A Delay Composition Theorem for Real-Time Pipelines

Praveen Jayachandran and Tarek Abdelzaher
Department of Computer Science
University of Illinois
Urbana-Champaign, IL 61801
e-mail: pjayach2@uiuc.edu, zaher@uiuc.edu *

Abstract

Uniprocessor schedulability theory made great strides, in part, due to the simplicity of composing the delay of a job from the execution times of higher-priority jobs that preempt it. In this paper, we bound the end-to-end delay of a job in a multistage pipeline as a function of higher-priority job execution times on different stages. We show that the end-to-end delay is bounded by that of a single virtual “bottleneck” stage plus a small additive component. This contribution effectively transforms the pipeline into a single stage system. The wealth of schedulability analysis techniques derived for uniprocessors can then be applied to decide the schedulability of the pipeline. The transformation does not require imposing artificial per-stage deadlines, but rather models the pipeline as a whole and uses the end-to-end deadlines directly in the single-stage analysis. It also does not make assumptions on job arrival patterns or periodicity and thus can be applied to periodic and aperiodic tasks alike. We show through simulations that this approach outperforms previous pipeline schedulability tests except for very short pipelines or when deadlines are sufficiently large. The reason lies in the way we account for execution overlap among stages. We discuss how previous approaches account for overlap and point out interesting differences that lead to different performance advantages in different cases. We hope that the pipeline delay composition rule, derived in this paper, may be a step towards a general schedulability analysis foundation for large distributed systems.

1. Introduction

Understanding the end-to-end temporal behavior of distributed real-time systems is a fundamental concern of real-time computing. Execution pipelines (where tasks visit a sequence of machines for processing) are an especially common way to scale up real-time processing. Like multiprocessors, execution pipelines multiply the computing power of a single machine to increase real-time throughput. While a plethora of schedulability analysis techniques addressed multiprocessors, pipelines (as a fundamental processing construct) have received much less attention in the real-time community.

In this paper, we are concerned with pipelined systems that process several classes of real-time tasks, which traverse multiple stages of execution and must exit the system within specified end-to-end latency bounds. We derive a delay composition rule that allows the worst-case delay of a task invocation to be expressed in terms of the execution times of higher priority task invocations. According to this rule, the delay of a task in the pipeline has two components; (i) a job-additive component that is proportional to the sum of invocation execution times on a single stage (but is not proportional to the number of stages), and (ii) a stage-additive component that is proportional to the number of stages (but not the number of task invocations). Observe that this expression is better by a multiplicative factor than one that does not account for execution overlap (i.e., assumes that a task is preempted by all invocations of higher priority tasks on all stages). The delay in that last case is proportional to the product of the two components above as opposed to their sum. Consequently, our composition rule yields tight delay estimates that lead to good schedulability results.

Our composition rule does not make assumptions on the scheduling policy other than that it assigns the same priority to a task invocation at all stages. No assumption on periodicity of the task set is made. No assumption is made on whether different invocations of the same task have the same priority. Hence, this rule applies to static-priority scheduling (such as rate-monotonic), dynamic-priority scheduling (such as EDF) and aperiodic task scheduling alike. The simple expression of end-to-end delay computed by the aforementioned composition rule leads to a reduction of the multi-stage pipeline system to an equivalent single-stage system. Using this transformation, it becomes possible to use a wealth of existing schedulability analysis techniques on the new single-processor task set to analyze the original pipeline.

Prior work in analyzing the schedulability of real-time tasks in multistage systems can be broadly classified into two classes. The first class consists of offline schedulability tests that divide the end-to-end deadline into individual stage deadlines and analyze each stage independently [4, 9, 10, 16]. As we show later, these tests work well for periodic tasks with per-stage deadlines equal to periods. Holistic analysis [13] also performs well when end-to-end dead-

*The work reported in this paper was supported in part by the National Science Foundation under grants CCR-0093144, ANI-0105873, and EHS-0208769
lines are larger than periods. We show in our evaluation that the expressions derived in this paper outperform prior approaches when end-to-end deadlines are of the order of or smaller than periods. This includes aperiodic tasks (infinite periods). The second class of tests are exact tests that use response time analysis to precisely determine whether a task set is schedulable [11, 14, 5]. However, these tests generally have exponential or pseudo-polynomial running time complexity, which makes them less scalable to systems with large number of stages and large number of concurrent tasks.

With the growing complexity and scale of distributed systems, it can be argued that a designer will no longer seek complex optimal schedulability conditions that maximize the utilization of available resources. For practical reasons, designers of real-time systems will be more interested in simple sufficient conditions that are less error-prone, more scalable, and can be easily understood and applied by the common engineer. The schedulability conditions we derive in this paper fall into this category. In addition, as distributed systems become larger, it becomes critical that schedulability tests do not grow more pessimistic with system scale. We show that our test for pipeline schedulability outperforms previous tests by a factor that grows larger with pipeline length. For short pipelines, previous literature is adequate.

The remainder of this paper is organized as follows. Section 2 briefly describes the system model, states the main result, and outlines some intuitions into the delay composition theorem. In Section 3, this theorem is proved. Section 4 constructs a transformation of the pipeline into a single stage system accounting for execution overlap among stages in the original pipeline. Using this transformation, in Section 5, we illustrate how to use single stage schedulability analyses to analyze pipelines. In Section 6, we show results of simulation experiments that demonstrate how our new transformation outperforms previous schedulability analysis when end-to-end deadlines are small. Related work is reviewed in Section 7. We conclude in Section 8 with directions for future work.

