Data Freshness Over-Engineering: Formulation and Basic Results

Dagaen Golomb

University of Pennsylvania

I. Introduction

There are many applications where a task needs to consume data in order to perform its duties. In real-time systems, this input is often sensor data that is crucial for determining the behavior of the system. Input may also be from the output of other tasks. In these types of systems, there is usually an explicit or implicit timeliness requested 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, tasks have been over-engineered to provide fresh data. Imagine a task that must perform some computation on data, produced by another task, every one second. Should the task producing the required data run every second? 500 milliseconds? 100 milliseconds? How is the designer to decide? In these situations, engineers may err on the side of caution and choose to produce the input data faster than necessary, which they then need to test in order to evaluate safety. For example, A task that reads a sensor value and forwards it to another task may be dispatched fifty times a second even though the consuming task only needs data younger than one tenth of a second for safety. While this is presumably safe, it reduces the efficiency (and possibly schedulability) of the system, and there is no formal method for us to decide if this value is safe regardless. This leads to more dependence on testing. As Microsoft acknowledges, overenginnering alway leads to wasted effort [4]. This often leads to several cycles of parameter tuning in order to ensure the safety and efficiency of the system without ultimately guaranteeing optimal efficiency, as noted by several others in the field [2, 6, 3, 1]. This paper outlines a formalization of this over-engineering strategy and presents results for choosing the period of input tasks given little information about the underlying system.

II. Motivation

Something goes here.

III. Goal

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, or alternatively staleness, is not a new concept. A wide variety of domains consider this notion, with the definition tailored to the domain, all of which agree that in their domains the quality of data is not merely a function of its "correctness" or accuracy. Common notions use wording such as currency [5], which describes how much time has passed since data collection or generation, and timeliness [7], which measures how old the data is when collected. For our purposes, these two are essentially the same notion. We will consider freshness to be

the time elapsed since the data was produced, particularly when it is output by a generation or collection task.

In particular, we are going to examine the following scenario: given a task Z with fixed period that consumes data that makes its way through a task chain $A, B, C, \ldots$, choose the periods for $A, B, C, \ldots$ such as to ensure our freshness bound is always enforced. That is to say, when task Z runs the age of the data created by task A is less than our freshness bound.

The solution to this problem is not always unique. Given a set of possible solutions, we wish to rate them against some metric to select one that best fits our needs. There are several metrics one could use; in this work we focus on minimizing total 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 sector. 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.

IV. Formalization

I. Definitions

Our model assumes a periodic task set on a unicore system. For periodic task sets, each task A is characterized by the following:

Period	P_A
Relative Deadline	D_A
Worst-Case Execution Time	E_A^u
Best-Case Execution Time	E_A^l

A job is an instance of a task, i.e. one of the actual executions of a task. Each task can, and often does, produce multiple jobs. In our periodic model, a job is released at most every P_A time. Each i^{th} job of task A has the following attributes:

Release Time	r_A^i
Finish Time	f_A^i

Note that $f_A^i - r_A^i$ will always be bounded below by the best-case execution deadline (when the task is executed immediately and ran to completed) but not bounded by the worst-case execution time if the system allows preemption or if the job has to wait before being scheduled onto the processor. However, assuming the system is schedulable, this difference is bounded by the task period. If this is not the case then the job has experienced a deadline miss.

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 jobs of task A and consumed by jobs of 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.

II. Assumptions

Since we do not know the nature of the particular 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 time. This means that data they consume must be produced before they start execution. Since this could happen immediately at release time, without knowing the scheduler and thus when the task may execute, we assume jobs read input values at the time of their release.

On the other hand, we assume data is produced at the very end of a task execution. Hence, if a task produces some data, it does so 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 same task. Also note that in our notion, we will consider the data to be generated after the initial task outputs it. While this does not explicitly consider the execution time of this first task, collection tasks are often small. This definition keeps our formulation consistent in that data is only considered updated at the end of a task. If the initial task has non-trivial execution time, or the designer would like to consider it, its WCET can be subtracted from the desired freshness bound.

