smaller than 1, it can be scheduled [4]. In this case, A can be expressed in a compact form as $A = \{(a_1, m_1), (a_2, m_2)\}$, which a_1, a_2 are the integers of frequency and m_1, m_2 are the numbers of tasks respectively. Related important conclusions are listed as follows.

- 1) As long as $m_1/a_1 + m_2/a_2 \leq 1$, it can be scheduled.
- 2) There exists cyclic schedule for it, the length can be

$$H_1(A) = \frac{a_2 * LCM(m_1, a_2 - m_2)}{a_2 - m_2} \quad (7)$$

- 3) $H_1(A)$ are non optimal cyclic lengths and there is an optimal length $LM(A)$
- 4) For $H_1(A)$, $Place1(i)$ and $Place2(i)$ identify i th slot for items of frequency a_1 and a_2 respectively. ($l = H_1(A)$ in following two equations).

$$Place1(i) = i + \lceil \frac{i m_2}{a_2 - m_2} \rceil, 0 \leq i < \frac{l}{a_2} (a_2 - m_2) \quad (8)$$

$$Place2(i) = \lfloor \frac{i a_2}{m_2} \rfloor + 1, 0 \leq i < \frac{l}{a_2} m_2 \quad (9)$$

Based on above conclusions, similar to single integer reduction, we can make use of instances special to two integers. Such instances can be described as follows. Let $A' = A'_1 \cup A'_2$, where A'_1 is special to integer a_1 and A'_2 is special to integer a_2 . Let $m_1 = \lceil a_1 \rho(A'_1) \rceil, m_2 = \lceil a_2 \rho(A'_2) \rceil$. Then, the scheduling problem becomes finding a schedule for the instance $\{(a_1, m_1), (a_2, m_2)\}$. Hence, as long as $\lceil a_1 \rho(A'_1) \rceil / a_1 + \lceil a_2 \rho(A'_2) \rceil / a_2 \leq 1$, A' can be scheduled. Based on this, *SpecialDouble* is proposed in [5].

3) *Three Distinct Integers*: Pinwheel with three distinct integers is firstly mentioned in [3]. A more general case is investigated in [14]. It proposes an $O(1)$ scheduler *Pinwheel3*, which can guarantee to schedule instances with density no bigger than $5/6$. Different from single integer and double integers cases, till now, there is no paper about reduction of three integers pinwheel model, which limits its application seriously. Hence, we only list our lower separation bounds in the next section and can serve as sufficient conditions for schedulability test.

4) *PinFair*: *PinFair* is proposed in [6]. It can be described as follows.

For a pinwheel instance $A = \{(a_1, m_1), (a_2, m_2), \dots, (a_n, m_n)\}$, define $w_i = (m_i + 1)/a_i$ for each $(a_i, m_i), i \in [1, n]$, then, as long as $\sum_i^n w_i \leq 1$, a *PFair* [15] schedule which meets the pinwheel requirements can be generated. The time slot assigned to task τ_i for the j th time should be in a range $[earliest(\tau_i, j), latest(\tau_i, j)]$, where

$$earliest(\tau_i, j) = \lfloor \frac{j}{w_i} \rfloor \quad (10)$$

$$latest(\tau_i, j) = \lceil \frac{j+1}{w_i} \rceil - 1 \quad (11)$$

We shall use the following known floor and ceiling non-equations in our proofs next.

$$\lfloor r+x \rfloor - \lfloor r \rfloor \geq \lfloor x \rfloor \quad r, x \in \mathbb{R} \quad (12)$$

$$\lfloor r \rfloor - \lfloor x \rfloor \geq \lfloor r-x \rfloor \quad r, x \in \mathbb{R} \quad (13)$$

$$\lceil r+x \rceil - \lceil r \rceil \geq \lceil x \rceil \quad r, x \in \mathbb{R} \quad (14)$$

$$\lceil r \rceil + k = \lceil r+k \rceil \quad r \in \mathbb{R} \quad k \in \mathcal{N} \quad (15)$$

B. Reducing *UMinMax* Problem to One Integer Pinwheel Model

As stated before, we shall first derive the minimum distance separation for pinwheel schedulers. Since S_a is only a special case of S_x , we only discuss S_x here. It can be calculated directly from the algorithm:

Corollary V.1: Given a pinwheel instance $A = \{a_1, a_2, \dots, a_n\}, a_1 \leq a_2 \leq \dots \leq a_n$, if it can be scheduled by S_x , then, we can find a schedule such that for a task with frequency $a_i, i \in [1, n]$, the distance separation in the schedule is exactly $x * 2^{\lfloor \log_2(a_i/x) \rfloor}$, where $x \in (a_1/2, a_1]$.

Based on corollary V.1, we only need to check whether $x * 2^{\lfloor \log_2(a_i/x) \rfloor}$ is no smaller than min_i . Algorithm 1 is the pinwheel schedule algorithm revised for *UMinMax*.