2. System Model and Problem Statement

Consider a multi-stage distributed data processing pipeline. Periodic or aperiodic tasks arrive at this system and require execution on a set of resources (such as processors), each performing one stage of task execution. For the sake of deriving a general delay composition theorem, we consider individual task invocations in isolation, not to make any implicit periodicity assumptions. We call these invocations, jobs. In a given system, many different jobs may have the same priority (e.g., invocations of the same task in fixed-priority scheduling). However, there is typically a tie-breaking rule among such jobs (e.g., FIFO). Taking the tie-breaker into account, we can assume without loss of generality that each individual job has its own priority. This assumption will simplify the notations used in the derivations.

By definition of a pipeline, we assume that all the jobs require processing on all the stages and in the same order. Let the total number of stages be $N$. We number these stages from 1 to $N$, in the order visited by the jobs. Let $A_{i,j}$ be the arrival time of job $J_i$ at stage $j$, where $1 \leq j \leq N$. The arrival time of the job to the entire system, called $A_i$, is the same as its arrival to the first stage, $A_i = A_{i,1}$. Let $D_i$ be the end-to-end (relative) deadline of $J_i$. It denotes the maximum allowable latency for $J_i$ to complete its computation in the system. Hence, $J_i$ must exit the system by time $A_i + D_i$.

The computation time of $J_i$ at stage $j$, referred to as the stage execution time, is denoted by $C_{i,j}$, for $1 \leq j \leq N$. Finally, let $S_{i,j}$, called the stage start time, be the time at which $J_i$ starts executing on a stage $j$, and let $F_{i,j}$, called the stage finish time, be the time at which $J_i$ completes executing on stage $j$.

The main contribution of this paper lies in deriving a delay composition theorem to bound the delay experienced by any job as a function of the execution times of higher-priority jobs in the pipeline. Let the job whose delay is to be estimated be $J_1$, without loss of generality. Let $S$ denote the set of all higher-priority jobs that have execution intervals in the pipeline between $J_1$’s arrival and finish time ($S$ includes $J_1$). Also, let the quantities $C_{i,\text{max}1}$ and $C_{i,\text{max}2}$, for any job $J_i$, denote its largest and second largest stage execution time respectively. The delay composition theorem for $J_1$ is stated as follows:

**Pipeline Delay Composition Theorem.** Assuming a preemptive scheduling policy with the same priorities across all stages for each job, the end-to-end delay of a job $J_1$ in an $N$-stage pipeline can be composed from the execution parameters of jobs that preempt or delay it (denoted by set $S$) as follows:

$$\text{Delay}(J_1) \leq \sum_{i \in S} C_{eq_i} + \sum_{j=1}^{N-1} \max_{i \in S}(C_{i,j})$$

$$C_{eq_i} = \begin{cases} C_{i,\text{max}1} + C_{i,\text{max}2}, & \text{if } A_1 < A_i \\ C_{i,\text{max}1}, & \text{if } A_1 \geq A_i \end{cases}$$

Observe that, from the perspective of deriving the delay composition theorem, we are not concerned (for the moment) with how to determine set $S$. We are merely concerned with proving the fundamental property of delay composition over any such set. From the perspective of schedulability analysis, however, it is useful to estimate a worst case $S$ to compute worst-case delay. Trivially, in the worst case, $S$ would include all jobs $J_i$ whose active intervals $[A_i, A_i + D_i]$ overlap that of $J_1$ (i.e., overlap $[A_1, A_1 + D_1]$). This is true because a job $J_1$ whose deadline precedes the arrival of $J_i$ or whose arrival is after the deadline of $J_1$ has no execution time intervals between $J_1$’s arrival time and deadline (in a schedulable system), and hence cannot be part of $S$. The use of the delay composition theorem for schedulability analysis is further elaborated in Section 4.

---

1 While we equate a resource to a processor, the same discussion applies to other resources such as network links and disks as long as they are scheduled in priority order. A resource pipeline can thus contain heterogeneous resources that include processing, communication and disk I/O stages.
To appreciate the significance of the delay composition theorem let us consider a numeric example. Consider a set of two periodic tasks, $T_1$ and $T_2$, executing on a six-stage pipeline. Let the computation time of each task on each stage be the same and equal to 1. Let $T_1$ have a period of 9, equal to its end-to-end deadline. Let $T_2$ have a period of 6, also equal to its end-to-end deadline. We further assume that the first job (i.e., invocation) of each task arrives to the system at the same time. Is the task set schedulable? Assume that EDF is used on each stage.

A common way to solve this problem is to partition the end-to-end deadline of each task into per-stage deadlines then analyze the schedulability of each stage independently. In this example, since the load is equal on all stages, we divide the end-to-end deadlines equally among stages, leading to a per-stage deadline of 1.5 for $T_1$ and 1 for $T_2$. Note that, $T_2$ has zero slack on each stage. It runs first and meets its per-stage deadlines. However, $T_1$ needs up to two time units to complete on a stage, which is larger than its 1.5 per-stage deadline. For $T_1$ to be guaranteed in this six-stage system, the above analysis requires its end-to-end deadline to be at least $2 \times 6 = 12$.

Now, let us apply Equation (1) to calculate the delay of an invocation of $T_1$. Since, in this example, we know that $T_1$ can be preempted by at most 2 invocations of $T_2$, the set $S$, in Equation (1), contains only two invocations of $T_2$ along with the invocation of $T_1$ under consideration. Moreover, in any given period of $T_1$ only one of the two invocations of $T_2$ satisfies $A_1 < A_2$ (leading to $C_{eq_1} = 2$ for one invocation and 1 for the other). $C_{eq_1} = 1$. Hence, the first summation is equal to $2 + 1 + 1 = 4$. The second summation adds 5 leading to a total delay of 9 for $T_1$. This is lower than 12 above and does not exceed $T_1$’s end-to-end deadline. The system is found schedulable. In other words, our results can lead to less pessimistic pipeline schedulability analysis. The explanation is as follows.

