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 implied 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. For example, A task that reads a sensor value and forwards it to another task may run one thousand 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 detracts from the efficiency (and possibly schedulability) of the system. This paper outlines a formalization of this over-engineering strategy and presents results for choosing the period of input tasks.

II. 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.

In particular, we are going to examine the following scenario: given a task Z with fixed period that consumes data that makes it 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.

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. 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.

III. Formalization

I. Definitions

Our model assumes a periodic task set, common in real-time systems, 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

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 every P_A time. Each i^{th} job of task A has the following attributes:

Release Time	r^i_A
Finish Time	f^i_A

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 $C_A(r^i_B)$ denote the most recently completed job of task A before the 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 work off of the extreme case. In this case 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, 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 tasks 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.

While the above to 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. This seems to be a reasonable definition of freshness to us, and we will formally define in the next section.

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

$$r^i_B - f_A^{C_A(r^i_B)}$$

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

$$r^i_B - f_A^{C_A(r^i_B)} \leq d_{A \to B}$$

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

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

Note that we will not explicitly consider schedulability in the formulation for both generality among algorithms and to simplify the formulation. 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, and is thereby invalidated if the system is ultimately unscheduable.

IV. Problem Statement

Our formal problem statement is as follows.

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

IV. 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.

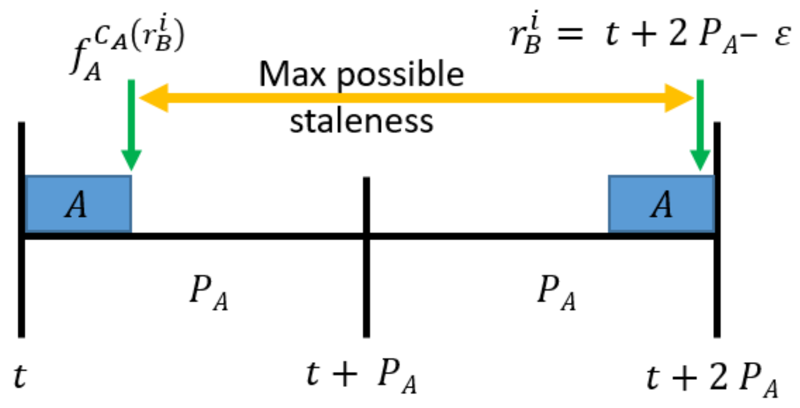

Figure 1: *Maximum Staleness Scenario for Data from Task A.*

Figure 1 labels the maximum separation between the publishing of output data from the depicted task. If we take $\epsilon \to 0$ 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 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, $d_{A\to B}$, is composed of two periods of A less one execution time of A less ϵ, i.e. $d_{A\to B} \leq 2P_A - E_A - \epsilon$. We can then solve for P_A to prove the following lemma.

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

Proof

$$
\begin{aligned}
d_{A\to B} &= 2P_A - E_A - \epsilon && \text{From Figure 1} \\
d_{A\to B} &= 2P_A - E_A && \epsilon \to 0 \\
2P_A &= d_{A\to B} + E_A && \text{Add } E_A \text{ and swap sides} \\
P_A &= \frac{d_{A\to B} + E_A}{2} && \text{Divide by 2} \\
P_A &\leq \frac{d_{A\to B} + E_A}{2} && \text{Lesser Periods Produce Fresher Output}
\end{aligned}
$$

■

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.

V. 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	$\forall i, r_C^i - f_A^{C_A(r_B^i)} \leq d_{A \to C}$

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

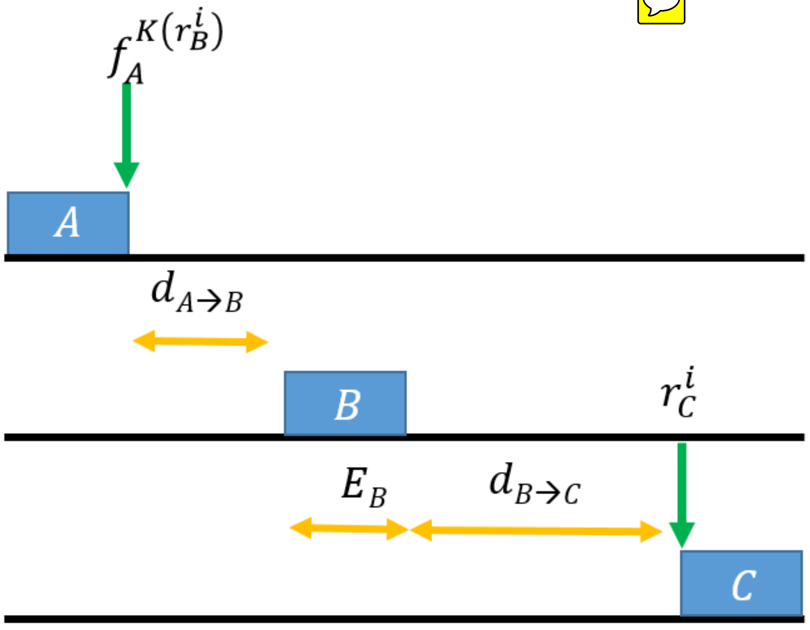

Figure 2: *Maximum Staleness Scenario for Data from Task A.*

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.

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,

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

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

$$P_A \leq \frac{d_{A\to B} + E_A}{2}$$

and

$$P_B \leq \frac{d_{B\to C} + E_B}{2}$$

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:

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

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*

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

and

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

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 $x + b + y = z$, we use te following:

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

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

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

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

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

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:

$$\begin{aligned}-\frac{2E_A}{(E_A + d_{A\to B})^2} &= -\frac{2E_B}{(E_B + d_{B\to C})^2} \\ d_{A\to B} + E_B + d_{B\to C} &= d_{A\to C}\end{aligned}$$

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

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

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

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

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

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

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:

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

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

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.

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.

VI. 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 tighter (likely longer) period bounds.

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.

VII. 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.