A Delay Composition Theorem for Real-Time Distributed Directed Acyclic Systems

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

In this paper, we present a delay composition rule that bounds the worst-case end-to-end delay of a job as a function of per-stage execution times of higher priority jobs along its path, in a multistage distributed system where the routes of jobs form a directed acyclic graph. The delay composition rule makes no assumption on scheduling policy (except that jobs are assigned the same priority on all stages), and makes no assumption on periodicity. Applying the rule to a particular job only requires knowledge of execution times of higher priority jobs along the path followed by the job, which is in contrast with traditional schedulability analysis techniques that require global knowledge of all jobs and routes in the distributed system, which may be difficult or expensive to obtain. Inspired by the resulting simple delay expression of our composition rule, a transformation of the system to an equivalent single stage system becomes apparent. The wealth of schedulability analysis techniques derived for uniprocessors can then be applied to decide schedulability of tasks in a DAG. We compare our analysis technique with traditional techniques using simulations.

1. Introduction

Several real-time and embedded systems today are being implemented as distributed systems, and understanding the end-to-end temporal behavior of such systems is a fundamental concern of real-time computing. While a plethora of schedulability analysis techniques addressed multiprocessors, there is a distinct lack of theoretical tools to conduct such analysis on distributed systems.

In this paper, we are concerned with distributed systems that process several classes of real-time tasks, whose execution paths form a Directed Acyclic Graph (DAG). Each task traverses through multiple stages of execution and must exit the system within specified end-to-end latency bounds. Preemptive fixed priority scheduling is assumed at each node of the DAG. 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 system has two components; (i) a *job-additive* component that is proportional to the amount of 'cross' traffic, that is, traffic due to tasks that merge with the task under consideration at different stages of its execution, and (ii) a *stage-additive* component that is proportional to the number of stages (but not the number of task invocations). The main advantage of the delay composition rule is that it provides a bound on the worst case end-to-end delay of a task using only information of execution times of higher priority tasks along its route. In contrast, traditional schedulability analysis, such as holistic analysis [13], require global knowledge of task routes and execution times, in order to predict the worst case end-to-end delay. With the growing size of distributed and embedded systems, such global knowledge could be difficult or expensive to obtain.

The key factor that allows the delay composition rule to provide end-to-end delay bounds based on purely local information, is its distinction of cross traffic from pipelined traffic. In the presence of pipelined traffic, the overlap in the execution of successive stages allows for a much tighter bound on the end-to-end delay. However, with cross traffic, in the absence of global task arrival knowledge, the delay composition rule assumes that the cross traffic could arrive in a manner so as to cause maximum delay on lower priority task invocations. This assumption does cause the delay composition rule to be pessimistic in its end-to-end delay estimate, but such an estimate might be acceptable for large distributed systems, where traditional schedulability analysis is not feasible.

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 assump-

tion 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 distributed 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 distributed system.

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]. Such tests require global knowledge of the distributed system and tend to ignore the overlap in the execution of successive stages for pipelined traffic. This causes such techniques to become more pessimistic for large systems. Although holistic analysis [13] and aperiodic pipeline schedulability analysis [6] do not divide the end-to-end deadline into individual stage deadlines, they nevertheless do not account for the overlap in the execution of tasks in different stages. Further, holistic analysis requires global knowledge of the distributed system and has exponential running time complexity. 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 also have exponential or pseudo-polynomial running time complexity, which makes them less scalable to systems with a large number of stages and a 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.

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 DAG into a single stage system. Using this transformation, in Section 5, we illustrate how to use single stage schedulability analyses to analyze DAGs. In Section 6, we show results of simulation experiments comparing our analysis technique with previous approaches. 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 system. Periodic or aperiodic tasks arrive at this system and require execution on a set of resources (such as processors[1]), each performing one stage of task execution. Specifically, we consider a multi-stage distributed data processing system, whose topology can be expressed as a Directed Acyclic Graph (DAG). An edge in the DAG between stage i and stage j, indicates that a task that completes execution on stage i, could move on to execute at stage j. 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*. We assume that the priority of each job is the same across all the stages at which it executes. 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.