In Algorithm 1, d_{sx} on line 7 denotes the density bound of S_x [5]. *SimpleGreedy*(B) on line 30 is a pinwheel scheduler proposed in [3].

The reduction part of Algorithm 1 takes $O(min_max * n)$ time where min_max is the minimum value of max_i s. It is a little different compared to the $O(n)$ complexity of S_x . For pinwheel, S_x only needs to consider the cases when min_max is smaller than n because if min_max is no smaller than n , the density will be no greater than 1 and no need to search x . However, for *UMinMax*, it is necessary to search all the possibilities. For example, let a *UMinMax* instance $\mathcal{T} = \{(3, 5), (5, 6), (6, 7)\}$ and it is reduced to a pinwheel instance as $A = \{5, 6, 7\}$. For S_x , it

Algorithm 1 *UMinMax* pinwheel scheduler

```

1: // Input: A UMinMax instance  $\{(min_i, max_i)\}, i \in [1, n]$ 
2: // Output: Either a UMinMax schedule or failure
3: // Reduction part
4: // Find the smallest value  $min\_max$  from all  $max_i$ s
5:  $x \leftarrow min\_max$ 
6: // Check whether it is over the density bound of  $S_x$ 
7: if  $\sum_i^n 1/max_i \leq d_{sx}$  then
8:   repeat
9:      $flag = true$ 
10:     $B \leftarrow$  specialize  $\{max_i, i \in [1, n]\}$  with respect to  $x$ 
11:    if  $\rho(B) \leq 1$  then
12:      // Check the minimum separation requirement
13:      for all  $i$  in  $[1, n]$  do
14:        if  $min_i > x * 2^{\lfloor \log_2(max_i/x) \rfloor}$  then
15:           $flag = false$ ; break; // Check failure
16:        end if
17:      end for
18:    else
19:       $flag = false$  // We have to search next  $x$ 
20:    end if
21:     $x \leftarrow x - 1$ 
22:  until  $x \leq min\_max/2$  or  $flag = true$ 
23: else
24:   return failure;
25: end if
26: if  $flag == false$  then
27:   return failure;
28: end if
29: // Scheduling Part
30: Generate a schedule with SimpleGreedy( $B$ )

```

does not need search x and x is 5. However, for *UMinMax*, $x = 5$ does not satisfy the minimum separation constraint of (6, 7) and it has to search x until $x = 3$.

Also, based on Corollary V.1, we can claim a sufficient condition of schedulability for *UMinMax*.

Corollary V.2: Consider a *UMinMax* instance $\mathcal{T} = \{(min_i, max_i), i \in [1, n]\}$, if it satisfies $\sum_{i=1}^n (1/max_i) \leq d_{sx}$ and $\forall i \in [1, n], max_i/min_i \geq 2$, then there exists a schedule for it. Furthermore, the max_i/min_i ratio is bounded from below by 2.0.

Proof: First, since $\sum_{i=1}^n (1/max_i) \leq d_{sx}$, it can be scheduled by S_x . Then, for each task τ_i , suppose the distance of two consecutive jobs in the schedule is $x * 2^{\lfloor \log_2(a_i/x) \rfloor}$, then $min_i \leq max_i/2 < x * 2^{\lfloor \log_2(a_i/x) \rfloor - 1} < x * 2^{\lfloor \log_2(a_i/x) \rfloor}$. Hence, the schedule exists.

For the latter, suppose there exists a real number r smaller than 2, which it can be claimed as a bound. Then, we can always find a counter example. Consider an instance $\mathcal{T}$, let all tasks except τ_i satisfy $\forall j \in [1, n] \wedge j \neq i, min_j = max_j = x * 2^{k_j}$, where $k_j \in \mathcal{N}$, and for $\tau_i, min_i = x * 2^{k_i} + 1, max_i = x * 2^{k_i+1} - 1$. It is easy to see $\mathcal{T}$ is not schedulable. However, the ratio max_i/min_i can be close enough to 2 and can exceed any r smaller than 2. Hence, such r does not exist. $\square$

C. Reducing UMinMax Problem to Double Integer Pinwheel Problem

1) *lower separation bound deduction* : Here, only non-optimal cycle length will be discussed. One main reason is that the optimal case tends to have larger distance range than the non-optimal case, which in turn reduces the acceptance ratio for UMinMax. For non-optimal cycle length case, we have the following result.

Corollary V.3: For a pinwheel instance with two distinct integers $A = \{(a_1, m_1), (a_2, m_2)\}$, we can find a cyclic pinwheel schedule which satisfies

- For tasks with frequency a_1 , the distances between jobs are in the range $[\lfloor \frac{m_1 a_2}{a_2 - m_2} \rfloor, a_1]$. The bound is tight.
- For tasks with frequency a_2 , the distances between jobs can be exact a_2 .

