5-2018

Data Freshness Over-Engineering: Formulation and Results

Dagaen Golomb
University of Pennsylvania, dgolomb@seas.upenn.edu

Deepak Gangadharan
University of Pennsylvania, deepakg@seas.upenn.edu

Sanjian Chen
University of Pennsylvania, sanjian@seas.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

IEEE ISORC 2018, Singapore, May 29 - 31, 2018

Data Freshness Over-Engineering: Formulation and Results

Abstract
In many application scenarios, data consumed by real-time tasks are required to meet a maximum age, or freshness, guarantee. In this paper, we consider the end-to-end freshness constraint of data that is passed along a chain of tasks in a uniprocessor setting. We do so with few assumptions regarding the scheduling algorithm used. We present a method for selecting the periods of tasks in chains of length two and three such that the end-to-end freshness requirement is satisfied, and then extend our method to arbitrary chains. We perform evaluations of both methods using parameters from an embedded benchmark suite (E3S) and several schedulers to support our result.

Keywords
real-time systems, data freshness, schedulability

Data Freshness Over-Engineering: Formulation and Results

Dagaen Golomb, Deepak Gangadharan, Sanjian Chen, Oleg Sokolsky, and Insup Lee
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA, USA
{dgolomb,deepakg,sanjian}@seas.upenn.edu,{sokolsky,lee}@cis.upenn.edu

***Abstract*—In many application scenarios, data consumed by real-time tasks are required to meet a maximum age, or freshness, guarantee. In this paper, we consider the end-to-end freshness constraint of data that is passed along a chain of tasks in a uniprocessor setting. We do so with few assumptions regarding the scheduling algorithm used. We present a method for selecting the periods of tasks in chains of length two and three such that the end-to-end freshness requirement is satisfied, and then extend our method to arbitrary chains. We perform evaluations of both methods using parameters from an embedded benchmark suite (E3S) and several schedulers to support our result.**

***Index Terms*—real-time systems; data freshness; schedulability;**

I. Introduction & Motivation

In real-time systems it is common for applications to consist of tasks which consume input data in order to perform a particular function. Typically this input is either sensor data that is crucial for determining the state of the system or to be used as input to other tasks. In these types of systems, there is usually an explicit or implicit timeliness requirement for this data. The relevance of a computation can be intuitively evaluated by the age, or "freshness," of the data that was used during the computation.

Traditionally, task sets have been over-engineered to provide fresh data. Assuming a task must perform some computation on its input data every one second, the obvious question is what should be the rate at which the task producing the required data execute? In these situations, engineers may err on the side of caution and choose to produce the input data faster than necessary, which then needs to be tested to check for freshness in the worst-case scheduling scenario. For example, a task that reads a sensor value and forwards it may be dispatched fifty times a second although the consuming task only needs data younger than one tenth of a second for safety. While safe, this reduces the efficiency and schedulability of the system, and there is no analytical framework to ensure the freshness constraint. This leads to more dependence on testing and several cycles of parameter tuning to ensure the safety, while ultimately providing no guarantee of optimality, as noted by others in the field [1], [2], [3], [4]. This paper outlines an analytical formulation of this over-engineering strategy for 2 and 3 task chains, and then uses insight from these results to build an optimization problem for arbitrary task chains. Our solution provides a strategy for choosing the period of producer tasks with nearly no information about the scheduling algorithm used.

Real-time systems are ones where tasks must be executed in a timely manner for correct results. For real-time systems that require managing and passing data, it is understood that this data may need some notion of timeliness as well. The most intuitive way of time stamping data is to consider when it was created or obtained, e.g. when a sensor polling task reads it.

In broad terms, the goal we wish to achieve is for the input of a given task to meet predetermined freshness guarantees, where freshness represents the age of the data. An example of this would be a task B which needs to use speed sensor data produced by a task A that is at most 100 milliseconds old.

The notion of freshness is not a new concept. A variety of domains consider this notion, with the definition tailored to each domain, all of which agree that the quality of data is not merely a function of its "correctness" or accuracy. Common notions use terms such as currency [5], which describes how much time has passed since data collection, and timeliness [6], which measures how old the data is when collected. We will consider freshness to be the time elapsed since the data was produced, particularly when it is output by a producer task. In particular, we are going to examine the following scenario: given a task Z with fixed period, which consumes data that makes its way through a task chain $A, B, C, \ldots$, choose periods for $A, B, C, \ldots$ so as to ensure a freshness bound, the desired maximum age of consumed data, is always enforced. That is to say, when task Z runs, the age of the data created by task A which was used to eventually produce Z's input is less than the given freshness bound.

The solution to this problem is not always unique. Given a set of solutions, we wish to rank them against some metric to select the one that best fits our needs. There are several metrics one could use; in this work we focus on minimizing task set utilization. More precisely, we will attempt to minimize maximum system utilization. We chose this metric because it is a common metric of schedulability and efficiency in the real-time systems domain. Intuitively, a low task set utilization provides the necessary performance at the lowest computational cost, allowing for more tasks to be introduced to the system and increasing the likelihood of schedulability. We will attempt to solve this problem with minimal assumptions about the system and scheduling policy. This could be useful

for task sets which may run on many hardware and software combinations.

II. Formalization

A. Task Definitions

Our model assumes a periodic task set on a unicore system. For periodic task sets, each task A is characterized by a period P_A, a relative deadline D_A, and a worst-case execution time (WCET) E_A^u. We additionally include a best-case execution time (BCET) E_A^l.

A job is an instance of a task, i.e. one of the actual executions of a task. Each task produces multiple jobs and in the periodic model, a job is released every P_A time. Each i^{th} job of task A has a release time r_A^i and a finish time f_A^i. Note that $f_A^i - r_A^i$ is lower bounded by BCET (when the task is executed immediately and runs to completion) but not upper bounded by the WCET if the job can be preempted. However, assuming the system is schedulable, the job is completed before its deadline. If this is not the case then the job has experienced a deadline miss. In this work we will consider deadlines to be equal to periods for simplicity and clarity, but our model could be extended to account for other deadlines.