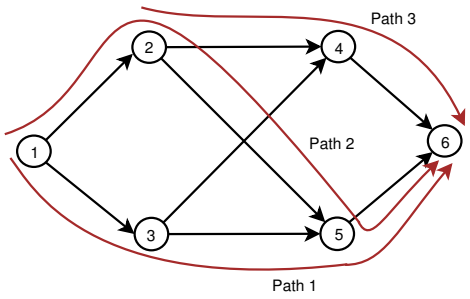

Figure 1. Example directed acyclic graph with job-flow paths, and a feasible number assignment for stages

A job can enter the system at any stage, request processing on a sequence of stages (a path in the DAG), and leave the system at any stage. Figure 1 shows an example of a directed acyclic distributed processing system along with a few sample job-flow paths. Let the total number of stages be $N^{'}$. We number these stages from 1 to $N^{'}$, in the order visited by the jobs. In other words, if there exists an edge in the DAG between stages i and j, then $i < j$. Such a numbering always exists as there are no cycles in the topology. However, this numbering may not be unique, and we choose any

[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 distributed system can thus contain heterogeneous resources that include processing, communication and disk I/O stages.

one number assignment that satisfies the above mentioned condition. Let $Path_i$ denote the set of stages comprising the path chosen by job J_i in the system. 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 its first stage, $A_i = A_{i,t}$, where t is the smallest numbered stage in $Path_i$. 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 $1 \leq j \leq N^{'}$. If a job J_i does not execute at stage j, that is $j \notin Path_i$, then $C_{i,j}$ is zero. 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.

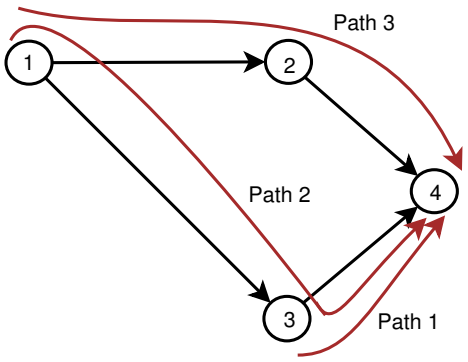

Figure 2. A sub-DAG chosen from the larger DAG, assuming J_1 follows path 3; stages are re-numbered

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 that have a common execution stage with this job. Let the job whose delay is to be estimated be J_1, without loss of generality. As we are interested in the delay of J_1, we need to only consider those stages that may potentially influence the delay of J_1. We consider a stage i, only if stage i is reachable in the DAG from the first stage at which J_1 executes, and the last stage at which J_1 executes is reachable from stage i. We remove all stages that do not satisfy this condition, and renumber the stages from 1 through N ($N \leq N'$). By the above definition, stage 1 is the first stage at which J_1 executes, and stage N is the last stage at which J_1 executes. As before, it is ensured that if there exists an edge between stage i and stage j, then $i < j$. Note that considering only a subset of the stages constructs a sub-graph of the original DAG, and is therefore still a DAG. The job-flow path of each job J_i is accordingly truncated, to only consider the sub-path belonging to the chosen sub-DAG. Figure 2 shows such a sub-DAG, assuming J_1 follows path 3 of the DAG shown in Figure 1. Let S denote the set of all higher-priority jobs that have execution intervals in the system between J_1's arrival and finish time, and have some common execution stage with J_1 (S includes J_1). Jobs that do not have a common execution stage with J_1 do not affect the delay of J_1. Let $C_{i,max}$, for any job J_i, denote its largest stage execution time, on stages where both J_i and J_1 execute.

We define a *split-merge* between the paths of jobs J_i and J_1, as a scenario where the path of J_i splits from the path of J_1, and intersects (merges with) the path of J_1 at a later stage. In more concrete terms, if there exists consecutive stages $j_1, j_2, \ldots, j_k (k \geq 2)$ in the path of J_1, and of these stages only j_1 and j_k belong to the path of J_i, and there is at least one other stage j' ($j_1 < j' < j_k$) on which J_i executes, then a split-merge is said to exist between J_i and J_1. Figure 2 shows a split-merge between paths 2 and 3. The total number of split-merges between the paths of J_i and J_1 is denoted by $SM_{i,1}$.