Proof: We need only to prove $Place1(i + m_1) - Place1(i) \geq \lfloor \frac{m_1 a_2}{a_2 - m_2} \rfloor$. The rest are already proved in [4].

$$\begin{aligned} & Place1(i + m_1) - Place1(i) \\ &= i + m_1 + \lceil \frac{(i+m_1)m_2}{a_2 - m_2} \rceil - i - \lceil \frac{im_2}{a_2 - m_2} \rceil \\ &= m_1 + \lceil \frac{(i+m_1)m_2}{a_2 - m_2} \rceil - \lceil \frac{im_2}{a_2 - m_2} \rceil \\ &\geq m_1 + \lfloor \frac{m_1 m_2}{a_2 - m_2} \rfloor \quad (\text{See non-equation 14}) \\ &= \lfloor \frac{m_1 a_2}{a_2 - m_2} \rfloor \quad (\text{See equation 15}) \end{aligned}$$

Now, we need to prove $\lfloor \frac{m_1 a_2}{a_2 - m_2} \rfloor$ is tight. This can be proved by a simple example. Let $A = \{(8, 5), (3, 1)\}$. The lower separation bound of distance range for tasks with frequency 8 is 7, which is just $\lfloor \frac{m_1 a_2}{a_2 - m_2} \rfloor$. $\square$

The above result is not satisfactory because $\lfloor \frac{m_1 a_2}{a_2 - m_2} \rfloor$ may be unnecessarily small. We hope to make the distance range smaller. This can be implemented by ‘padding’ with empty tasks. The following shows a more interesting result.

Lemma V.3: Given $a_1, a_2, m_1, m_2 \in \mathcal{N}$, where $m_1 < a_1, m_2 < a_2, a_1 a_2 - m_1 a_2 - m_2 a_1 \geq 0, \exists m'_1, m'_2 \Rightarrow a_1 a_2 - m'_1 a_2 - m'_2 a_1 \in [0, \min(a_1, a_2))$, where $m'_1 \geq m_1, m'_2 \geq m_2$.

Proof: Without lose of generality, suppose $\min(a_1, a_2) = a_2$. If $a_1 a_2 - m_1 a_2 - m_2 a_1 \in [0, a_2)$, then let $m'_1 = m_1, m'_2 = m_2$. If not, let $m'_1 = \lfloor (1 - m_2/a_2)a_1 \rfloor, m'_2 = m_2, a_1 a_2 - m'_1 a_1 - m'_2 a_2 \in [0, a_2)$. Since in this case, $a_1 a_2 - m_1 a_2 - m_2 a_1 > a_2, m_1 < (1 - m_2/a_2)a_1 - 1 < m'_1$. $\square$ Based on Lemma V.3, Corollary V.3 can be improved.

Corollary V.4: For pinwheel problem with two distinct numbers $A = \{(a_1, m_1), (a_2, m_2)\}$, we can find a cyclic pinwheel schedule which satisfies

- For tasks with frequency a_1 , the distances between jobs are in the range $[\lfloor \frac{(1-m_2/a_2)a_1 a_2}{(a_2 - m_2)} \rfloor, a_1]$. The bounds are tight.
- For tasks with frequency a_2 , the distances between jobs can be exact a_2 .

Proof: By plugging in the maximum possible value of m_1 as defined in Lemma V.3 the result easily follows. $\square$ Example V.1 shows the differences between ‘padding’ and without ‘padding’.

Example V.1: $A = \{(8, 1), (3, 1)\}$, without padding, the distance range between tasks with frequency 8 can be $[1, 8]$; however, after padding 4 empty tasks with frequency 8, the range narrows down to $[7, 8]$.

It is easy to extend Corollary V.4 to double integer reduction case in the following.

Corollary V.5: For a pinwheel instance $A = \{a_1, a_2, \dots, a_n\}$ solved by double integer reduction, let $A = A_x \cup A_y$, which

A_x is special to integer x and A_y is special to integer y . Let $p = \lceil x\rho(A_x) \rceil, q = \lceil y\rho(A_y) \rceil$. There exists a cyclic schedule which satisfies

- For each task with frequency $a_i \in A_x, i \in [1, n]$, the actual distances between jobs in the final schedule are in the range $[k\lceil x - 2 - \frac{q-2}{y-q} \rceil, kx], k = 2^{\lfloor \log_2(a_i/x) \rfloor}$.
- For each task with frequency $a_i \in A_y, i \in [1, n]$, the actual distances between jobs in the final schedule can be exact $y * 2^{\lfloor \log_2(a_i/y) \rfloor}$

2) *Reduction*: Based on Corollary V.5, we can design Algorithm 2 to verify the schedulability of UMinMax with double integer pinwheel schedulers. A sample modification for S_{xy} [5] is shown in Algorithm 3 and other schedulers can be modified in the similar manner. Compared to the original schedulers, modification only adds $O(n)$ in time complexity. Since all pinwheel double integers reduction schedulers are pseudo-polynomial, modified schedulers does not increase the O time complexity.