The traditional analysis (i.e., breaking the end-to-end deadline into per-stage deadlines and performing a single stage schedulability test) is pessimistic because it assumes a worst-case arrival pattern. In other words, it assumes that an invocation of $T_1$ and $T_2$ arrive together, leading to a delay of 2 for $T_1$. In reality, this is not true of each stage. For example, if this arrival pattern was true at the first stage, $T_2$ would execute ahead of $T_1$ on that stage and move on to the next. From then on, $T_1$ would execute on each stage concurrently with the execution of $T_2$ on stage $n + 1$. $T_1$ would never wait for $T_2$ again, since every time $T_1$ would advance to the next stage, $T_2$ would leave it to the one after. It is important to account for this execution overlap. Indeed, if $T_1$ and $T_2$ start together, $T_1$ will take 2 time units on the first stage and one of each subsequent stage, finishing in only 7 time units.\(^2\) Clearly, need arises to better account for the effect of pipelining and execution overlap, which is what we purport to do in this paper.

The following question might then arise: is the common practice of partitioning end-to-end deadlines into per-stage deadlines always pessimistic? The answer is no. For example, consider a task set with per-stage deadlines equal to their periods. The set is schedulable using EDF at up to 100% utilization on each machine. There is no room for improvement in this case. The difference between this and the previous example lies in the ratio of task end-to-end deadlines to periods. In the current example, this ratio is equal to the number of stages. In the previous example this ratio was 1. While the results of this paper are general, they offer improvement over the state of the art only in the case where the ratio of end-to-end deadlines to periods of tasks is sufficiently smaller than the number of stages. In particular, the theory offers great improvements for aperiodic tasks (where periods are “infinite” and hence satisfy the above condition).

Finally, it is interesting to note that preemption in pipelines can reduce execution overlap among stages (which explains why $C_{eq_1}$, in the delay composition rule, depends on which job comes first). For example, consider the case of a two-job pipeline system shown in Figure 1. In Figure 1(i), the higher-priority job $J_i$ arrives together with $J_1$ and is given the (first-stage) CPU. When $J_1$ moves on to the second stage, $J_i$ can execute in parallel on the first. However, as shown in Figure 1(ii), if $J_i$ arrives after $J_1$, and preempts it, when $J_i$ moves on to the next stage, only the unfinished part of $J_1$ on the stage where it was preempted can overlap with $J_1$’s execution on the next stage. In other words, execution overlap is reduced and $J_i$ takes longer to finish than it did in the previous case. For instance, in our six-stage example, presented above, the aforementioned arrival scenario gives an actual delay of 8, not 7, for $T_1$.

With the intuitions explained above, we now prove the pipeline delay composition theorem. In the proof below, we consider individual jobs not tasks in order to be general. By considering jobs we do not restrict the results to the special case of periodic arrivals.

![Figure 1. Figure showing the possible cases of two jobs in the system.](image)

### 3. Delay Composition for Pipelined Systems

The delay composition theorem can be proved by induction on task priority. We first prove the theorem for a two-job scenario (Lemma 1). We then prove the induction step, where we assume that the delay composition theorem is true for $k - 1$ jobs, $k \geq 3$, add a $k^{th}$ job with highest priority, and prove that the delay composition theorem still holds.

**Lemma 1.** When $J_1$ and $J_2$ are the only two jobs in the system, and $J_2$ has a higher priority than $J_1$, the delay experienced by $J_1$ is at most

---

\(^2\)We show later that this is not the worst case scenario, but the system is indeed schedulable.
\[ Q = \sum_{i=1}^{2} C_{eqi} + \sum_{j=1}^{N-1} \max_{i=1,2}(C_{i,j}), \]

where:

\[ C_{eqi} = \begin{cases} C_{i,\text{max}1}, & \text{if } A_i \leq A_1 \\ C_{i,\text{max}1} + C_{i,\text{max}2}, & \text{if } A_i > A_1 \end{cases} \]

**Proof.** We shall prove the lemma by considering two cases; \(J_2\) arrived before (or together with) \(J_1\), and \(J_2\) arrived after \(J_1\) (special cases where one task arrives after the other exits the system can be trivially shown to satisfy the Lemma as well).

**Case 1:** \(J_2\) arrived before or together with \(J_1\) to the system

Since \(J_2\) is the highest-priority job in the system, it executes uninterrupted on all stages, completing each stage exactly after a time equal to \(C_{2,1} + \ldots + C_{2,k}\). Job \(J_1\) executes after \(J_2\) on the first stage. When \(J_1\) finishes some stage, it moves to the next, where it may encounter \(J_2\) (again) and must wait for it to finish. If \(J_2\) had already cleared that stage, \(J_1\) can execute there immediately. Let stage \(L\) be the last stage where \(J_1\) had to wait for \(J_2\). In this case, as shown in Figure 2-a, \(J_1\) completes the pipeline with a delay at most equal to:

\[ \text{Delay}(J_1) \leq C_{2,1} + \ldots + C_{2,L} + C_{1,L} + \ldots + C_{1,N} \quad (3) \]

**Figure 2. Figure showing the delay for the two cases of Lemma 1.**

Note that, \(C_{2,1} + \ldots + C_{2,L}\) takes us to the completion time of \(J_2\) on stage \(L\) (where \(J_1\) last waited for \(J_2\)). \(C_{1,L} + \ldots + C_{1,N}\) is the additional time taken by \(J_1\) to execute on \(L\) and the remaining stages. The delay expression in Inequality (3) has \(N+1\) terms, each representing a per-stage job computation time. There is exactly one per-stage computation in this expression from each stage, except stage \(L\) that contributes two. To compute a delay bound, let us replace one per-stage computation time at each of the first \(N-1\) stages by \(\max_{i=1,2}(C_{i,j})\) for that stage. Inequality 3 can now be rewritten as:

\[ \text{Delay}(J_1) \leq \sum_{j=1}^{N-1} \max_{i=1,2}(C_{i,j}) + C_{2,L} + C_{1,N} \quad (4) \]

Since the last two terms are smaller than \(C_{eq2}\) and \(C_{eq1}\), respectively, the expression in the lemma is a valid upper bound.