The delay composition theorem for J_1 is stated as follows:

DAG 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 DAG can be composed from the execution parameters of jobs that preempt or delay it (denoted by set S) as follows:*

$$Delay(J_1) \leq \sum_{i \in S} 2C_{i,max}(1+SM_{i,1}) + \sum_{\substack{j \in Path_1 \\ j \leq N-1}} \max_{i \in S}(C_{i,j}) \quad (1)$$

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_i whose deadline precedes the arrival of J_1 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.

Finally, it is interesting to note that preemption can reduce execution overlap among stages. For example, consider the case of a two-job system, where both jobs execute

on all stages, shown in Figure 3. In Figure 3(i), the higher-priority job J_i arrives together with J_1 and is given the (first-stage) CPU. When J_i moves on to the second stage, J_1 can execute in parallel on the first. However, as shown in Figure 3(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_i's execution on the next stage. In other words, execution overlap is reduced and J_1 takes longer to finish than it did in the previous case. Hence, in deriving the delay composition theorem, we shall assume that whenever the path of J_i intersects the path of J_1, J_i preempts J_1 (rather than arriving together with or prior to J_1) in such a manner so as to cause maximum delay. With the intuitions explained above, we now prove the delay composition theorem for directed acyclic graphs.

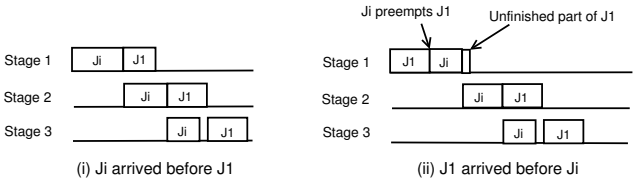

Figure 3. Figure showing the possible cases of two jobs in the system.

3. Delay Composition for Directed Acyclic Systems

Before we proceed to prove the delay composition theorem, we first prove a simple helper lemma.

Lemma 1. *The number of times a higher priority job J_i preempts J_1 is at most one more than the number of split-merges between the paths of J_i and J_1 ($SM_{i,1}$).*

Proof. The proof is by a simple induction on the number of split-merges between the paths of J_i and J_1. The basis step is when there are no split-merges, $SM_{i,1} = 0$. In this case, J_i can preempt J_1 at most once, as after J_i preempts J_1, it will always execute ahead of J_1 on every future stage, as the priorities are the same on all stages. Once the paths of J_i and J_1 split, the path of J_i never intersects the path of J_1, and J_i will cause no further preemptions.

Assume that the lemma is true for all $SM_{i,1} \leq k-1$, for some $k \geq 1$. To prove the result for $SM_{i,1} = k$. Let stage j be the last stage where both J_i and J_1 execute. As $SM_{i,1} \geq 1$, there exists a stage $j' < j$, where the paths of J_i and J_1 split. Further, let stage j' be the last such split in the paths of J_i and J_1. Figure 4 illustrates this scenario. Up to and including stage j', the number of split-merges is $k-1$, and hence from induction assumption, the number of times J_i preempts J_1 up to stage j' is at most k. Starting from stage $j'+1$, there are no split-merges in the paths of

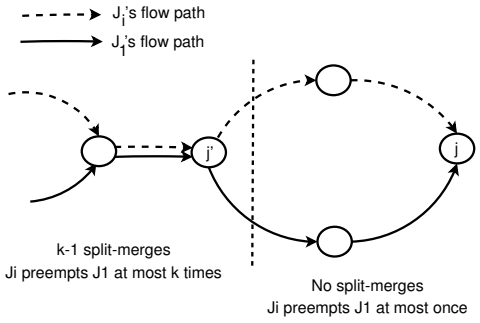

Figure 4. Figure illustrating proof of Lemma 1.

J_i and J_1 (the last split occurs at stage j'). From the basis step, the number of preemptions beyond stage $j'+1$ is at most one. Therefore, when $SM_{i,1} = k$, J_i preempts J_1 at most $k+1$ times. ■

The delay composition theorem can be proved by induction on task priority. We first prove the theorem for a two-job scenario (Lemma 2). 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 (with arbitrary job-flow path), and prove that the delay composition theorem still holds.