Algorithm 2 Verify whether UMinMax is schedulable with double integer pinwheel schedulers

```

1: // Input: A UMinMax instance  $\mathcal{T} = \{(\min_i, \max_i)\}$ 
2: // Input:  $x, y, p, q$  defined in Corollary V.5
3: // Input:  $\mathcal{T}_x$  task set with respect to  $x$ 
4: // Input:  $\mathcal{T}_y$  task set with respect to  $y$ 
5: // Output: Success or Failure.
6: for all  $\tau_i$  in  $\mathcal{T}$  do
7:   if  $\tau_i \in \mathcal{T}_x$  then
8:     // Check lower separation bound
9:     if  $2^{\lfloor \log_2(a_i/x) \rfloor} * \lceil x - 2 - \frac{q-2}{y-q} \rceil < \min_i$  then
10:       break; return Failure;
11:   end if
12: else
13:   //  $\tau_i \in \mathcal{T}_y$  Check lower separation bound
14:   if  $y * 2^{\lfloor \log_2(a_i/y) \rfloor} < \min_i$  then
15:     break; return Failure;
16:   end if
17: end if
18: end for
19: // Checking finished, it is schedulable
20: Padding  $\lfloor (1 - q/y)x \rfloor - p$  empty tasks with frequency  $x$  to  $\mathcal{T}_x$ 
21: return Success;

```

D. Reducing UMinMax Problem to Three Integers Pinwheel Model

As stated above, we will only list a claim about the lower separation bounds, which can serve as a sufficient condition for schedulability determination.

Corollary V.6: For a pinwheel instance with three distinct numbers $A = \{(a_1, m_1), (a_2, m_2), (a_3, m_3)\}$, if the density is smaller than $5/6$, there exists a pinwheel schedule which satisfies following requirements.

- 1) For tasks with frequency a_1 , the distances between two consecutive jobs are exactly a_1
- 2) For tasks with frequency a_2 , the distances between two consecutive jobs are in the range $(p_1, a_2]$, where $p_1 > 0 \wedge p_1 > a_2 - 6$

Algorithm 3 *UMinMax* pinwheel2 scheduler based on S_{xy}

```

1: //Input: A UMinMax instance  $\mathcal{T} = \{(min_i, max_i)\}$ 
2: //Output: A UMinMax schedule or failure
3: // Create a corresponding pinwheel instance  $A$ 
4:  $A \leftarrow \{max_i | i \in [1, n]\}$ 
5: // Run  $S_{xy}$  to get  $x, y$  and  $\mathcal{T}_x, \mathcal{T}_y$ 
6: if  $\lceil x\rho(\mathcal{T}_x) \rceil / x + \lceil y\rho(\mathcal{T}_y) \rceil / y \leq 1$  then
7:   if Algorithm 2 return success then
8:     ScheduleDouble( $\mathcal{T}_x \cup \mathcal{T}_y$ )
9:   else
10:    return failure
11:   end if
12: else
13:   return failure
14: end if

```

- 3) For tasks with frequency a_3 , the distances between two consecutive jobs are in the range $(p_2, d]$, where $d \leq a_3, p_2 > 0 \wedge p_2 > d - 7$.

Proof: For the first claim, it is proven in [14].

For the second, the proof is as follows:

$$\begin{aligned}
& Place2(i + m_2) - Place2(i) \\
&= \lfloor \frac{(i+m_2)y}{a_1-m_1} \rfloor - \lfloor \frac{ky}{a_1-m_1} \rfloor \\
&\geq \lfloor \frac{(i+m_2)y}{a_1-m_1} \rfloor - \lfloor \frac{ky}{a_1-m_1} \rfloor \quad (\text{See non-equation 13}) \\
&\geq \lfloor \frac{m_2y}{a_1-m_1} \rfloor \quad (\text{See non-equation 12})
\end{aligned}$$

where $y = \frac{m_2}{a_2(1-m_1/a_1)}$, then

$$\begin{aligned}
\lfloor m_2y \rfloor a_1 &= \lfloor a_2(1 - m_1/a_1) \rfloor a_1 \\
&= \lfloor a_2 \frac{a_1-m_1}{a_1} \rfloor a_1 \\
&> a_2(a_1 - m_1) - a_1
\end{aligned}$$

Hence,

$$\begin{aligned}
& \lfloor \frac{m_2y}{a_1-m_1} \rfloor \\
&> \lfloor \frac{a_2(a_1-m_1)-a_1}{a_1-m_1} \rfloor \\
&\geq \lfloor a_2 - \frac{a_1}{a_1-m_1} \rfloor
\end{aligned}$$

Since $m_1/a_1 < 5/6$, $\frac{a_1}{a_1-m_1} < 6$. The second claim is proven.