Since we wish to uphold a data freshness guarantee, we define $d_{A\to B}$ to be the desired upper bound on data freshness for data produced by task A and consumed by task B. Lastly, we define a value that will help us formulate the requirements of our system. Let $C_A(r_B^i)$ denote the most recently completed job of task A before the release of job i of task B. This will be useful since we assume the most recently produced output from a task is the freshest. This assumption holds because of our deadline = period assumption since in this case multiple jobs of a task in the system implies a deadline miss and hence unschedulability. However, this is easily enforced when the period = deadline assumption is removed by only allowing jobs to output data if they are overwriting a value which was last updated by an older (earlier release time) job.

B. Communication Definitions

We include communication, storage, or other latencies into our formulation. For our purposes, data latency denotes any delay in storing or transferring a value to another task. Examples include the delay to store a value in a database, inter-processor communication, and network delays. For our formulation, we require the following values to be provided: Minimum Data Delay DL^{min} and Maximum Data Delay DL^{max}.

Since our formulation will consider data to be available at task completion, we can abstract the communication delay into our task execution times as follows:

$$E_A^{l,min} = E_A^l + DL^{min} \quad \text{and} \quad E_A^{u,max} = E_A^u + DL^{max}$$

C. Assumptions

Since we do not know the nature of the tasks, particularly when they produce and consume data, we choose to abstract the specifics of data production and consumption. We assume tasks consume data at the very beginning of their execution as this results in shortest freshness deadline. Since execution could happen immediately at release time, without knowledge of the scheduler, we assume jobs read input values at the time of their release.

On the other hand, we assume data is produced at finish time. Therefore, we must wait until the completion of a job before we can consider its data available, at which time it overwrites the data from the last completed job of the task. While we do not consider delay from the polling task, these tasks are often small and we consider this trivial. If the initial task has non-trivial execution time its WCET can be subtracted from the desired freshness bound. While producing at the end of a task does not produce the worst staleness, it is easy to hold off production until the end of task in order to make this hold. Additionally, similar to how we can account for WCET in the producer task, one can subtract the difference between the end of a job and the earliest it could produce data from the freshness constraint.

Since the freshness of the final, fully-transformed data is often of interest, we assume that the period of the final consuming task is fixed, i.e., if a designer wants the transformed data to be at most 50 milliseconds old, it is intuitive that they would select this as the period of the final task. We will assume the designer selects an appropriate period for this task and would like to compute the input tasks' periods to meet their freshness requirement. In this work we present our approach considering unicore systems for simplicity. However, our formulation easily scales to multi-processor architectures as our single-core assumption provides stricter resource constraints and higher preemption. A more detailed discussion of the extension to multi-processor systems is found in a later section.

D. Requirements

Using the notation and assumptions above, for a producer (A) and consumer (B) pair we define the freshness requirement of data produced by task A and consumed by job i of task B as follows:

$$\forall i, r_B^i - f_A^{C_A(r_B^i)} \leq d_{A\to B}$$

That is, at the release of any job i of task B, the time since the completion of the last job of task A is less than or equal to the freshness bound $d_{A\to B}$.

E. Problem Statement

The problem statement is as follows. Let $E_k^{*,*}$ denote that we have both worst-case and best-case execution times including communication and other latencies for task k, and let $U(T)$ denote the system utilization. We have n tasks in the chain.

$$
\begin{array}{ll}
\text{Given} & \forall k, E_k^{*,*};\ \text{and}\ d_{1\to n} \\
\text{Find} & \forall k < n, P_k \\
\text{Minimizing} & U(T) = \sum_n \frac{E_n^{u,max}}{P_n} \\
\text{Subject To} & \forall i, \sum_{j=2}^{n}(r_j^i - f_{j-1}^{C_j(r_{j-1}^i)}) + \sum_{j=2}^{n-1} E_n^{u,max} \leq d_{1\to n}
\end{array}
$$

In words, we minimize utilization while keeping the sum of task WCETs between the original producer and the final consumer plus the sum of any time spent between the end of a job of task j and the release of a job of the next task $j+1$, to be less than our freshness bound. This quantity is the total time the data spends in flight. We use WCETs since this case produces the most strict local deadlines, which we describe later, for each pair of tasks.

We do not consider schedulability in the formulation as to maintain maximum generality. Therefore, our solution will not guarantee schedulability, which must be confirmed using scheduler-specific methods afterwards. However, our solution relies on an assumption of the schedulability of the final task set, since we assume a job of each task runs to completion during each task period, and is thereby invalidated if this is not the case.

It is clear this minimization has a solution. Assume all periods are low enough to ensure the freshness bound (all close to 0 for example, as it is clear all periods arbitrarily close to 0 would cause the bound to hold). We can lower utilization by increasing any P_k, but at some point increasing P_k will violate our freshness bound. For example, if $P_k = 2 \cdot d_{1\to n}$ freshness will not hold, while setting P_k arbitrarily closer to 0 will cause the bound to hold (disregarding schedulability). At some point between these two the system switches between upholding and violating the freshness bound. This point is one solution.

III. Two Task Result

First we consider the simplest case where there are only two tasks: one producer (task A) and one consumer (task B). Considering just task A we see the worst case depicted in Figure 1. This scenario assumes schedulability to ensure that one job of A runs to completion in each period. This is where our schedulability assumption becomes vital for our solution.

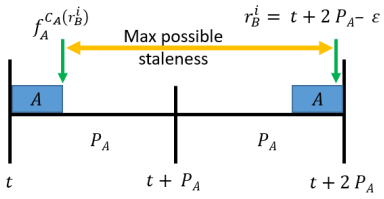

Fig. 1. Maximum Staleness Scenario for Data from Task A.

Figure 1 labels the maximum separation between two publishes of output data from the depicted task. To maximize staleness, we assume a job of task B is released arbitrarily close to the finish of the second job of task A and that the first (BCET) job of task A ran to completion at the beginning of the previous period.

We assume deadlines are equal to periods. However, it is easy to see that a shorter deadline moves the latest possible execution of the second job of task A earlier, and hence decreases the max staleness. We will not consider this case further but it will be clear later how this could be substituted in place of our period = deadline example.