Lemma 2. *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*

$$Q = \sum_{i=1}^{2} 2C_{i,max}(1 + SM_{i,1}) + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max_{i=1,2}(C_{i,j}) \quad (2)$$

Proof. As the delay that J_2 inflicts on J_1 is more when J_2 preempts J_1, rather than when J_2 simply executes ahead of J_1, we assume that J_2 preempts J_1 every time the paths of the two jobs meet (the case where J_2 does not always preempt J_1 can be easily shown to cause a lower delay for J_1).

From Lemma 1, J_2 preempts J_1 at most $SM_{2,1} + 1$ times. Let the stages at which J_2 preeempts J_1 be $j_1, j_2, \ldots, j_{SM_{2,1}+1}$. Each such preemption occurs after a split-merge in the paths of J_2 and J_1. Let stage j'_k be a stage between j_k and j_{k+1} in the path of job J_1 ($j_{k+1} > j'_k \geq j_k$), such that j'_k is the last stage before stage j_{k+1} where J_1 waits for J_2, for $1 \leq k < SM_{2,1}+1$. For $k = SM_{2,1}+1$, j'_k is the last stage where J_1 waits for J_2 before completing its execution in the system. For notational simplicity, define stage $j_{SM_{2,1}+2}$ to be stage N. In other words, J_1 does not wait for J_2 between stages j'_k and j_{k+1}, for $1 \leq k \leq SM_{2,1}+1$. Figure 5 illustrates the delay experienced by J_1. Until stage j_1, job J_1 does not wait for job J_2. The delay of J_1 up to the time J_1 is preempted on stage j_1 is at most $C_{1,1} + \ldots + C_{1,j_1}$ (in the worst case, J_2 preempts

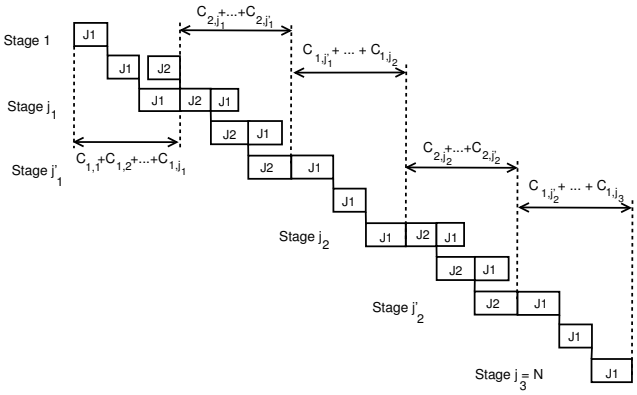

Figure 5. Figure showing the delay for job J_1, illustrating Lemma 2.

J_1 when J_1 has almost completed execution on stage j_1). Starting from the time J_1 is preempted on stage j_k and until J_1 starts execution on stage j'_k, for all k, the delay is given by

$$\sum_{\substack{t \in Path_1 \\ j_k \leq t \leq j'_k}} C_{2,t}$$

Stage j'_k is the last stage where J_1 waits for J_2 before a split-merge occurs and J_2 preempts J_1. Starting from J_1's execution on stage j'_k and up to the time J_1 is preempted on stage j_{k+1} (or completes execution in the system), the delay is given by

$$\sum_{\substack{t \in Path_1 \\ j'_k \leq t \leq j_{k+1}}} C_{1,t}$$

Notice that, in the above expressions for the delay of J_1, there is at most one execution time of a job on every stage 1 through N (in the path of J_1), except for stages j_k and j'_k, $1 \leq k \leq SM_{2,1}+1$, which contain two execution times, one from each job. To compute a delay bound, let us replace one per-stage computation time at each of the stages up to $N-1$ (that belong to $Path_1$) by $\max_{i=1,2} C_{i,t}$ for that stage. The delay of J_1 can therefore be written as,

$$\begin{aligned} Q &\leq \sum_{k=1}^{SM_{2,1}+1} (C_{2,j_k} + C_{2,j'_k}) + (\sum_{\substack{t \in Path_1 \\ t \leq N-1}} \max_{i=1,2} C_{i,t}) + C_{1,N} \\ &\leq 2C_{2,max}(1+SM_{2,1}) + C_{1,max} + \sum_{\substack{t \in Path_1 \\ t \leq N-1}} \max_{i=1,2} C_{i,t} \quad (3) \end{aligned}$$