For the third, [14] only guarantees the distance is no greater than a_3 , in the algorithm, it uses another parameter $d = \lceil \frac{m_3/(1-1/y)}{1-m_1/a_1} \rceil$. If $d > a_3$, the algorithm stops. However, if the density is below $5/6$, d is always no bigger than a_3 . However, we do not know to estimate the range of d with respect to a_3 and we can only get the lower separation bound in the representation of d . First, let $\beta = a_1/(a_1 - m_1)$

$$\begin{aligned}
& Place3(i + m_3) - Place3(i) \\
&= \lfloor \lceil (i + m_3)\alpha \rceil - 1 \rceil \beta - \lfloor \lceil (i)\alpha \rceil - 1 \rceil \beta \\
&\geq \lfloor \lceil (i + m_3)\alpha \rceil - \lceil (i)\alpha \rceil \rceil \beta \\
&\geq \lfloor \lceil m_3\alpha \rceil \rceil \beta \quad (\text{See non-equation 14}) \\
&\geq \lfloor \frac{m_3/(1-1/y)}{(1-m_1/a_1)} \rfloor \\
&\geq \lfloor \frac{m_3/(1-1/y)}{(1-m_1/a_1)} - \frac{1}{(1-m_1/a_1)} \rfloor \\
&\geq \lfloor \frac{m_3/(1-1/y)}{(1-m_1/a_1)} \rfloor - \lceil \frac{1}{(1-m_1/a_1)} \rceil \\
&\geq \lceil \frac{m_3/(1-1/y)}{(1-m_1/a_1)} \rceil - 1 - \lceil \frac{1}{(1-m_1/a_1)} \rceil \\
&> d - 7
\end{aligned}$$

□

E. Reducing MinMax Problem to PinFair Model

Similarly, we first claim the following about the distance range.

TABLE III: *UMinMax* schedulers and their explanations

Scheduler Name	Explanation
<i>UMM_EDF</i> ₁	EDF scheduler based on Theorem IV.2.
<i>UMM_EDF</i> ₂	EDF scheduler based on Theorem IV.3.
<i>UMM_ML</i>	More-Less based scheduler
<i>UMM_PW</i>	Pinwheel scheduler based on Algorithms 1 and 3
<i>UMM_PW</i> ₂	Pinwheel scheduler based on Algorithm 3
<i>UMM_PF</i>	PinFair scheduler based on Algorithm 4

Corollary V.7: Given a pinwheel instance $A = \{(a_1, m_1), \dots, (a_n, m_n)\}$, let $w_i = (a_i + 1)/m_i$. If A can be schedulable by *PinFair*, then for tasks with frequency $a_i, i \in [1, n]$, the distances between two consecutive jobs in the generated *PFair* schedule are in the range $[\lfloor \frac{(m_i-1)a_i}{m_i+1} \rfloor, a_i]$

Proof: The maximum distance is proven in [6]. The lower separation bound is proved as follows.

$$\begin{aligned}
& earliest(x, j + m_i) - latest(x, j) \\
&= \lfloor \frac{j+m_i}{w_i} \rfloor - (\lceil \frac{j+1}{w_i} \rceil - 1) \\
&\geq \lfloor \frac{j+m_i}{w_i} \rfloor - \lfloor \frac{j+1}{w_i} \rfloor \\
&\geq \lfloor \frac{m_i-1}{w_i} \rfloor \\
&\geq \lfloor \frac{(m_i-1)a_i}{m_i+1} \rfloor
\end{aligned}$$

□

The lower separation bound above seems not so encouraging. Because, if $m_i = 1$, the lower separation bound is 0. However, similar to pinwheel double integers reduction, ‘padding’ empty tasks can be applied to satisfy the minimum separation requirements. Since ‘padding’ may increase both the density and density bound of schedulability, it can only be done by searching all the possibilities, which is shown in Algorithm 4 with $O(n^2)$ time complexity.

While Algorithm 4 only searches the minimum m_{min} to satisfy the separation bound, it is also possible to try to search the maximum m_{min} (*greedy padding*) to elevate the minimum separation as large as possible. However, as shown in the simulation section, this results in higher jitter ratio in the final generated schedule, which is not recommended.

VI. SIMULATION

In this section, we compare *UMinMax* schedulers through simulation. We implement most of the solutions described in the above two sections. The names of *UMinMax* schedulers and their explanations are listed in Table III. We will compare them with two important metrics : acceptance ratio and jitter.

A. Acceptance Ratio Comparison

In our simulation experiments, the acceptance ratio of a scheduler is defined as the ratio between the number of task sets schedulable by it and the total number of task sets generated. For each *UMinMax* scheduler, the acceptance ratio is affected by two factors: (1) the utilization/density bound of the scheduler, (2) the tightness of the minimum separation requirement. Intuitively, for any *UMinMax* scheduler, a lower utilization ratio or smaller minimum separation requirement will result in a higher acceptance ratio. To obtain an empirical assessment, the following simulation experiment is performed.