Theorem 1. *The scenario in Figure 1 is the upper bound scenario for data freshness for data produced by a task.*

Proof. Note that one job must be executed within each period as per the definition of periodic tasks and our schedulability assumption. Consider any placement of two jobs of a task within two consecutive periods. Assume this instance is not the one depicted in Figure 1. Then one of the following applies:

Case 1. *The job in the first period is not completed as soon as possible. In this case, move the start of this job execution ϵ earlier. This increases the staleness by ϵ.*

Case 2. *The job in the second period is not completed as late as possible. In this case, move the start of this job execution ϵ later. This increases the staleness by ϵ.*

Since all other instantiations of the problem can be moved closer to the depicted instance while strictly increasing data staleness, it follows that the depicted instance is the unique worst case for output data staleness. □

Now that we have proven the above scenario is the worst case freshness of data from task A, we can analytically solve for the constraint on P_A. From Figure 1, we can see that the maximum staleness is composed of two periods of A less one execution of A less ϵ. We want this less than our freshness bound $d_{A\to B}$. We can then solve for P_A to prove the following lemma. We assume the BCET and data delay for the first job in order to maximize staleness. The execution time of the second job is irrelevant.

Lemma 1. *To ensure the output of task A is always at most $d_{A\to B}$ old, choose $P_A \leq \frac{d_{A\to B} + E_A^{l,min}}{2}$.*

Proof.

$$
\begin{aligned}
d_{A\to B} &\geq 2P_A - E_A^{l,min} - \epsilon && \text{From Figure 1} \\
P_A &\leq \frac{d_{A\to B} + E_A^{l,min}}{2} && \text{Rearrange; } \epsilon \to 0
\end{aligned}
$$

□

Thus if we set P_A equal to the above quantity we ensure that the output of task A is always at most $d_{A\to B}$ old, and therefore can never be older when consumed by task B. We can also set P_A less than this quantity. Recall that our fitness metric is total utilization. It is trivial to see that choosing P_A as

large as possible will minimize the task set utilization. Thus the solution for the two task scenario while minimizing utilization is to set P_A equal to the quantity in the lemma.

IV. Three Task Result

We now extend the above idea to three tasks. In this scenario, let our tasks be denoted as A, B, and C. Task A produces output consumed by task B, which in turn produces output consumed by task C. We want to limit the age of the output of A that is eventually used by C.

For this scenario, define $d_{A\to C}$ as the maximum age of the data produced by task A that is used by task C. Note that we are not making any assumptions about when a job of task B is executed between jobs of tasks A and C. The formalization is similar to the two task scenario:

$$\begin{aligned}
&\text{Given:} && E_A^{*,*}, E_B^{*,*}, E_C^{*,*}, P_C \text{ and } d_{A\to C}\\
&\text{Find:} && P_A \text{ and } P_B\\
&\text{That Minimizes:} && U(T) = \sum_i \frac{E_i^{u,max}}{P_i}\\
&\text{Subject To:}
\end{aligned}$$

$$\forall i, r_C^i - f_B^{C_B(r_C^i)} + E_B^{u,max} + r_B^{C_B(r_C^i)} - f_A^{C_A(r_B^{C_B(r_C^i)})} \leq d_{A\to C}$$

Note that we assume worst case execution and data delay for task B. This will produce a more urgent local deadline that will enforce freshness also when this does not occur. More rigorously, let $\delta = E_B^{u,max} - E_B^{l,min}$. Using the max value as we do instead of the least value decreases $d_{B\to C}$ by δ. As can be seen from Lemma 1 this decreases P_B by $\delta/2$. Viewing task B and C alone as their pair, it is easier to see that the freshness is made of two periods of B for a total decrease of δ. Hence, in the case of minimal execution of task B the execution increases by δ but this same amount has already been compensated for in the formulation.

The execution of the three tasks is outlined in the figure 2.

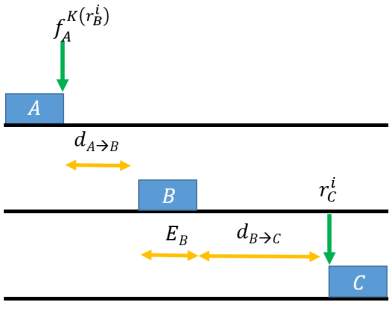

Fig. 2. Maximum Staleness Scenario for Data from Task A.

Figure 2 labels the several intervals in the execution of the three tasks. Note how between tasks A and B and between tasks B and C we've added local freshness constraints. By meeting these two freshness constraints, the freshness of data from task A to task C is ensured. These added constraints will not be present in our final solution but are added here to allow us to introduce Lemma 1 while considering task B. These local constraints are used as free variables in our optimization, and once determined we will use our lemma to calculate period assignments that rely only on the end-to-end constraint and task execution times.

Note that there could be multiple jobs of B between A and C but that this does not change the age of the data as it travels from A to C. For any job of task B between A and C the sum of the values depicted in Figure 2 remain unchanged. Anyways, note that once we optimize for utilization it is unlikely that multiple jobs of B will be executed since this implies a shorter period of task B and hence higher utilization.

From Figure 2 we can see that the maximum age of the data from task A used by task C, $d_{A\to C}$, is the sum of three values. Concretely,

$$d_{A\to C} = d_{A\to B} + E_B^{u,max} + d_{B\to C}$$

We now use our lemma to extend the two task result to this scenario. Using Lemma 1…

$$P_A \leq \frac{d_{A\to B} + E_A^{l,min}}{2} \text{ and } P_B \leq \frac{d_{B\to C} + E_B^{l,min}}{2}$$

Our lemma uses the BCET for the first task in each pair in order to maximize the possible staleness. For our freshness guarantee, we will use the BCET to be pessimistic with the data age, whereas when calculating utilization we will use the WCET. That is, our solution will ensure freshness even with best-case execution of the first task, while minimizing the maximum utilization the system could experience.

We simplify the objective by removing constants from the utilization expression and then transform it by substituting from Lemma 1, using equality since lower periods increase utilization:

$$\begin{aligned}
U(T) &= \left(\frac{E_A^{u,max}}{P_A} + \frac{E_B^{u,max}}{P_B}\right) && \text{Objective}\\
&= \frac{2E_A^{u,max}}{d_{A\to B} + E_A^{l,min}} + \frac{2E_B^{u,max}}{d_{B\to C} + E_B^{l,min}} && \text{Substitution}
\end{aligned}$$

We minimize this objective to arrive at the following solution.