We also assume that data transfers are instantaneous. While this is not the case for real system, data transfer times are often quick in comparison to task execution times. We note that slower methods of data storage or transfer may invalidate this formulation. However, one method to account for data storage or access time is to inflate the task WCET to include these. This works because of our previous two assumptions on when data is consumed and produced. From here on, we will assume data transfer times are either negligible or accounted for in task execution times.

While the above two assumptions are "extremes" it should not be construed to imply worst case. We use these assumptions to generalize the problem to any task set. Due to these assumptions, our notion of freshness will be relative to the completion of data generating tasks and the release of consuming tasks. We feel this is a reasonable definition of freshness under given our assumptions and abstraction, and we will formally define this in the next section.

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.

For simplicity, we will only be considering unicore systems.

III. Requirements

Using the notation and assumptions above, we can define the freshness of data produced by task A and consumed by job i of task B with the quantity

$$r_B^i - f_A^{C_A(r_B^i)}$$

Since we demand that our data freshness from A to B be bounded by $d_{a\to b}$, and the above represents the data freshness of job i of task B, we need

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

in order to ensure job i of task B consumes fresh data.

Finally, since data freshness must hold for all jobs of task B, our final freshness requirement is as follows

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

IV. Problem Statement For Two Tasks

Our formal problem statement is as follows. Let E_A^* denote that we have both worst-case and best-case execution times for task A, and let $U(T)$ denote the system utilization.

Given	$E_A^*, E_B^*, P_B,$ and $d_{A\to B}$
Find	P_A
That Minimizes	$U(T)$
Subject To	$\forall i, r_B^i - f_A^{C_A(r_B^i)} \leq d_{A\to B}$

Note that we do not explicitly consider schedulability in the formulation for both generality among algorithms and for simplicity. Due to this, our solution will not guarantee schedulability, which must instead be confirmed using algorithm-specific methods afterwards. On the contrary, our solution actually relies on the schedulability of the system for the end result as described later, and is thereby invalidated if the system is ultimately unscheduable.

It is clear this minimization problem has some solution: we can lower utilization by increasing our only free variable P_A, but for any constant freshness guarantee at some point increasing P_A will cause our freshness bound to not hold. For example, if we set $P_A = 2 \cdot P_B$ it is clear freshness will not hold, while setting P_A arbitrarily close to 0 will cause the bound to hold (disregarding schedulability). At some point between these two, the system switches from upholding the freshness bound to not upholding it.

V. Two Task Result

We will first consider the simplest case, where there are only two tasks: one producing a value and one consuming it. By considering just the first task A we can see that the worst case is shown in Figure 1. Note that this scenario assumes schedulability to ensure that one job of A must run and complete within each period. This is where our assumption of schedulability becomes vital for our correct solution.

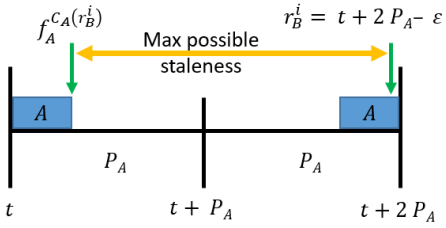

Figure 1: *Maximum Staleness Scenario for Data from Task A.*

Figure 1 labels the maximum separation between the publishing of output data from the depicted task. In order to maximize the staleness, we assume the reading task, task B, is released arbitrarily close, but before, the finish of the second job of task A. If we take push the second task towards the end of its period we see that the maximum staleness is when the second job finishes at the instant its period ends.

Theorem 1 *The scenario in Figure 1, i.e., when a job is executed and completed at the very beginning of a period and the next job completes at the very end of the next period, is the upper bound scenario for data freshness, as defined in Section III, produced by that task.*

Proof First note that one job must be executed within each period, as per definition of periodic tasks and our assumption that the task set is schedulable.