Inequality 3 follows from the fact that stages j_k and j'_k (for every k) contribute an execution time of job J_2, each of which is less than $C_{2,max}$, and there are $2(SM_{2,1}+1)$ such terms. Stage N contributes an execution time of $C_{1,N} \leq C_{1,max}$. This proves the lemma. ■

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

DAG 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 DAG with $n-1$ higher priority jobs is at most*

$$Delay(J_1) \leq \sum_{i=1}^{n} 2C_{i,max}(1+SM_{i,1}) + \sum_{\substack{t \in Path_1 \\ t \leq N-1}} \max_{i=1}^{n}(C_{i,t})$$

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 when there are only two jobs in the system, J_1 and J_2. The delay composition theorem for two jobs is precisely Lemma 2.

Assume that the result is true for $n = k-1$ jobs, $k \geq 3$. That is,

$$Delay_{k-1}(J_1) \leq \sum_{i=1}^{k-1} 2C_{i,max}(1+SM_{i,1}) + \sum_{\substack{t \in Path_1 \\ t \leq N-1}} \max_{i \leq k-1} C_{i,t} \quad (4)$$

We need to show the result when a k^{th} job J_k, with highest priority and arbitrary flow path, is added. Let L_k be a system with k jobs, with arbitrary arrival times for each of the jobs. Let L_{k-1} be the system without job J_k, and with the same arrival times and flow paths for all the other jobs as in system L_k.

The number of split-merges in the paths of J_k and J_1 is $SM_{k,1}$. By breaking the path of J_1 after each split in the paths of J_k and J_1, the path of J_1 can be split into $SM_{k,1}+1$ parts. In each of these parts, J_k can preempt J_1 at most once. A key observation here is that job J_k in these parts, can be thought of as $SM_{k,1}+1$ independent jobs $J_{k_1}, J_{k_2}, \ldots, J_{k_{SM_{k,1}+1}}$. Each J_{k_i} executes in the i^{th} part ($1 \leq i \leq SM_{k,1}+1$), and does not meet J_1 at any of the other parts. We shall show that for every J_{k_i}, the job-additive component of J_1's delay due to J_{k_i} is at most $2C_{k,max}$. The delay composition theorem when job J_k is added will follow naturally.

We now consider three cases. The first case considers that J_{k_i} arrived before (or together with) J_1 to the first common stage where J_{k_i} completed execution before J_1, and such a first common stage is stage 1 (this can happen only for $i = 1$). The second case generalizes the first case, so that the first common stage where J_{k_i} completed execution before J_1 can be any stage $j > 1$. The case where J_{k_i} preempts J_1, is considered as the third case.

Case 1: J_{k_i} does not preempt J_1, and stage 1 is the first common stage between J_{k_i} and J_1.

If J_{k_i} executed after J_1 on some stage and never preempts J_1, it does not cause any delay to J_1. Therefore, it

is safe to assume that J_{k_i} arrived before or together with J_1 to stage 1, and to every subsequent common stage. Notice that, if there exists an idle time between the execution of J_{k_i} and J_1 on some stage j, the delay of J_1 on stage j is independent of the execution time of J_{k_i} (and other jobs that execute before the idle time) on stage j. Therefore, beyond the last stage j, where there is no idle time between the execution of J_{k_i} and J_1 (or J_1 waits for J_k to complete execution), J_{k_i} will not influence the delay of J_1 (jobs that J_{k_i} preempts on a stage, will also execute before the idle period on that stage). After J_{k_i} completes execution on stage j, the delay of J_1 in system L_k is identical to its delay in the system L_{k-1}, with only $k-1$ tasks and starting from stage j. Therefore, the delay of J_1 can be expressed as the delay up to the time J_{k_i} completes execution on stage j (J_k arrives before J_1 to the system), added to the worst case delay of J_1 in system L_{k-1} starting from stage j (as shown in Equation 5). This is shown in Figure 6.