The size of a simulated task set is ten. For each task, the maximum distance is randomly generated from $[1, 100]$. The minimum distance is assigned according to a certain ratio (*min/max* ratio) between 0 and 1, in which 0 means the traditional model without minimum separation constraint. We define the density of a task

Algorithm 4 *UMinMax PinFair* scheduler

```
1: // Input: A UMinMax instance  $\mathcal{T} = \{(min_i, max_i)\}$ 
2: // Output: A UMinMax schedule or failure
3: // First, create a pinwheel instance  $A = \{(a_j, m_j)\}$  for  $\mathcal{T}$ 
4: for all  $\tau_i$  in  $\mathcal{T}$  do
5:   // Search whether there is  $a_j$  in  $A$  where  $a_j = max_i$ 
6:    $flag \leftarrow false$ 
7:   for all  $a_j$  in  $A$  do
8:     if  $a_j == max_i$  then
9:       // Check minimum distance separation requirement
10:      if  $\lfloor \frac{(m_j-1)a_j}{m_j+1} \rfloor < min_i$  then
11:        // 'Padding' empty tasks
12:         $m_j \leftarrow \lceil \frac{1+min_i/max_i}{1-min_i/max_i} \rceil$ 
13:      end if
14:       $flag \leftarrow true$ 
15:    end if
16:  end for
17:  // Add a new record in  $A$ 
18:  if  $flag == false$  then
19:     $a_k \leftarrow max_i, m_k \leftarrow \lceil \frac{1+min_i/max_i}{1-min_i/max_i} \rceil$ 
20:     $A \leftarrow A \cup (a_k, m_k)$ 
21:  end if
22: end for
23: // Second, search minimum  $m_j$  to satisfy  $\frac{m_j}{m_j+1} \geq \rho(A)$ 
24:  $m_{min} \leftarrow \text{minimum } m_j$ 
25: while  $\frac{m_{min}}{m_{min}+1} < \rho(A)$  and  $\rho(A) < 1$  do
26:    $m_{min} \leftarrow m_{min} + 1$ 
27:   update all  $m_{min}$ s in  $A$ 
28: end while
29: if  $\frac{m_{min}}{m_{min}+1} \geq \rho(A)$  then
30:   // Success
31:   Schedule  $A$  with PinFair
32: else
33:   return failure;
34: end if
```

set as the sum of the reciprocals of maximum distances. Several tests are carried out with different average densities. The results are shown in Figure 6, which confirm that lower utilization ratio or smaller minimum separation requirement will result in higher acceptance ratio. Also, we can see the performance differences between different solutions. When the density is high, only *UMM_EDF₂* and *UMM_PW* can work. The irregularities of performance in response to the increase of min/max ratio result from the non-linear requirements of both schedulers. For *UMM_EDF₂*, satisfying equation 6 is quite case by case when the density is high. This is similar for *UMM_PW* on cases with densities higher than its density bound. For *UMM_PF*, it only works well when the density is very low and it is better than *UMM_ML*, whose performance is still moderate with low density.

B. Jitter Comparison

A goal of introducing *MinMax* is obtaining jitter guarantees. In this subsection we compare the jitter performance between different *UMinMax* schedulers. To do this, we define jitter ratio

$jitter_ratio_i$ for each task i in a schedule S as follows.

$$jitter_ratio_i = \frac{jitter_i}{Average_Distance_i} \quad (16)$$

where $jitter_i$ denotes the jitter of task i defined in Definition III.2 and $Average_Distance_i$ denotes the average distance between two consecutive jobs of task i in the schedule.

Figure 7 shows how the minimum separation constraint affects the jitter ratio for *UMM_EDF₂* for different task density ratios. With minimum separation close to the maximum distance, the jitter ratio goes down gradually. However, if the density ratio is low, the minimum separation is not affected as much as the case with higher density ratios.

Figure 9 shows the jitter performance of different schedulers. Unsurprisingly, *UMM_PW* is the best and the jitter ratio is close to zero. Since *More-Less* tries to maximize the period P_i , it has smaller D_i s than EDF schedulers and thus the jitter ratio is lower. The last is the *PinFair* scheduler. While EDF schedulers tries to minimize the distance between job execution time and periodic job release time, *PinFair* only considers the average time fairness and thus it has higher jitter ratio. Compared to Figure 6, performance of each scheduler is relative more stable.

C. Differences with Padding and without Padding

As stated in the previous section, 'padding' with empty tasks is employed in two solutions: Algorithm 3 and Algorithm 4. In this section, we shall show the effects of 'padding'. Different from above simulations, we specially implement *UMM_PW₂* based on Algorithm 3. For *UMM_PW₂*, 'padding' improves acceptance ratio and minimizes the jitter ratio. For *UMM_PF*, 'padding' only improves the acceptance ratio but keeps the same jitter ratio.

The acceptance ratio results are shown in Figure 10. While the density ratio is high, there is no space for padding and hence the acceptance ratios are the same. However, if there is bandwidth left, 'padding' can greatly improve the acceptance ratio.