**Case 2:** \(J_2\) arrived after \(J_1\) to the system

Let \(J_2\) preempt \(J_1\) on some stage \(j\). Up to stage \(j-1\), the delay of \(J_1\) on each stage is simply its own execution time. At stage \(j\), \(J_2\) preempts \(J_1\) after the latter has executed for some time \(C_{1,j}^* < C_{1,j}\). As in the case above, \(J_1\) executes after \(J_2\) on subsequent stages. Let \(L\) be the last stage where \(J_1\) waits for \(J_2\). The delay of \(J_1\) is thus \(C_{1,1} + \ldots + C_{1,j}^* + C_{2,j} + \ldots + C_{2,L} + C_{1,L} + \ldots + C_{1,N}\), as shown in Figure 2-b. Following the same substitution as above, we can show that:

\[ \text{Delay}_n \leq \sum_{j=1}^{N-1} \max_{i=1,2}(C_{i,j}) + C_{2,j} + C_{2,L} + C_{1,N} \quad (5) \]

Since \(C_{2,j} + C_{2,L} \leq C_{eq2}\) and \(C_{1,N} \leq C_{eq1}\), the expression in the lemma is a valid upper bound in this case as well. This completes the proof of the lemma.

We shall now prove the general form of the pipeline delay composition theorem by induction on job priority.

**Pipeline Delay Composition Theorem.** Assuming a preemptive scheduling policy with the same priorities across all stages for each job, the end-to-end delay of a job \(J_1\) of lowest priority in a distributed pipeline with \(n-1\) higher priority jobs is at most

\[ \text{Delay}(J_1) \leq \sum_{i=1}^{n} C_{eqi} + \sum_{j=1}^{N-1} \max_{i=1}^{n}(C_{i,j}) \]

where \(C_{eqi}\) is as defined in Lemma 1.

**Proof.** Without loss of generality, we assume that a job \(J_i\) has a higher priority than a job \(J_k\), if \(i > k, i, k \leq n\). That is, \(J_n\) has the highest priority, and \(J_1\) has the least priority.

The basis step is the case where there are only two jobs in the system, \(J_1\) and \(J_2\). The delay composition theorem for two jobs is precisely Lemma 1.

Assume that the result is true for \(n = k-1\) jobs, \(k > 3\). That is,

\[ \text{Delay}_{k-1}(J_1) \leq \sum_{i=1}^{k-1} C_{eqi} + \sum_{j=1}^{N-1} \max_{i=1}^{k-1}(C_{i,j}) \]

We need to show the result when a \(k^{th}\) job \(J_k\), with highest priority, is added. Let \(L_k\) be a pipelined system with \(k\) jobs, with arbitrary arrival times for each of the jobs. Let \(L_{k-1}\) be the system without job \(J_k\). The outline of the proof is similar to the proof of Lemma 1. We consider two cases, \(J_k\) arrived before (or together with) \(J_1\) to the system, and \(J_k\) arrived after \(J_1\) to the system.

**Case 1:** \(J_k\) arrived before or together with \(J_1\) to system \(L_k\).

Notice that, if there exists an idle time between the execution of \(J_k\) and \(J_1\) on some stage \(j\), the delay of \(J_1\) on stage
delay in system \( L_{k-1} \) starting from stage \( j \). As we make no assumption on the arrival pattern of higher priority jobs, the delay composition theorem provides the worst case delay for any possible arrival pattern of jobs. Although, adding job \( J_k \) does perturb the schedule, the worst case delay due to jobs \( J_2 \) through \( J_{k-1} \) as per the delay composition theorem accounts for any arrival pattern of \( J_2 \) through \( J_{k-1} \). We can therefore apply induction assumption starting from stage \( j \). Therefore, the delay of \( J_1 \) can be expressed as the delay up to the time \( J_k \) completes execution on stage \( j \), added to the worst case delay of \( J_1 \) in system \( L_{k-1} \) starting from stage \( j \) (as shown in Equation 7). This is shown in Figure 3.

\[
\text{Delay}_k(J_1) = F_{1,N} - A_{1,1} = (F_{1,N} - F_{k,j}) + (F_{k,j} - A_{1,1}) \quad (7)
\]

As \( J_k \) arrived before \( J_1 \) to the system, the duration between the arrival of \( J_1 \) to the system \((A_{1,1})\) and the completion of \( J_k \)'s execution on stage \( j \) \((F_{k,j})\), is at most the time \( J_k \) takes to complete execution up to stage \( j \) \((F_{k,j} - A_{k,1})\) (Inequality 8). \( J_k \) is the highest priority job in the system, and does not wait to execute on any of the stages. The time for \( J_k \) to complete execution up to stage \( j \) is \((\sum_{t=1}^{j-1} C_{k,t} + C_{k,j})\). In addition to this, from induction assumption, the delay of \( J_1 \) from stages \( j \) through \( N \) is \( \sum_{t=1}^{k-1} C_{eq_t} + \sum_{t=j}^{N-1} \max_{i \leq k-1} (C_{i,t}) \) (Inequality 9). Thus,\n
\[
\sum_{t=1}^{k-1} C_{eq_t} + \sum_{t=j}^{N-1} \max_{i \leq k-1} (C_{i,t}) \leq \sum_{t=1}^{N-1} \max_{i \leq k-1} (C_{i,t}) \leq \sum_{t=1}^{N-1} \max_{i \leq k} (C_{i,t})
\]

which proves the delay composition theorem.