Theorem 2. *Given $E_A^{*,*}$, $E_B^{*,*}$, $E_C^{*,*}$, and P_C, to minimize utilization while enforcing the freshness bound $d_{A\to C}$, choose*

$$P_A = \frac{\sqrt{\frac{E_A^{u,max}}{E_B^{u,max}}}(E_B^{l,min} + d_{A\to C} - E_B^{u,max} + E_A^{l,min})}{2(1 + \sqrt{\frac{E_A^{u,max}}{E_B^{u,max}}})}$$

$$P_B = \frac{\sqrt{\frac{E_A^{u,max}}{E_B^{u,max}}}(E_A^{l,min} + E_B^{l,min}) + d_{A\to C} - E_B^{u,max} + 2E_B^{l,min}}{2(1 + \sqrt{\frac{E_A^{u,max}}{E_B^{u,max}}})}$$

Proof. Due to space constraint, we provide the detailed proof in [7]. □

As intended, our solution is expressed solely in terms of the task parameters and the end-to-end freshness requirement.

The formulation may assign non-integer periods which we consider acceptable. The designer could floor such values to the nearest platform-compatible value while preserving the freshness guarantee. If the designer is concerned about a large number of tasks in the system, several tasks with similar periods could be combined into one task with the lowest period of the component tasks. However, both of these modifications will increase utilization.

Our solution may not be schedulable. In the case it is not, there may be a schedulable parameter set that guarantees the desired freshness, depending on the system's scheduling policies. Finding the optimal parameters in such a case may be non-trivial.

It may be possible to extend this optimization using calculus minimization problems for a given number of tasks. The primary challenge would be solving increasingly difficult optimizations. It may be possible to generalize this method to n tasks in this manner. However, moving forward we will express the formulation as a general optimization problem to be solved with a software solver.

V. N-Task Extension

In this section we extend our formulation to task chains of arbitrary length and with forks and merges. We do this through the use of our 2-task and 3-task formulation insights, as constraints in an optimization problem which can then be solved using constrained optimization software.

We again formulate any end-to-end deadlines as the combination of several local constraints. In particular, there is a local freshness constraint from each producer task i to its consumer task j. Let each pair $\{i,j\} \in E$ and E_k be the subset of pairs in a chain for a particular freshness constraint D_k. Each of these local freshness constraints are between two tasks, and thus we can ensure the consuming task receives data according to the local constraint by using Lemma 1: $P_i \leq \frac{d_{i\to j}+E_i^{l,min}}{2}$, with this inequality being equal in order to minimize utilization.

As we saw in the three task scenario, it is sufficient to solve for local deadlines as they can be transformed into periods of the tasks. Our optimization follows suit by using this idea in the constraints and then solving for the local deadlines which yield the least utilization. Once a solution is found we convert the local constraints into periods for producers. A concrete example to illustrate how we will encode our freshness constraints, and hence period selection, is shown in Figure 3.

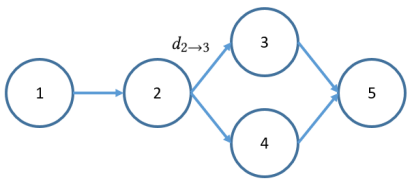

Fig. 3. Example of chain with fork and merge.

Suppose data consumed by task 2 must be at most 10 time units old and data consumed by task 5 must be at most 50 time units old. Our optimization would look like the following:

$$\begin{aligned}
\text{Given: } & E_i^{*,*} \text{ for each } i \text{ in } 1\dots 5 \\
\text{Find: } & P_1,\dots,P_5 (\text{ i.e. } d_{1\to 2}...d_{4\to 5}) \\
\text{Minimize: } & \text{Maximum Utilization} \\
\text{Subject To: } & d_{1\to 2} \leq 10 \\
\text{and: } & d_{1\to 2} + E_2^{u,max} + d_{2\to 3} + E_3^{u,max} + d_{3\to 5} \leq 50 \\
\text{and: } & d_{1\to 2} + E_2^{u,max} + d_{2\to 4} + E_4^{u,max} + d_{4\to 5} \leq 50 \\
\text{and: } & \text{each } d_{\star\to\star} > 0
\end{aligned}$$

From here forward $\star$ is a wild card for any valid value that can be used in its place.

Note that this can be easily applied to several pieces of data and is not limited to a single datum. Constraints can be added for several pieces of data and the solution will select the most stringent period required for any one particular data piece.

Below we present the optimization problem for any chain and deadlines. Since we do not know the periods of the tasks during the optimization, we must represent utilization using the provided parameters. We do this using the local deadlines. The optimization problem is as follows:

$$\begin{aligned}
\text{Given: } & E_i^{l,min}, E_i^{u,max} \text{ for each } i \text{ in } 1\dots n \\
\text{Find: } & d_{i\to j} \text{ for each } \{i,j\} \in E \\
\text{Minimize: } & U = \sum_{i:\{i,\star\}\in E} \frac{E_i^{u,max}}{P_i} \\
& = \sum_{i:\{i,\star\}\in E} \frac{2 \times E_i^{u,max}}{\min(d_{i\to\star}) + E_i^{l,min}} \\
\text{Subject To: } & \forall k, \sum_{i:\{i,\star\}\in E_k} E_i^{u,max} + \sum_{i:\{i,j\}\in E_k} d_{i\to j} \leq D_k \\
\text{and: } & \forall \{i,j\} \in E, d_{i\to j} > 0
\end{aligned}$$

Note the problem is not linear due to the utilization objective, but the utilization monotonically increases as any local deadline decreases. As each variable changes the objective in the same manner (no saddles) and the constraints are simple addition and comparisons, this problem appears convex. While we cannot provide an analytical solution for the n, it performs the same optimization as the analytical case would for any n.

Given our intuition from the theory evaluation (later) which suggests earlier tasks in the chain are often assigned higher priority, we extended the optimization problem to be specific to rate-monotonic scheduling by adding additional constraints dictating that the period of each producing task in a chain is less than or equal to the period of its consuming task. Note that this does not always hold in the 3 task scenario depending upon the parameters, but it is a trend we noticed and was a basis for our intuition. The added constraints will cause earlier tasks to have higher priority with the intention of pushing new values through the chain before consuming older ones. The

RM-specific formulation is the same as above but with the added constraint:

$$\begin{aligned} &\forall\{i,j\} \in E, P_i \leq P_j \\ \implies &\forall\{i,j\} \in E, \forall k : \{j,k\} \in E, \\ &\frac{d_{i\to j} + E_i^{l,min}}{2} \leq \frac{d_{j\to k} + E_j^{l,min}}{2} \end{aligned}$$

Other constraints may be added by the designer if desired. For example, in either formulation, if a polling task T_1 must run at least every 50 time units as per the Nyquist requirement, an additional constraint could be added: $P_1 \leq 50$. Note that in this case, P_1 would have to converted to a quantity which uses local deadlines.