Figure 8 shows the jitter ratio difference with 'padding' and without 'padding'. With padding, jitter ratio of *UMM_PW* drops to near zero. For *UMinMax_PF*, 'padding' is only necessary when the task set is not schedulable; when the task set is schedulable, minimum 'padding' is no 'padding'. Hence, the two have the same jitter ratio. However, with *greedy padding*, the jitter ratio is higher and thus it is not recommended.

VII. APPLICATION OF *UMinMax* TO PROCESS CONTROL

In this section, we apply *UMinMax* to fieldbus scheduling, which is widely used in process control systems. We first describe an example of process control system and then show the advantages of applying *UMinMax* model.

A. Example Fieldbus Process Control System

In this experiment, an example fieldbus based system like Figure 1 is simulated. There are n control loops attached to the fieldbus. Each loop is associated with three entities: a sensor for sensing, an actuator for controlling and a PID (Proportional, Integral, Derivative) controller. For each loop, a sensor should get present process data and send it to the corresponding PID controller through the fieldbus. The PID controller then generates an output periodically according to the latest sensor data and sends

Fig. 6: Acceptance ratio comparison between *UMinMax* schedulers

Fig. 7: Effects of min/max ratio on jitter ratio of *UMM_EDF2*

Fig. 8: Effects of 'padding' on jitter ratio

Fig. 9: Jitter ratio comparison between *UMinMax* schedulers

Fig. 10: Effects of 'padding' on acceptance ratio

it to related actuator. And, finally, actuator will adjust the process. Such loop will continue to keep the process data at a desired value (set point). There is also a fieldbus scheduler that schedules all the communications between sensors and PID controllers. The communication protocol is based on TDMA (Time Division Multiple Access), each time slot of fixed size can only allow one pair of sensor and PID controller talking and the duration of time slot is long enough to transmit a data unit. Since a PID controller should send data to the actuator periodically, the corresponding sensor should keep the PID controller updated with the latest process data. Hence, there is a maximum distance constraint for each sensor. However, it is also a waste for a sensor to send out data too frequently, especially when the communication is wireless.

For simplicity, a linear process control algorithm is simulated. The process function ($P(t)$) and PID gain (u) are listed in Equation 17 and Equation 18. In Equation 17, t_0 is the latest time slot when the PID sends out its gain to the corresponding actuator. λ and $init_v$ are process parameters, which will be set beforehand. By experience, λ is between $[0,1]$ and the period

Fig. 11: Relationship between jitter ratio and IAE under the same utilization ratio

of PID controller should not exceed the reciprocal of λ . In Equation 18, K_p, K_i, K_d are PID parameters. In our simulation, K_i and K_d are set to 0 for simplicity. The metric of performance is Integral Absolute Error (IAE) as expressed in Equation 19.

It is easy to see that IAE can be affected by irregular latencies between sensing and controlling. In other words, the smaller the jitter, the better the result. This is shown in Figure 11. Under

TABLE IV: IAE_U and U_ratio with different density ratios

(a) Density Ratio = 0.20				
Schedulers	Min/Max Ratio			
	0.2		0.4	
	IAE_U	U_ratio	IAE_U	U_ratio
<i>Half-Half EDF</i>	596.1	0.41	671.0	0.42
<i>UMM_EDF₂</i>	454.8	0.35	489.4	0.30
<i>UMM_PW</i>	422.5	0.26	479.5	0.26

(b) Density Ratio = 0.30				
Schedulers	Min/Max Ratio			
	0.2		0.4	
	IAE_U	U_ratio	IAE_U	U_ratio
<i>Half-Half EDF</i>	937.9	0.63	647.2	0.62
<i>UMM_EDF₂</i>	756.4	0.53	470.4	0.45
<i>UMM_PW</i>	639.2	0.40	485.3	0.41

(c) Density Ratio = 0.40				
Schedulers	Min/Max Ratio			
	0.2		0.4	
	IAE_U	U_ratio	IAE_U	U_ratio
<i>Half-Half EDF</i>	956.5	0.87	809.7	0.85
<i>UMM_EDF₂</i>	744.7	0.71	604.3	0.61
<i>UMM_PW</i>	600.1	0.52	540.7	0.53

the same utilization ratio, IAE is heavily affected by jitter. It is also possible to minimize the jitter by increasing sampling rate but this involves higher utilization ratio of fieldbus bandwidth. Hence, in order to show the performance of different schedulers, we use the multiple of IAE and utilization ratio ($IAE_U = IAE * utilization_ratio$) as the metrics. It is easy to see that the lower the IAE_U , the better performance.

$$P(t) = init_v + (u + setpoint - init_v) * \lambda * (t - t_0) \quad (17)$$

$$u = K_p * e(t) + K_i * \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \quad (18)$$

$$IAE = \int_0^t |e(t)|dt \quad (19)$$

B. Results Comparison

We show the comparison between *UMinMax* solutions and a solution used in a traditional control system, in which the period of a sensor is set to be the half of the corresponding PID controller's, which can be named as *Half-Half* principle [13]. For ease of expression, we define the sum of the reciprocals of periods of PID controllers as the density ratio of the system. It is easy to see the density ratio cannot be higher than 0.5 for *Half-Half* principle.