Consider any placement of two jobs of a task within two consecutive periods. Assume this instance is not the instance depicted in Figure 1. Then at least one of the following apply:

Case 1 *The job in the first period is not completed as soon as possible. In this case, move when this task first starts execution ϵ earlier. This increases the staleness by ϵ and moves the job's finishing time towards the tasks execution time after that period.*

Case 2 *The job in the second period is not completed as late as possible. In this case, move when this task first starts execution ϵ later. This increases the staleness by ϵ and moves the job's finishing time close to the end of the period.*

Since all instantiations of the problem can be moved closer to the depicted instance while strictly increasing the staleness, it holds that this instance is the unique worst case for staleness.

Now that we have proved the above scenario is the worst case with regards to the freshness of data from task A, we can use algebra to 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 time of A less ϵ, and we want this to be less than our freshness bound $d_{A\to B}$, i.e. $d_{A\to B} \geq 2P_A - E_A^l - \epsilon$. We can then solve for P_A to prove the following lemma.

Note that we assume the best case execution time for the first task in order to stretch the freshness as long as possible. If we assume that the first task in a two task scenario executes at its worst case, then E_A^l in our solution will be replaced with E_A^u.

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

Proof

$$
\begin{aligned}
d_{A\to B} &\geq 2P_A - E_A^l - \epsilon && \text{From Figure 1}\\
d_{A\to B} &\geq 2P_A - E_A^l && \epsilon \to 0\\
2P_A &\leq d_{A\to B} + E_A^l && \text{Add } E_A\text{, swap sides}\\
P_A &\leq \frac{d_{A\to B} + E_A^l}{2} && \text{Divide by 2}
\end{aligned}
$$

Thus we see that if we set P_A to the the above quantity we ensure that the output of task A is always at most $d_{A\to B}$ old, and therefore can only ever be at most this old when consumed by task B. Since P_A is the only parameter under our control, we can also set it less than this quantity and maintain the freshness guarantee (this would in essence force $d_{A\to B}$ smaller, and a smaller freshness guarantee will always ensure a larger one as well).

Recall that our fitness metric is total utilization. Since the parameters of B are irrelevant in the solution and E_A is fixed, our only optimization parameter is P_A. It is trivial to see that choosing P_A as large as possible will minimize the task set utilization, and thus the above, when set equal to the right-hand quantity, is the solution for the two task scenario while optimizing over task set utilization. Note that this value may not produce a schedulable task set. If this is the case, the largest schedulable period that meets the above inequality is the solution. Note that for many systems decreasing P_A can only worsen schedulability. For these systems, if the task set is unschedulable when set equal to the above quantity then there does not exist any P_A for which the freshness guarantee is met with a schedulable task set.

VI. Three Task Result

We will 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 that is consumed by task B, which in turn produces output

that is consumed by task C. Here we will now focus on a maximum staleness for the input which was used by B to in turn produce the output that C uses. In other words, we want to limit the staleness of the output of A that is eventually used by C.

In this scenario we have a new requirement, defined similarly to the two task scenario. 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 the jobs of tasks A and task C. The formalization is similar to the two task scenario:

Given	E_A^*, E_B^*, E_C^*, P_C and $d_{A \to C}$
Find	P_A and P_B
That Minimizes	$U(T)$
Subject To	$\forall i, r_C^i - f_B^{C_B(r_C^i)} + E_B^u + r_B^{C_B(r_C^i)} - f_A^{C_A(r_B^{C_B(r_C^i)})} \leq d_{A \to C}$

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

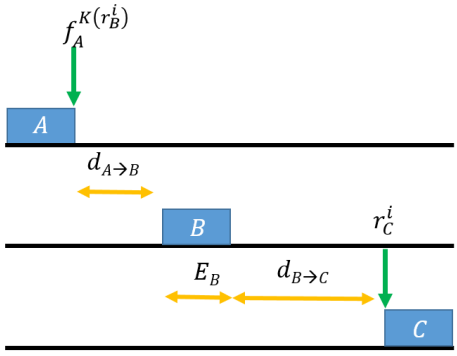

Figure 2: *Maximum Staleness Scenario for Data from Task A.*

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 before and after the execution of task B, the freshness of data from task A to task C is ensured. Note that these added constraints will not be present in our final solution. They are added here to allow us to introduce Lemma 1 while being able to consider task B (considering just $d_{A \to C}$ would not guarantee that task B is ran between A and C). These local constraints are used as free variables and are never assigned concrete values. We will optimize with regards to them and use our Lemma to determine period assignments that rely only on the end-to-end constraint and the task execution times.