**Case 2:** \( J_k \) arrived after \( J_1 \) to the system.

Until the time \( J_k \) preempts \( J_1 \), the delay of \( J_1 \) is independent of \( J_k \). Let \( J_k \) preempt \( J_1 \) at stage \( j \). Beyond stage \( j \), \( J_k \) arrives at each stage before \( J_1 \). Therefore, the pipeline beyond stage \( j \) can be thought of as one having \( N - j \) stages, and \( J_k \) arriving before \( J_1 \). We can then apply the result from case 1.

The fact that \( J_k \) preempted some job at stage \( j \) (it is possible that \( J_k \) preempted some job, which in turn had preempted \( J_1 \)), implies that there was a job executing when \( J_k \) arrived at stage \( j \). Further, there is no idle time between the executions of \( J_k \) and \( J_1 \). Let \( J_{i_1}, J_{i_2}, \ldots, J_{i_s} \), be the jobs that execute between \( J_k \) and \( J_1 \) on stage \( j \) (Figure 4). \( J_{i_1} \) is delayed by \( J_k \) up to stage \( j \) by at most \( C_{k,j} \). Similarly, irrespective of previous stages, each of \( J_{i_2}, J_{i_3}, \ldots, J_{i_s} \), and \( J_1 \) are delayed by an amount \( C_{k,j} \) due to \( J_k \) up to stage \( j \).

**Figure 4.** Figure showing the case when \( J_k \) arrived after \( J_1 \) and preempts \( J_1 \) at stage \( j \).

Beyond stage \( j \), as mentioned earlier the system is identical to case 1 (as \( J_k \) arrived before \( J_1 \) to stage \( j+1 \)). From the result of case 1, the additional delay that \( J_k \) causes \( J_1 \) is one maximum stage execution time between stages \( j \) and \( j+1 \) through \( N \), apart from \( J_k \)'s contribution to the stage-additive component \( \max_{t \leq j} (C_{i,t}) \), for \( j+1 \leq t \leq N-1 \) (from Inequality 9). Figure 4 shows this scenario. We showed that the delay due to \( J_k \) up to stage \( j \) is at most \( C_{k,j} \). Therefore, the total job-additive delay to \( J_1 \) due to \( J_k \) is at most the sum of the two maximum stage execution times of \( J_k \), that is \( C_{k,\text{max}1} + C_{k,\text{max}2} = C_{eq_k} \).

This proves the induction step. Using this together with Lemma 1, the delay composition theorem is proved.

### 4. Schedulability and Pipeline Reduction

In this section, we illustrate a systematic reduction of the pipeline schedulability problem to an equivalent single stage problem using the delay composition theorem. Since delay predicted by the delay composition theorem grows with set \( S \), let us first define the worst-case (i.e., largest) set \( S \), denoted \( S_{\text{wc}} \), of higher priority jobs that delay or preempt \( J_1 \). In this paper, we suggest a very simple (and somewhat conservative) definition of set \( S_{\text{wc}} \). We expect that future work can improve upon this definition using more in-depth analysis. In the absence of further information, set \( S_{\text{wc}} \) is
Theorem: The worst-case set $S_{wc}$ of higher priority jobs that delay or preempt job $J_1$ (hence, include execution intervals between the arrival and finish time of $J_1$) includes all jobs $J_i$ whose intervals $[A_i, A_i + D_i]$ overlap the interval where $J_1$ was present in the pipeline, $[A_1, A_1 + delay(J_1)]$.

Observe that the above is a conservative definition. It simply excludes the impossible. In a schedulable system, a job $J_i$ that does not satisfy the above condition either completes prior to the the arrival of $J_1$ or arrives after its completion. Hence, it cannot possibly have execution intervals that delay or preempt $J_1$.

Let us divide the set $S_{wc}$ into the subset $S_{bef} \subset S_{wc}$ that contains those jobs with $A_i \leq A_1$, and a subset $S_{after} \subset S_{wc}$ that contains those jobs with $A_i > A_1$. We can now rewrite the delay composition theorem, separating its first summation into two: one for invocations that arrive before (or with) $T_1$, and one for those that arrive after. This allows us to substitute for $C_{eq}$, accordingly in each summation, resulting in the following:

\[
 delay(J_1) \leq \sum_{i \in S_{bef}} C_{i, \text{max}1} + \sum_{i \in S_{after}} (C_{i, \text{max}1} + C_{i, \text{max}2}) \]

\[
 + \sum_{j=1}^{N-1} \max(C_{i,j}) \quad (11)\]

The reduction to a single stage system is then conducted by (i) replacing each pipeline job $J_i$ in $S_{bef}$ by an equivalent single stage job (with the same priority and deadline) of execution time equal to $C_{i, \text{max}1}$, (ii) replacing each pipeline job $J_i$ in $S_{after}$ by an equivalent single stage job of execution time equal to $C_{i, \text{max}1} + C_{i, \text{max}2}$, and (iii) adding a lowest-priority job, $J_e^*$ of execution time equal to $\sum_{j=1}^{N-1} \max(C_{i,j})$ (which is the last term in Inequality (11)), and deadline same as that of $J_1$. By the delay composition theorem, the total delay incurred by $J_1$ in the pipeline is no larger than the delay of $J_e^*$ on the uniprocessor, since the latter adds up to the delay bound expressed on the right hand of Inequality (11).