Finally, note that this method works for multiple disconnected task chains. Each chain can be encoded while the objective function remains the utilization of all tasks. An entire system of task chains with forks, merges, etc. can be simultaneously optimized in a single optimization problem.

VI. Preemption and Other Job Delays

While Figure 2 depicts task B running without preemption, here we will explain why preemption (or other task delays during execution) does not invalidate our theory.

Theorem 3. *Preemption and other job delays do not cause data freshness violations.*

Proof. Assume this is not the case, i.e. that job preemption and other delays cause a freshness miss. Let all preemption and other job delays be denoted by p.

Consider any producer (A) and consumer (B) task pair. Note the maximum wait scenario in Figure 1 holds regardless of the execution time of the second job of task A and regardless of the release time of task B. Thus our concern is the first execution of A. Note that the first job of A experiencing preemption produces fresher data, so preemption of the first job does not extend $d_{A\to B}$ either. Hence, preemption of task A or B cannot extend $d_{A\to B}$ as long as the task set remains schedulable. Therefore each local deadline is not violated by preemption.

Now consider the intermediate tasks in a chain, which are a part of two, two-task scenarios. If preemption extends a job, its does not affect the pair for which it is the consumer since the execution time of consumers are irrelevant. In the pair where it is a producer, preemption extends the completion time of the job which results in fresher local data for its consumer. Therefore preemption of intermediate jobs does not violate freshness. Any preemption and other delays are experienced entirely within a local constraint, i.e. p in preemption time comes with data p newer than the task's local constraint in worst case.

Since all subcomponents of the freshness bound are not violated by preemption, the end-to-end deadline is not violated by preemption. □

While preemption will not violate our freshness constraint, note that preemptability may be key to determining schedulability of the solution.

VII. Multiprocessor Systems

Our formulation extends nicely to multiprocessor systems since we do not make assumptions on how to schedule tasks. The worst case is that all tasks in the chain end up on the same processor which is the case considered. However, no assumptions on task location were made so tasks could be executed on different processors. Our formulation selects periods for each producer and is agnostic to the consumer. For each producer-consumer pair, the period of the producer is chosen so that the output is updated within the desired freshness constraint. The consumer's location is irrelevant as long as delays introduced in a multiprocessor system are accounted for in the data delay values. With this, it is simple to see that the generality of the formulation includes the multiprocessor scenario, and adding processors does not change the correctness of our solution. On the other hand, more processors helps ensure the schedulability of the resulting task set by increasing the available resources and reducing job preemption. However, one must remember to use the schedulability test for the multiprocessor algorithm used.

VIII. Evaluation

A. 3-Task Theory Evaluation

For evaluating our three-task theory, we perform four experiments. In the first, we run the tasks on a real-time system and record the miss rate and average freshness of data. Next, we take chains of tasks and a given freshness guarantee, and then check if the produced results are theoretically schedulable. Later, we evaluate how schedulability and utilization may change when altering the desired freshness. Lastly, we examine one task chain and see how the period assignments change when we modify the freshness bound.

For our evaluation we consider uniprocessor systems for simplicity. To test schedulability we use response time analysis for rate monotonic (RM) scheduling and a utilization bound of 1 for earliest deadline first (EDF) scheduling. For our task parameters, we use values from the E3S benchmark[1]. We used all sections of the benchmark, including task sets from the automotive, consumer electronics, networking, office automation, and telecommunications industries.

The task graphs we obtained have between 2 and 10 tasks. To gather as many relevant examples as possible for these special cases, we sometimes combined several serial tasks into one large task, or used several branches of parallel tasks as separate examples for 2-and-3-task evaluations. However, we always included the beginning and ending tasks and some chain from the former to the latter, so that the period values in the benchmark could be used meaningfully. We used WCETs of the first hardware configuration given in the benchmark, and chose BCETs to be half of WCET. We considered communication negligible for evaluation as we global variables between threads of the same process. In total, we collected 26 three-task scenarios and 29 two-task scenarios from the benchmark suite.

[1] http://ziyang.eecs.umich.edu/~dickrp/e3s/

For our first experiment, we ran task chains with parameters chosen by our theory on a real system, using a freshness bound of 25% of the period of the final task so as to have difficult enough chains for meaningful evaluation. We used a Linux testbed[2] with either the fixed priority scheduler (SCHED_FIFO) with priorities set to RM, or the EDF scheduler (SCHED_DEADLINE). We set up the tasks to run for 99% of the specified WCET to account for scheduler overhead and context switching. The final task maintains statistics such as average freshness and freshness miss ratio. We ran this for all task sets that passed the respective schedulability test after having parameters selected by our formulation. The average utilization of the task sets was 40.6% with a low of 1.9% and a high of 97.4%. We ran each task set for one minute, which resulted in hundreds to hundreds of thousands of executions of each task. Tasks were all run on the same core with no external loads added.

Under both RM and EDF, all reads by the last task were within the desired freshness bound, i.e. there were no misses, except for far outliers (3-5x the bound) which we discarded as OS, background task, and other outside interferences, as these are far larger than could be reasonably expected. The average percentage of the freshness bound that was consumed before reading a value as well as the average percentage bound consumed for the stalest value were recorded. The top section of Table I summarizes the results. We see that on average the value was well within the maximum bound. Our experiments suggest that RM slightly outperforms EDF when considering average freshness. RM shows the greatest advantages for maximum staleness observed. We believe this is because our 2-and-3-task theory often assigns lower periods, and hence higher priorities, to tasks earlier in the task chain. This may help freshness since producer tasks are executed first which prioritizes pushing newer values though the chain over consuming older values.

TABLE I
COMPARISON OF AVERAGE FRESHNESS AND MISS PERCENTAGE FOR TASK SETS UNDER RM AND EDF SCHEDULING

Original Task Set	RM	EDF
Freshness Miss Rate (%)	0	0
Average Freshness (% of bound)	7.8	10.3
Average Max Freshness (% of bound)	12.9	52.9
Stringent Task Set	RM	EDF
Freshness Miss Rate (%)	0	0
Average Freshness (% of bound)	11.5	13.2
Average Max Freshness (% of bound)	26.2	81.9

The statistics collected in the trial were weighted down by several easily scheduled task sets. To get an idea of how the formulation performed on heavier task sets, we removed task sets where 20% or less of the bound was consumed on average. In this case the average utilization was 45.2% with a low of 20.6% and a high of 83.4%. We observed the same trend as before where RM outperforms EDF on average, and especially in the maximum staleness metric. The results are shown in the bottom section of Table I.