Note that we have one execution of task B between tasks A and C. It is up to the scheduler to ensure this. Also note that we use E_B^u in this formulation. This works fine if task B runs without preemption. Since we do not know the scheduling in our system, this is not guaranteed. If task B may run with preemption, E_B should be inflated to task B's worst case response time (WCRT) instead. It would be job of the system designer to determine this WCRT with respect to the scheduler and other tasks including the ones here, which may require iteratively solving for the period/WCRT combination. We will not consider this here. In particular, probably the worst assumption with this formulation as written is that task B needs to run immediately upon entering the system and without preemption.

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,

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

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

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

Notice how, once again, we use the best-case execution time for the first task in the chain in order to maximize the possible staleness. Note how this will be a minor issue, as we are trying to minimize utilization but are now using the best-case execution time where a worst-case execution may occur. Since a worst-case execution will produce fresher output than a best-case under our assumption that results are produced at the end of a task, we will solve this by introducing some pessimism. For our freshness guarantee, we will use the best-case execution time whereas when calculating utilization we will use the worst-case execution time. That is, our solution will ensure freshness even in with best-case execution of the first task, while minimizing the maximum utilization the system could experience. This means that our solution for $d_{A\to B}$ will be under the assumption that the worst case-execution occurs. Hence, when we find out $d_{A\to B}$ we will subtract the difference between E_A^l andE_A^u so that this freshness holds even during best-case execution of task A, i.e., we will further decrease the period of A to account for this.

Note that in our above formulation for three task scenario, the only free variables that effect utilization are P_A and P_B. Therefore, we can achieve the same solution by minimizing over $\left(\frac{E_A}{P_A} + \frac{E_B}{P_B}\right)$ since the utilization from task C is constant. This is what we will use as our minimization objective.

Substituting the values derived from Lemma 1, we can transform the minimization objective:

$$\begin{aligned}
U(T) &= \left(\frac{E_A^u}{P_A} + \frac{E_B^u}{P_B}\right) && \text{Revised Objective}\\
&\geq \left(\frac{E_A^u}{\frac{d_{A\to B}+E_A^l}{2}} + \frac{E_B^u}{\frac{d_{B\to C}+E_B^l}{2}}\right) && \text{Substitution}\\
&= \left(\frac{E_A^u}{\frac{d_{A\to B}+E_A^l}{2}} + \frac{E_B^u}{\frac{d_{B\to C}+E_B^l}{2}}\right) && \text{Min } U(T) \to \text{Max Periods}\\
&= \frac{2E_A^u}{d_{A\to B} + E_A^l} + \frac{2E_B^u}{d_{B\to C} + E_B^l} && \text{Simplification}
\end{aligned}$$

Using this modified objective, we can minimize 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

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

and

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

Proof This is a simple optimization problem that can be solved using elementary calculus methods. In particular, we will use Lagrangian multipliers. We will optimize our two controllable parameters, the local constraints, and then use these two decide upon periods for the tasks as per our Lemma.