For example, let us illustrate this transformation in the case of rate-monotonic scheduling of periodic tasks with periods equal to end-to-end deadlines. Consider a set of periodic tasks arriving at a pipeline, where each task $T_i$ has a period $P_i$. As shown in Figure 5, there can be at most one invocation of each higher-priority task $T_i$ in set $S_{bef}$. Similarly, the number of invocations of each task $T_i$ that arrive after the invocation of $T_1$ and delay it, is no larger than \( \frac{\text{delay}(T_1)}{P_1} \). Following the reduction outlined above, then aggregating jobs of the same period into single periodic tasks, the following periodic task set is reached:

- Task $T_e^*$ (of lowest priority), with a computation time $C_e^* = \sum_i C_{i, \text{max}1} + \sum_{j=1}^{N-1} \max_i(C_{i,j})$. The task further has the same period and deadline as $T_n$ in the original set.
- Tasks $T_i^*$, each has the same period and deadline as one

For the worst-case set $S_{wc}$ of higher priority jobs that delay or preempt job $J_1$ (hence, include execution intervals between the arrival and finish time of $J_1$) includes all jobs $J_i$ whose intervals $[A_i, A_i + D_i]$ overlap the interval where $J_1$ was present in the pipeline, $[A_1, A_1 + delay(J_1)]$.

Observe that the above is a conservative definition. It simply excludes the impossible. In a schedulable system, a job $J_i$ that does not satisfy the above condition either completes prior to the the arrival of $J_1$ or arrives after its completion. Hence, it cannot possibly have execution intervals that delay or preempt $J_1$.

Let us divide the set $S_{wc}$ into the subset $S_{bef} \subset S_{wc}$ that contains those jobs with $A_i \leq A_1$, and a subset $S_{after} \subset S_{wc}$ that contains those jobs with $A_i > A_1$. We can now rewrite the delay composition theorem, separating its first summation into two: one for invocations that arrive before (or with) $T_1$, and one for those that arrive after. This allows us to substitute for $C_{eq}$, accordingly in each summation, resulting in the following:

\[
 delay(J_1) \leq \sum_{i \in S_{bef}} C_{i, \text{max}1} + \sum_{i \in S_{after}} (C_{i, \text{max}1} + C_{i, \text{max}2}) \]

\[
 + \sum_{j=1}^{N-1} \max(C_{i,j}) \quad (11)\]

The reduction to a single stage system is then conducted by (i) replacing each pipeline job $J_i$ in $S_{bef}$ by an equivalent single stage job (with the same priority and deadline) of execution time equal to $C_{i, \text{max}1}$, (ii) replacing each pipeline job $J_i$ in $S_{after}$ by an equivalent single stage job of execution time equal to $C_{i, \text{max}1} + C_{i, \text{max}2}$, and (iii) adding a lowest-priority job, $J_e^*$ of execution time equal to $\sum_{j=1}^{N-1} \max(C_{i,j})$ (which is the last term in Inequality (11)), and deadline same as that of $J_1$. By the delay composition theorem, the total delay incurred by $J_1$ in the pipeline is no larger than the delay of $J_e^*$ on the uniprocessor, since the latter adds up to the delay bound expressed on the right hand of Inequality (11).

For example, let us illustrate this transformation in the case of rate-monotonic scheduling of periodic tasks with periods equal to end-to-end deadlines. Consider a set of periodic tasks arriving at a pipeline, where each task $T_i$ has a period $P_i$. As shown in Figure 5, there can be at most one invocation of each higher-priority task $T_i$ in set $S_{bef}$. Similarly, the number of invocations of each task $T_i$ that arrive after the invocation of $T_1$ and delay it, is no larger than \( \frac{\text{delay}(T_1)}{P_1} \). Following the reduction outlined above, then aggregating jobs of the same period into single periodic tasks, the following periodic task set is reached:

- Task $T_e^*$ (of lowest priority), with a computation time $C_e^* = \sum_i C_{i, \text{max}1} + \sum_{j=1}^{N-1} \max_i(C_{i,j})$. The task further has the same period and deadline as $T_n$ in the original set.
- Tasks $T_i^*$, each has the same period and deadline as one
\[ R_i^{(0)} = C_i^*(i) \]
\[ R_i^{(k)} = C_i^*(i) + \sum_{j<i} \left( \frac{R_j^{(k-1)}}{P_j} \right) C_j^* \]

The worst case response time for task \( T_i \) is given by the value of \( R_i^{(k)} \), such that \( R_i^{(k)} = R_i^{(k-1)} \). For the task set to be schedulable, for each task \( T_i \), the worst case response time needs to be at most \( D_i \). During implementation, the following optimization can be performed. Notice that, each time a job of a higher priority task causes a delay equal to two single stage execution times (accounted for in \( C_j^* \)), the lower priority job progresses by at least one stage in the pipeline. Therefore, the number of jobs of a higher priority task that can cause a delay equal to two single stage execution times is at most equal to the number of stages, \( N \). If \( R_i^{(k-1)}/P_j > N \), then \( N \) jobs contribute two single stage execution times, whereas the interference of all other jobs is simply one stage execution time. This optimization has been performed in our simulations.

6. Simulation Results

To evaluate the actual performance of our delay composition rule and reduction to a single stage system, we constructed a simulator that models a distributed pipelined system. In order to maintain real-time guarantees within the system, an admission controller is used. For periodic tasks, the admission controller is based on a single stage schedulability test for deadline monotonic scheduling, such as the Liu and Layland bound [8], the hyperbolic test [3], or response time analysis [2], together with our reduction of the multistage system to a single stage, as shown in Section 5. When a task arrives at the system, it is tentatively added to the set of tasks in the system. The admission controller then tests whether the new task set is schedulable. The new task is admitted if the task set is schedulable, and dropped if not.