[2]Ubuntu 14.04, i7-6770HQ CPU, 16GB DDR3 RAM

For our second experiment, we looked at the theoretical schedulability of the task sets. Figure 4 shows the percentage of task chains which were schedulable using RM or EDF scheduling. The first two sets of bars depict the schedulability percentage when freshness bounds are set equal to the period of the last task in the chain. The second two sets of bars depict the schedulability after we quarter the freshness bound. If the freshness bound is too tight, our formulation will sometimes result in non-positive periods. These cases were included in the evaluation and were considered unschedulable task chains.

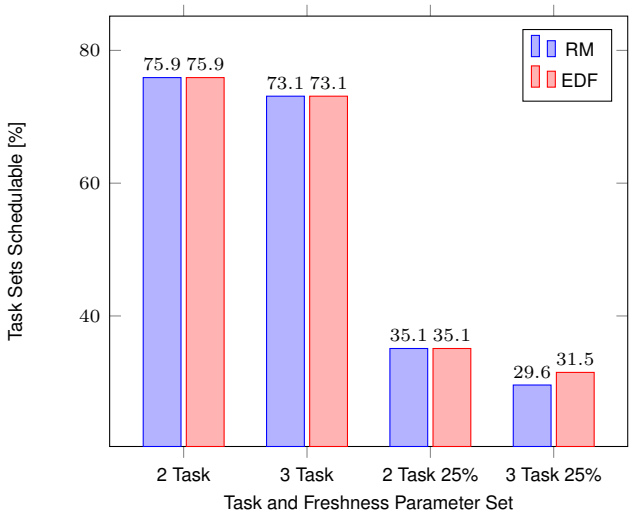

Fig. 4. Formulation Schedulability. The first two bar sets show schedulability of task chains with freshness bound equal to period of the last task. The last two bar sets show schedulability when the bound is quartered.

Both schedulers scheduled the same percentage of tasks for the first 3 bar sets and inspection revealed both scheduled the same task chains. The last bar set depicts a task set which showed discrepancy between RM and EDF schedulers, where EDF shows a slight schedulability advantage.

For our third experiment, we choose to repeatedly decrease the freshness bound of each schedulable task set by 10% from an original freshness bound (equal to the period of the last task in the chain). We then recorded the average utilization and percentage of task chains that are schedulable under EDF. This gives us an idea of the trade-off between freshness and schedulability. The results are charted in the top two lines of Figure 5. We can see that above about 50% of the original bound a 10% decrease in freshness bound results in about 5% increase in utilization. The areas where utilization decreases (30% and lower) is due to less task sets being considered, since non-schedulable task chains were omitted from the utilization calculation. We see schedulability is not greatly hindered until we get to around 30% of the original freshness bound. The primary significance of Figure 5 is that above 30% of original freshness bound, where the task set stays largely the same, we see only modest increases in utilization when we tighten the freshness bound.

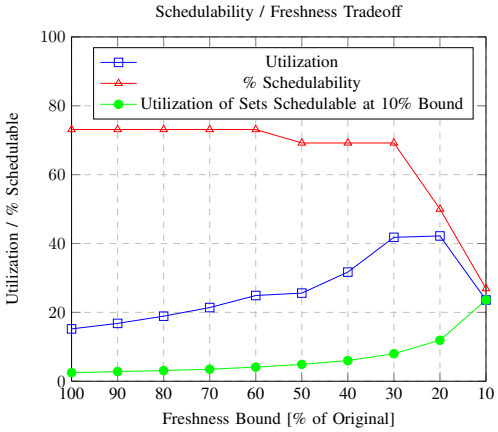

Fig. 5. Schedulability, Utilization vs. Freshness.

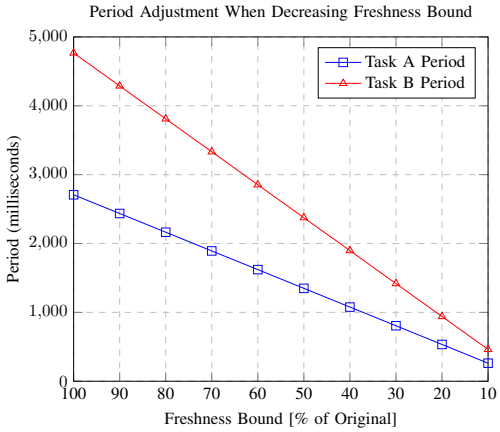

Fig. 6. Freshness vs. Periods. Using one task chain that is schedulable with a freshness bound at 10% of task C's period, we plot how our formulation assigns periods tasks A and B to maintain freshness.

Next we modified the previous experiment by performing the evaluation only on those chains which are schedulable at 10% of the original freshness bound, to prevent noise due to the differences in the schedulable task chains. In the bottom line of Figure 5 we see a more intuitive curve. It is lower because more difficult chains were not schedulable with very low freshness bounds. Note that a 10% reduction does not change utilization much if above 50% of the original freshness bound. We see again that utilization is more greatly affected when below 30% of the original freshness bound.

For our fourth experiment, we chose a three task chain which was schedulable with freshness bound of 10% of the period of the last task. This chain was from the automotive industry. We then calculated the periods of the input tasks according to our theory in increments of 10% of the original freshness bound. We'll name our tasks in the chain as $A \rightarrow B \rightarrow C$. In this case, task A has a WCET of 50, task B a WCET of 155, and task C a period and WCET of 15000 and 50 milliseconds respectively. BCETs were set to half of WCET. How the assigned periods of tasks A and B change to reflect the change in freshness bound is displayed in Figure 6. We see that changes in freshness causes linear changes in the periods of both tasks to account for the new bound.

B. Optimization Formulation Evaluation

To evaluate the optimization formulation we implemented the described optimization problems in MATLAB. In our figures we will denote the general formulation as "G-" and the RM-targeting formulation as "R-" followed by the scheduler used. For example, G-RM denotes data using solutions from the general formulation ran under a RM scheduler.