To optimize the above with our constraint of the form $x + b + y = z$, we use the following:

$$\frac{2E_A^u}{d_{A\to B}+E_A^u} + \frac{2E_B^u}{d_{B\to C}+E_B^u} + \lambda(d_{A\to C} - d_{A\to B} - E_B^u - d_{B\to C})$$

We now take the partials with respect to our free variables, $d_{A\to B}$ and $d_{B\to C}$:

$$\frac{\partial}{\partial d_{A\to B}} = -\frac{2E_A^u}{(E_A^u + d_{A\to B})^2} - \lambda$$
$$\frac{\partial}{\partial d_{B\to C}} = -\frac{2E_B^u}{(E_B^u + d_{B\to C})^2} - \lambda$$

We set these equal to zero and solve for our λ's ...

$$-\frac{2E_A^u}{(E_A^u + d_{A\to B})^2} - \lambda = 0 \to \lambda = -\frac{2E_A^u}{(E_A^u + d_{A\to B})^2}$$
$$-\frac{2E_B^u}{(E_B^u + d_{B\to C})^2} - \lambda = 0 \to \lambda = -\frac{2E_B^u}{(E_B^u + d_{B\to C})^2}$$

With two values of λ, we can form an equality, which we can use with our constraint equation to solve for $d_{A\to B}$ and $d_{B\to C}$ via a system of equations:

$$-\frac{2E_A^u}{(E_A^u + d_{A\to B})^2} = -\frac{2E_B^u}{(E_B^u + d_{B\to C})^2}$$
$$d_{A\to B} + E_B^u + d_{B\to C} = d_{A\to C}$$

We'll solve for $d_{A\to B}$ first.

$$
\begin{aligned}
-\frac{2E_A^u}{(E_A^u + d_{A\to B})^2} &= -\frac{2E_B^u}{(E_B^u + d_{B\to C})^2} \\
(2E_B^u)(E_A^u + d_{A\to B})^2 &= (2E_A^u)(E_B + d_{B\to C})^2 && \text{Multiply By -1 Then Cross Multiply} \\
(E_A^u + d_{A\to B})^2 &= \frac{E_A^u}{E_B^u}(E_B^u + d_{B\to C})^2 && \text{Rearrange and Simplify} \\
d_{A\to B} &= \pm\sqrt{\frac{E_A^u}{E_B^u}}(E_B^u + d_{B\to C}) - E_A^u \\
&= \pm\sqrt{\frac{E_A^u}{E_B^u}}(E_B^u + d_{A\to C} - E_B^u - d_{A\to B}) - E_A^u && \text{Substitute From Constraint} \\
&= \pm\sqrt{\frac{E_A^u}{E_B^u}}(d_{A\to C} - d_{A\to B}) - E_A^u && \text{Simplify} \\
&= \pm\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to C} - \pm\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to B} - E_A^u \\
(1 + \pm\sqrt{\frac{E_A^u}{E_B^u}})d_{A\to B} &= \pm\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to C} - E_A^u \\
d_{A\to B} &= \frac{\pm\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to C} - E_A^u}{(1 + \pm\sqrt{\frac{E_A^u}{E_B^u}})}
\end{aligned}
$$

Assigning the $\pm$ both ways, we obtain two possible solutions:

$$
d_{A\to B} = \frac{\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to C} - E_A^u}{(1 + \sqrt{\frac{E_A^u}{E_B^u}})} \qquad \text{or} \qquad d_{A\to B} = \frac{-\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to C} - E_A^u}{(1 - \sqrt{\frac{E_A^u}{E_B^u}})}
$$

And now we solve for $d_{B\to C}$ using this and the constraint equation:

$$
d_{B\to C} = d_{A\to C} - E_B^u - \frac{\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to C} - E_A^u}{(1 + \sqrt{\frac{E_A^u}{E_B^u}})} \quad \text{or} \quad d_{B\to C} = d_{A\to C} - E_B^u + \frac{\sqrt{\frac{E_A^u}{E_B^u}}d_{A\to C} + E_A^u}{(1 - \sqrt{\frac{E_A^u}{E_B^u}})}
$$

Note that we can drive up utilization arbitrarily higher (and even over-commit the system) by shortening the local constraints, and hence periods, so there is no maximum in our search space. There are also no saddle points: it is clear that from any initial value that increasing either parameter will strictly lower utilization. It follows that both of these points are minima. Now note

that the second potential solutions "cheats" by using negative values. Upon inspection its easy to see how $d_{A\to B}$ can become negative when $E_A > E_B$, and is undefined when these two are equal. Upon further inspection, we see that when this is not the case, $d_{B\to C}$ will be negative instead. This minimum achieves such by using positive and negative values to cancel one another out. Since our local constraints must be positive, this is not a feasible solution and we discard it.