Due to space limitations, we only present results for deadline monotonic scheduling. In the rest of this section, we use the term utilization to refer to the average per-stage utilization. Each point in the figures below represent average values obtained from 100 executions of the simulator, with each execution running for 30000 task invocations. End-to-end deadlines (equal to the periods) of tasks are chosen as \( 10^{10} \) simulation seconds, where \( x \) is uniformly varying between 0 and \( DR \) (deadline ratio parameter), and \( a = 500 + N \), where \( N \) is the number of stages in the system. Such a choice of deadlines enables the ratio of the longest task deadline to the shortest task deadline to be as large as \( 10^{DR} \). If \( DR \) is chosen close to zero, tasks would have similar deadlines. If \( DR \) is higher (for example \( DR = 3 \)), deadlines of tasks would differ more widely. As will be demonstrated later in this section, we observed from our simulations that the achievable utilization varied significantly with the choice of \( DR \). The default value for \( DR \) is taken to be 1. The execution time for each task on each stage was chosen based on the task resolution parameter, which is a measure of the ratio of the total computation time of a task over all stages to its deadline. The stage execution time of a task is calculated based on a uniform distribution with mean equal to \( DT/N \), where \( D \) is the deadline of the task and \( T \) is the task resolution. Task preemptions are assumed to be instantaneous, that is, the task switching time is zero. Load is defined as the sum of computation times of all tasks that arrive during the simulation divided by the duration of the experiment. Unless otherwise specified, we use the following default values - system load of 100\%, task resolution of 1 : 100, and 5 pipeline stages. The 95\% confidence interval for all the utilization values presented in this section is within 0.004 of the mean value, which is not plotted for the sake of legibility.

![Figure 6. Comparison of meta-schedulability test with the aperiodic pipeline bound](image)

We first consider the case of aperiodic tasks. Below, we refer to our new process of testing an “equivalent” single-stage system a meta-schedulability test. Recall that, in this approach, the entire pipeline is transformed into one single-stage system that takes the whole pipeline into account and is subjected to the original end-to-end deadlines. This is in contrast, for example, to approaches that partition end-to-end deadlines into per-stage deadlines then apply uniprocessor analysis to each stage independently.

For aperiodic tasks, we transform the pipeline into a single stage, then use, in the meta-schedulability test, the uniprocessor aperiodic utilization bound derived in [1]. We compare it with the pipeline bound presented in [6], which is based on the same aperiodic task bound. For both these tests, while keeping other simulation parameters constant, we varied the number of pipeline stages and measured the utilization, the results of which are shown in Figure 6. The average per-stage utilization of the aperiodic pipeline bound presented in [6] decreases linearly with the number of pipeline stages, as it does not account for the overlap in the execution of different pipeline stages. Our meta-schedulability test is able to achieve nearly the same utilization regardless of the number of pipeline stages. For the rest of this section, we shall concern ourselves only with periodic tasks.

We first compare our meta-schedulability test with holistic analysis [13], and two implementations of traditional pipeline schedulability tests, which divide the end-to-end deadline into equal individual single stage deadlines. The
first implementation, which we call ‘traditional’, tests for each stage if the sum of the ratios of computation times to per-stage deadlines over all tasks is less than the Liu and Layland bound for periodic tasks. Since this bound is pessimistic when per-stage deadlines are less than periods, our second implementation, which we call ‘traditional using RTA’, uses response time analysis based on deadline monotonic scheduling to analyze the schedulability of each stage. In this analysis, if the response times on every stage for all tasks are found to be less than their respective per-stage deadlines, then the task set is declared to be schedulable. As explained in the example in Section 2, tests that partition end-to-end deadlines to per-stage deadlines (and use single-stage analysis independently on each stage) may be pessimistic because they assume a worst-case arrival pattern at each stage. Holistic analysis avoids this problem. In holistic analysis, the response time on one stage is considered as the jitter for the next change. The analysis does not divide the end-to-end deadline into single stage deadlines. Nevertheless, by considering the previous stage response time as the jitter, it considers possible that a job is delayed by the same higher priority job on every stage of the pipeline. We compare the above approaches to the performance of our meta-schedulability test. In our test, we use both the Liu and Layland bound and response time analysis on the resulting single stage system.

We conducted experiments to measure the average per-stage utilization for different number of pipeline stages, when using admission controllers based on each of these five tests. In these experiments, task periods were set equal to their end-to-end deadlines. Figure 7 plots this comparison, for a DR value of 0.5. We observe that the utilization for both the traditional pipeline tests decreases proportionally with the number of stages in the pipeline system. As expected, response-time analysis performs better and holistic analysis outperforms both traditional tests. Its utilization nevertheless decreases with increasing number of pipeline stages. In contrast, our meta-schedulability test sustains nearly the same utilization, regardless of the number of pipeline stages. In other words, the pessimism in declaring task sets schedulable is not dependent on the number of pipeline stages. This property is a result of our delay composition rule.

Notice that the utilization slightly decreases with increasing number of stages. This is due to the stage-additive component of the delay, which is the sum of the maximum execution times of any higher priority task, over all pipeline stages. Clearly, the stage-additive component increases with the number of stages, and hence the utilization slightly decreases with increasing number of stages.