We compare three schedulers: *Half-Half EDF*, *UMM_EDF₂* and *UMM_PW*. For *Half-Half EDF*, there is no minimum separation constraint and deadlines are the same as periods. For *UMinMax* schedulers (*UMM_EDF₂* and *UMM_PW*), different *min/max* ratios are tried out. The results of IAE_U s and utilization ratios (U_ratio) of schedules are listed in Table IV. It can be seen from the table that both *UMM_EDF₂* and *UMM_PW* outperform *Half-Half EDF* in every test. The main reason is that *Half-Half EDF* results in much higher utilization ratio while it does not gain much on IAE values. For *UMM_PW*, the utilization ratio is only about 60% of *Half-Half EDF*, which could greatly save energy in wireless-based process control systems.

VIII. DISCUSSION AND CONCLUSION

In this paper we examine the issue of controlling the sampling intervals of sensors in industrial process control systems such as

TABLE V: Time complexity comparison of different reductions

Reduction	EDF	More-Less	Pinwheel1	Pinwheel2	PinFair
Complexity	$O(n)$	$O(n)$	$O(min_max * n)$	$O(n)$	$O(n^2)$

those based on Fieldbus. In contrast to traditional approaches, we do not require the sampling intervals to be strictly periodic and we study a more relevant scheduling model with both maximum distance and minimum separation constraints. This model more closely reflects the need from the real-world industrial automation applications. We look at its basic properties and reduce the associated scheduling problem to traditional task models at the expense of some loss of schedulability, which is measured by an acceptance ratio in our experimental evaluation. The advantage of our approach from a practical point of view is that we can reuse the well-established and rich scheduling methods and also improve the scheduling results such as jitter guarantees. Most of the reductions are efficient (Table V). We compare these solutions by simulations to evaluate the acceptance ratio and jitter ratio. Finally, we apply *UMinMax* to simulate a typical process control system and the results show that our *UMinMax* schedulers can outperform traditional schedulers by reducing utilization ratio with little increase in control error. In future work, we shall validate our simulation results in connection with the deployment of our *MinMax* schedulers on industrial fieldbus controllers in collaboration with our industry partners.

REFERENCES

- [1] C.-C. Han and K.-J. Lin, "Scheduling distance-constrained real-time tasks," in *Real-Time Systems Symposium*, 1992, Dec. 1992, pp. 300–308.
- [2] C.-C. Han, K.-J. Lin, J.W.S.Liu, and S. Liu, "Scheduling jobs with temporal distance constraints," *SIAM J. Comput.*, vol. 24, pp. 1104–1121, October 1995.
- [3] R. Holte, A. K. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The pinwheel: a real-time scheduling problem," in *System Sciences, 1989. Vol.II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on*, vol. 2, Jan. 1989, pp. 693–702 vol.2.
- [4] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel, "Pinwheel scheduling with two distinct numbers," *Theor. Comput. Sci.*, vol. 100, pp. 105–135, June 1992.
- [5] M. Chan and F. Chin, "General schedulers for the pinwheel problem based on double-integer reduction," *Computers, IEEE Transactions on*, vol. 41, no. 6, pp. 755–768, Jun. 1992.
- [6] S. K. Baruah and S.-S. Lin, "Pfair scheduling of generalized pinwheel task systems," *Computers, IEEE Transactions on*, vol. 47, no. 7, pp. 812–816, Jul. 1998.
- [7] L. Dong, R. Melhem, and D. Mossé, "Scheduling algorithms for dynamic message streams with distance constraints in tdma protocol," in *Proceedings of EuroMicro Conference on Real-Time Systems*, 2000.
- [8] C.-C. Han and K.-J. Lin, "Scheduling real-time computations with separation constraints," *Inf. Process. Lett.*, vol. 42, pp. 61–66, May 1992.
- [9] S. K. Baruah, D. Chen, and A. K. Mok, "Jitter concerns in periodic task systems," 1997.
- [10] S. K. Baruah, R. R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.
- [11] J.W.S.Liu, *Real-time Systems*. Prentice-Hall, NJ, 2000.
- [12] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized elastic scheduling," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 236–245.
- [13] M. Xiong and K. Ramamritham, "Deriving deadlines and periods for real-time update transactions," in *IEEE Transactions on Computers*. IEEE Computer Society Press, 1999, pp. 32–43.
- [14] S.-S. Lin and K.-J. Lin, "Pinwheel scheduling with three distinct numbers," in *Real-Time Systems, 1994. Proceedings., Sixth Euromicro Workshop on*, Jun. 1994, pp. 174–179.

- [15] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600–625, 1994.