For each chain length, we randomly generated tasks with between 1 and 5 units of work for their BCET, and added another 1 to 5 to the BCET to determine the WCET. We then generated an end-to-end deadline equal to the sum of the task WCETs multiplied by a random integer between 3 and 10, in order to add slack to the system that depends on the number of tasks and their execution lengths so that chain lengths and WCETs do not adversely affect the results. All random variables were drawn from a uniform distribution. We used the same necessary and sufficient schedulability uniprocessor tests for RM and EDF as before. Figure 7 depicts the schedulability of the generated task sets. Each bar represents the percentage of 1000 generated task sets that were schedulable. Note the same 1000 task sets were used for each formulation-scheduler combination.

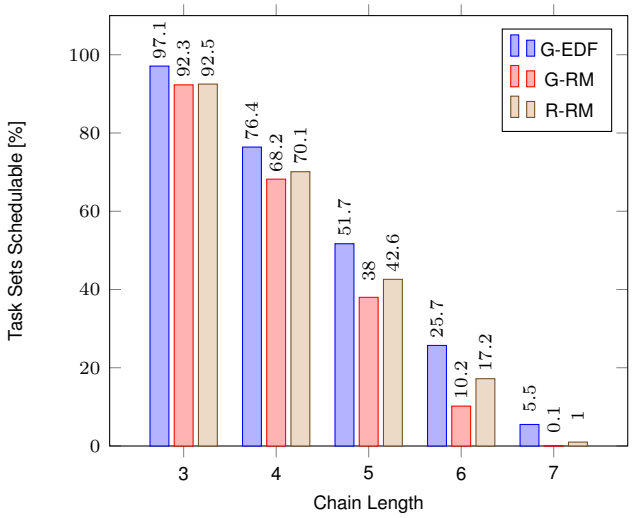

Fig. 7. Schedulability of Optimization Results with Differing Chain Lengths.

We see that EDF performs more favorably in terms of schedulability. However, we do see a small improvement in RM schedulability when we use our RM-targeted formulation. This improvement seems to increase with chain length.

To show that our method is applicable to longer chain lengths and to analyze the scaling ability of the method according to the task set, we ran the above optimization and analysis but with a harder and easier task sets. More precisely, we kept the same task set as depicted in Figure 7 but with the end-to-end freshness deadline either halved (twice as hard) or

doubled (half as hard). The results of the latter are shown in Figure 8.

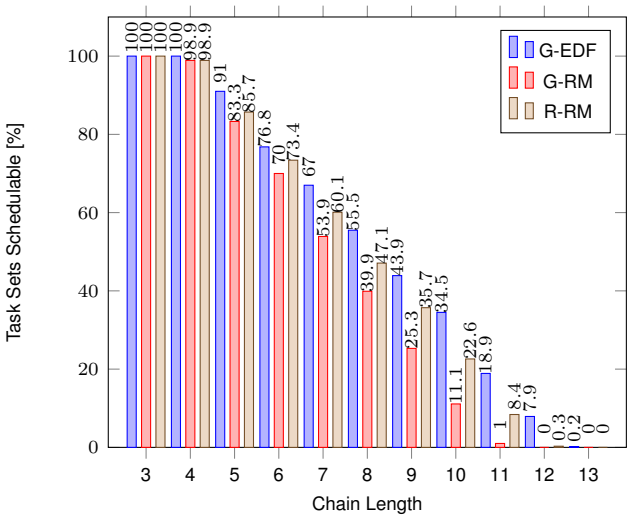

Fig. 8. Schedulability of Optimization Results with Differing Chain Lengths: Deadlines Doubled.

We can see that the trend remains about linear in schedulability after the modification. Similar scaling was seen in the more difficult task set but the figure is omitted due to space considerations. These experiments suggest that the formulation's schedulability generally scales inversely with chain length. There is no chain length from our trials that causes the schedulability to unexpectedly drop. It appears that our formulation can handle approximately twice the chain length for each halving in difficulty and vice versa, with some small losses due to pessimism.

In our final evaluation we again referenced the E3S benchmark to evaluate real-world performance with multiple chain lengths. We chose chains that all three formulations could schedule. We ran the chains and collected freshness data as we did in the three-task theory evaluation. The results are summarized in the Table II.

TABLE II
COMPARISON OF FRESHNESS FOR TASK CHAINS UNDER RM OR EDF SCHEDULING USING RESULTS FROM THE GENERAL OR RM-SPECIFIC FORMULATION.

	G-EDF	G-RM	RM-RM
Average Freshness (% of bound)	13.73	4.77	4.77
Average Max Freshness (% of bound)	56.20	33.75	31.89

RM scheduling provided better average and maximum staleness when executing the results from either optimization, further supporting that for real-world task sets similar to the E3S benchmark, RM provides greater freshness than EDF.

Overall, our optimization formulation seems adequate for short to moderate chain lengths. We suspect that many real-life systems use chain lengths within the evaluated ranges, including all of the E3S benchmark chains. As far as optimization efficiency, the optimization problem finished on average in less than a second. This suggests that the scaling of the optimization problem would not be a limiting factor in the use of our system.

IX. Discussion

Although these results are promising, we consider possible avenues of further improvement. Our approach abstracts away the scheduling algorithm. Therefore, given information about the scheduling algorithm, prioritization, and preemptability, one could likely produce freshness-enforcing parameters that result in lower utilization.

We reiterate the limitation that this method does not guarantee the schedulability of the task set. This is due to our scheduler agnosticism. However, this is easily remedied by scheduler-specific schedulability tests. If the particular value produced by this method is unschedulable, there may or may not exist other parameters that schedule the task set while ensuring freshness for a given system.

While we aimed to remain scheduler agnostic, future work will look into modifying the optimization problem to ensure schedulability. The RM-specific formulation has most promise since those results likely improve schedulability under RM. The RM formulation could also be modified to produce harmonic periods, which increases schedulability. It may be possible to accommodate other schedulers with additional constraints or added variables in the optimization problem. For example, an EDF formulation may be able to represent remaining deadline given our already-defined variables and some notion of time.

Finally, our current formulation specifies data freshness, network latencies, and other job latencies as absolute worse-case values. This is acceptable for systems where these values are well defined and bounded, such as dedicated hardware running only specified software. Future work could broaden these values to include functional values such as supply curves. Alternatively, statistical values and guarantees could be explored as apposed to absolute ones.