We further compared the utilization achieved by our meta-schedulability test based on RTA with holistic analysis, for two different deadline ratio parameters and for different number of pipeline stages. Figure 8 plots this comparison. For both analysis techniques, trends similar to those in Figure 7 are observed. However, as the deadline ratio parameter increases, the achievable per-stage utilization significantly increases. For high deadline ratio parameter values, the deadlines of lower priority tasks are very large compared to those of higher priority tasks (when $DR = 3$, the deadline ratio of the highest to the lowest priority task can be as high as 1000). At most times, some of these lower priority tasks exist in the system and can execute in the background, thereby providing high processor utilization. This figure helps to suggest in some sense, that the worst case situation in terms of reducing the achievable processor utilization, occurs when all tasks have very similar deadlines and stage execution times. Further, the values specified as ‘simulation’ were the lowest utilization values at which deadline misses were observed in the absence of any admission controller (for the same task parameters). This serves to indicate an upper bound on the achievable utilization.

A criticism of the above results is that they favor our tests by setting end-to-end deadlines equal to periods. As mentioned in Section 2, traditional tests that partition end-to-end deadlines work very well as long as deadlines are large compared to periods. In order to characterize the break even point after which our meta-schedulability test outperforms traditional schedulability analysis techniques, we compared the achievable utilization for different values of the ratio between the end-to-end deadline and the task period, while maintaining the system load constant (by proportionately changing the execution times of tasks). Response time analysis was
used as the single-stage schedulability test for both the techniques. Figures 9 and 10 plot this comparison for 5 and 8 pipeline stages, respectively. Note that a ratio of 5 for 5 stages, and 8 for 8 stages indicate that the period is equal to the per-stage deadline (for traditional schedulability analysis). When the ratio of the end-to-end deadline to period is higher, the laxity available to jobs is larger, and hence, the utilizations of both techniques are high. For higher values of the period, the meta-schedulability test outperforms the traditional test, while at lower values of the period, the traditional test performs better. The cross-over point, the value of the period where the meta-schedulability test outperforms the traditional test, is lower for 8 stages than for 5 stages, showing the pessimism of the traditional test with increasing number of pipeline stages.

### 7. Related Work

The first study of feasible regions in real-time systems was conducted by Liu and Layland [8]. They studied a set of real-time systems with specific restrictions and presented utilization bounds. In [3], the utilization bounds were extended to multiprocessor real-time systems. Moreover, it considered resource constraints and presented a single-stage utilization bound that was less pessimistic than Liu and Layland’s.

Several scheduling algorithms have been proposed for statically scheduling precedence constrained tasks in distributed systems [11, 14, 5]. Given a set of periodic tasks, such algorithms attempt to construct a schedule of length equal to the least common multiple of the task periods. The schedule will accurately specify the time intervals during which each task invocation will be executed. Needless to say, such algorithms have a large time complexity and are clearly unsuitable for complex, large scale distributed systems, where simplicity is of essence.

Analyzing the Worst Case Execution Times (WCET) of tasks in processor and memory pipeline architectures is a well studied problem in the area of real-time operating systems ([15, 12] and references thereof). Such algorithms execute in time that is exponential in the number of tasks in the system. Further, the approach would be difficult to implement in a distributed setting and is more error-prone.

A few offline schedulability tests have also been proposed for pipelined distributed systems. A distributed pipeline framework was presented in [4], where a complex, heterogeneous, multi-resource system is decomposed into a set of single resource scheduling problems. Each single resource scheduling problem corresponds to a stage in the multistage pipelined distributed system. Offset-based response time analysis techniques for distributed systems scheduled using EDF were proposed in [9, 10] which divide the end-to-end deadline into individual stage deadlines. Recently, [16] designed and implemented a middleware layer based on deferrable servers for aperiodic tasks with hard end-to-end deadlines in distributed real-time applications. Techniques to divide the end-to-end deadline into sub-deadlines for individual stages were presented. Holistic schedulability analysis for distributed hard real-time systems was first proposed in [13]. This technique does not divide the end-to-end delay into sub-deadlines for individual stages. Instead, the worst case delay at a stage is taken as the jitter for the next stage. Holistic analysis was extended to EDF in [7].

A schedulability test based on aperiodic scheduling theory was derived in [6], for fixed priority scheduling. Although this solution handles arbitrary-topology resource systems and resource blocking, it does not consider the overlap in the execution of multiple stages in the pipeline, which is a fundamental cause of pessimism. We account for this overlap in our pipeline delay composition theorem, and reduce the schedulability analysis of a multistage pipeline system to that of single stage systems. This largely increases schedulability, and the performance of the system does not become poorer with increasing number of pipeline stages.

### 8. Conclusions and Future Work

This paper presents a delay composition rule for pipelined systems. The rule demonstrates that the execution times of higher priority jobs compose sub-additively, rather than the implicit additive delay composition rule for uniprocessor systems. This composition rule leads to a reduction of the pipelined system to a single-stage system. Based on this
reduction, we define a meta-schedulability test, a test that uses another single stage schedulability test to analyze the schedulability of real-time tasks in pipelined systems. It outperforms previous pipeline schedulability tests for aperiodic tasks and does better than periodic pipeline schedulability approaches when end-to-end deadlines are sufficiently short.

This work opens the door for schedulability theory research in distributed systems in multiple directions. For example, there is no evidence that the proposed composition rule is optimal. More efficient composition rules, if identified, can help reduce the pessimism further.

Our pipeline delay composition rule, in itself, is independent of schedulability metrics such as utilization. It can therefore be used to identify other metrics that would enable more efficient schedulability analysis for distributed systems. While it is well-known that the deadline monotonic algorithm is the optimal fixed-priority scheduling algorithm for single stage systems, the optimal scheduling policy for distributed systems is still an open problem. The delay composition rule can be used to gain further insights such fundamental open research issues.

Extending the delay composition theorem and meta-schedulability test to non-preemptive scheduling and resource blocking can help widen the applicability of the result. Finally, the current work addresses only pipelined distributed systems. Analyzing distributed systems with different precedence graphs (possibly arbitrary) is a worthwhile extension.

References