Therefore, our solution is the first point, and we can now use our Lemma to convert these into period assignments:

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

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

Finally, recall that we need the bound to hold even during best-case execution of task A, whereas this assumes worst-case execution. To make sure that the freshness constraint holds even under best case execution, we will base the freshness guarantee from the best-case execution time instead of the worst-case. To do this, we will reduce the period so that even under the best execution of task A our freshness is ensured. Currently, we could be $E_A^u - E_A^l$ too old since we base from the worst case execution but the best base could occur. We want to shorten this freshness constraint. Recall from the two task scenario that the freshness constraint span two task periods. Therefore, if we can shorten each period by half of the above mentioned deficit to tighten our freshness guarantee. With this change, even best case execution will result in the desired freshness, and worst case executions of task A will produce data more fresh than the bound.

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

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

This concludes our proof.

Note that the solution is in regards to constants in our system that are given to us, and do not rely on our temporary local constraints in the solution, although these constraints can be extracted from the given solution.

Also note that our optimization is for worst-case utilization, and these freshness bounds may not be optimal when tasks, specifically task A, exhibits best-case execution. However, our goal is safety so the optimization is done for the worst-case execution.

Once again, this solution may not be schedulable. If it is not, there may or may not be a schedulable task set that produces data with the desired freshness, depending on the scheduling algorithm used. Finding the parameters in such a case may be non-trivial.

Lastly, note that this is a convex optimization problem and that we can always find an arbitrarily small periods that meets our freshness bound, i.e. we always have an interior point that satisfies our constraint and thus strictly feasible. It follows that this problem has strong duality and our solution is indeed our desired parameters.

VII. Discussion

We note several issues with this approach. For one, it generalizes for any scheduling algorithm. Given information about the scheduling algorithm, prioritization, and preemptability, one could likely produce parameters that result in lower utilization. For many schedulers this would mean larger periods for our input tasks than presented in our solution.

The most important limitation to note, as mentioned several times already, is that this method does not guarantee the schedulability of the task set. This is due to its scheduler agnosticism. However, this is easily remedied by scheduler-specific schedulability tests. A weak, universal, necessary assessment would be to check if the utilization is less than 1, since we are considering a uniprocessor system. 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 scheduler. This becomes particularly difficult if extended to multicore systems.

It may be possible to extend this model further to even more tasks. Using the same notation as used for the three task model, one could craft complex minimization problems for a given number of tasks. The primary challenge would then be solving increasingly difficult optimization problems. It may be possible to generalize this method to n tasks but this has not yet been pursued.

VIII. Related Work

Others have investigated proper period selection for tasks. However, most opt to consider a specific scheduler or other assumptions about the task set, or don't consider data freshness as the end goal. [To be expanded]

IX. 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 by the last task in the data flow chain. 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.

References

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

[2] Bini, E., and Di Natale, M. Optimal task rate selection in fixed priority systems. In *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International* (Dec 2005), pp. 11 pp.–409.

[3] Chantem, T., Wang, X., Lemmon, M., and Hu, X. Period and deadline selection for schedulability in real-time systems. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on* (July 2008), pp. 168–177.

[4] Czerwonka, J., Nagappan, N., Schulte, W., and Murphy, B. Codemine: Building a software development data analytics platform at microsoft. *Software, IEEE 30*, 4 (July 2013), 64–71.

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

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

[7] Wang, R. Y., and Strong, D. M. Beyond accuracy: What data quality means to data consumers. *J. Manage. Inf. Syst. 12*, 4 (Mar. 1996), 5–33.