X. Related Work

There are two lines of work in literature which are broadly related to the problem we have outlined. The first line of work looks into various aspects of data freshness as a data quality metric in systems such as Web Servers [8], Real-Time Databases [9], [10], etc. A second aspect which is important in our work is the selection of parameters of a set of real-time tasks such that the tasks are schedulable and particular system objectives are optimized. However, there are few works which have combined the problem of selecting periods of real-time tasks while guaranteeing end-to-end data freshness. We now present the differences between existing research and the work which we present.

In [11], [12], the authors present a comprehensive overview of freshness as a data quality metric and also a framework for analysis of freshness. Freshness is defined with respect to two notions, namely *currency factor* and *timeliness factor*. The currency factor represents how stale a data is with respect to its

source, while the timeliness factor represents how old a data is since its creation/update at the source. Several metrics are described to measure freshness according to these definitions. There are also works which have considered freshness related problems in real-time databases. In [9], the effects of update policies to maintain data freshness of derived data was studied in the context of a distributed real-time database and a novel update policy was proposed. Conversely, an immediate update policy based on a QoS management architecture is used by [10]. However, none of the above works consider freshness in a task context or deriving parameters to maintain the target freshness constraint.

The problem addressed in [13] was to optimize data freshness along with other objectives such as throughput in a multi-server information-update system. The authors propose a preemptive Last-Come First-Served policy and show that it optimizes freshness, throughput, and delay performance in infinite buffer systems. A deferrable scheduling algorithm is proposed in [14] for maintaining data freshness so as to minimize the update workload. The sampling time of a transaction job is deferred as late as possible while guaranteeing the temporal validity of the data. In contrast, our work provides an analytical framework to derive the period of tasks in order to provide the required end-to-end data freshness constraint and does not restrict the scheduling policies.

There are prior works that look at choosing task periods to meet individual latency requirements in real-time systems using dynamic priority [15] and static priority [2] scheduling methods. In [3], the authors propose a heuristic to derive a feasible period-deadline combination such that the task set becomes schedulable under the assumption that task deadlines are piecewise first-order differentiable functions of the respective periods. Wu et. al. [16] presented an approach to select the task periods and deadlines, under EDF scheduling, to enhance the control performance of a system. However, the above works do not consider data freshness as a constraint in their frameworks.

There is one work [17] that has looked at the problem of period selection combined with data freshness requirement. Three classes of timing constraints are considered in [17] namely freshness, correlation, and separation. An iterative pruning-based heuristic is used to derive the period, offset, and deadline of tasks such that the end-to-end constraints are met. In contrast, our work proposes an analytical framework to derive the task periods remaining completely agnostic to the scheduling strategy. Moreover, in [17], the periods considered for producer tasks were harmonic with respect to the periods of the consumer tasks. We do not impose such a restriction, which may allow for scheduling more task sets and allows more flexibility when minimizing our optimization objective.

XI. Conclusion

In this paper we considered the freshness of data consumed by tasks within a periodic task system. We aimed to select the periods of input tasks in order to ensure the freshness of data consumed later chain tasks. Without assumptions regarding the scheduler, we proved upper bounds on the periods of tasks in order to ensure the freshness of data through chains of tasks of length two and three for uniprocessor systems. We then extended the theory to an optimization problem suitable for any chain length and configuration.

Acknowledgment

This work was supported in part by ONR N00014-16-1-2195, NSF CNS 1138847, NSF CNS-1505799, and the Intel-NSF Partnership for Cyber-Physical Systems Security and Privacy.

References

[1] E. Bini and M. Di Natale, "Optimal task rate selection in fixed priority systems," in *Proceedings of 26th IEEE International Real-Time Systems Symposium*, Dec 2005, pp. 11 pp.–409.

[2] D. Seto, J. Lehoczky, and L. Sha, "Task period selection and schedulability in real-time systems," in *Proceedings of 19th IEEE Real-Time Systems Symposium*, Dec 1998, pp. 188–198.

[3] T. Chantem, X. Wang, M. Lemmon, and X. Hu, "Period and deadline selection for schedulability in real-time systems," in *Proceedings of Euromicro Conference on Real-Time Systems*, July 2008, pp. 168–177.

[4] C. Belwal and A. Cheng, "Generating bounded task periods for experimental schedulability analysis," in *IFIP 9th International Conference on Embedded and Ubiquitous Computing*, Oct 2011, pp. 249–254.

[5] A. Segev and W. Fang, "Currency-based updates to distributed materialized views," in *Proceedings of Sixth International Conference on Data Engineering*, Feb 1990, pp. 512–520.

[6] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *J. Manage. Inf. Syst.*, vol. 12, no. 4, pp. 5–33, Mar. 1996. [Online]. Available: http://dx.doi.org/10.1080/07421222.1996.11518099

[7] D. Golomb, D. Gangadharan, S. Chen, O. Sokolsky, and I. Lee, "Technical report: Data freshness over-engineering," https://www.dagaengolomb.com/techrep.pdf.

[8] A. Labrinidis and N. Roussopoulos, "Exploring the tradeoff between performance and data freshness in database-driven web servers," *The VLDB Journal*, vol. 13, no. 3, pp. 240–255, 2004.

[9] Y. Wei, S. H. Son, and J. A. Stankovic, "Maintaining data freshness in distributed real-time databases," in *Proceedings of 16th Euromicro Conference on Real-Time Systems*. IEEE, 2004, pp. 251–260.

[10] 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, 2004.

[11] M. Bouzeghoub and V. Peralta, "A framework for analysis of data freshness," in *Proceedings of the International Workshop on Information Quality in Information Systems*. ACM, 2004, pp. 59–67.

[12] V. Peralta, "Data freshness and data accuracy: a state of the art," *Reportes Técnicos 06-13*, 2006.

[13] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing data freshness, throughput, and delay in multi-server information-update systems," *arXiv preprint arXiv:1603.06185*, 2016.

[14] M. Xiong, S. Han, and K.-Y. Lam, "A deferrable scheduling algorithm for real-time transactions maintaining data freshness," in *Proceedings of 26th IEEE Real-Time Systems Symposium*. IEEE, 2005, pp. 11–pp.

[15] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proceedings of 17th IEEE Real-Time Systems Symposium*. IEEE, 1996, pp. 13–21.

[16] Y. Wu, G. Buttazzo, E. Bini, and A. Cervin, "Parameter selection for real-time controllers in resource-constrained systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 610–620, 2010.

[17] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Transactions on Software Engineering*, vol. 21, no. 7, pp. 579–592, 1995.