$$\begin{aligned} Delay_k(J_1) &= F_{1,N} - A_{1,1} \\ &= (F_{1,N} - F_{k_i,j}) + (F_{k_i,j} - A_{1,1}) \quad (5) \end{aligned}$$

As J_{k_i} 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_i}'s execution on stage j ($F_{k,j}$), is at most the time J_{k_i} takes to complete execution up to stage j ($F_{k_i,j} - A_{k_i,1}$) as shown in Inequality 6 (although, this induces pessimism, the pessimism is due to the lack of knowledge as to how much earlier J_{k_i} arrives compared to J_1; future work can attempt to quantify this difference in arrival times more accurately, to obtain a better bound). J_{k_i} is the highest priority job in the system, and does not wait to execute on any of the stages. The time for J_{k_i} to complete execution up to stage j is $(\sum_{t \in Path_1, t<j} C_{k_i,t}) + C_{k_i,j}$. In addition to this, from induction assumption, the delay of J_1 from stages j through N is $\sum_{i=1}^{k-1} 2C_{i,max}(1 + SM_{i,1}) + \sum_{t \in Path_1, j \le t \le N-1} \max_{i \le k-1}(C_{i,t})$ (Inequality 7). It should be noted that the delay composition theorem accounts for the delay of J_1 due to any worst case arrival pattern of higher priority jobs, and therefore applies to the arrival pattern of jobs at stage j in system L_k. Moreover, we are only concerned with the worst case delay of J_1, and we are not concerned at the moment about whether jobs meet their designated deadlines. In computing such a worst case delay, some higher priority jobs may be delayed even beyond their deadlines so as to inflict a worst case delay to J_1. Such a worst case arrival pattern may cause the system to be unschedulable, but the delay composition theorem does not concern itself with schedulability. Isolating the delay composition theorem from the notion of deadlines and schedulability, enables us to obtain an upper bound on the delay experienced by a job, purely in terms of computation times of higher priority jobs.

Thus,

$$\begin{aligned} Delay(J_1) &\le (F_{1,N} - F_{k_i,j}) + (F_{k_i,j} - A_{1,1}) \\ &\le (F_{1,N} - F_{k_i,j}) + (F_{k_i,j} - A_{k_i,1}),\ as\ A_{k_i,1} \le A_{1,1} \quad (6) \\ &\le \Big(\sum_{t \in Path_{k_i}, t<j} C_{k_i,t}\Big) + C_{k_i,j} + \sum_{i=1}^{k-1} 2C_{i,max}(1 + SM_{i,1}) \\ &\quad + \sum_{\substack{t \in Path_1 \\ j \le t \le N-1}} \max_{i \le k-1}(C_{i,t}) \quad (7) \\ &\le C_{k_i,j} + \sum_{i=1}^{k-1} 2C_{i,max}(1 + SM_{i,1}) + \sum_{\substack{t \in Path_1 \\ t \le N-1}} \max_{i \le k}(C_{i,t}) \quad (8) \end{aligned}$$

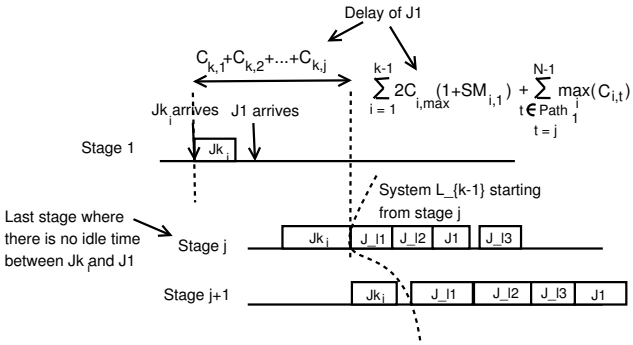

Figure 6. Figure showing the delay of J_1 for the case when J_k arrived before J_1 to the first common stage, which is stage 1.

Therefore, J_{k_i} delays J_1 by at most one maximum stage execution time (this is the job-additive component), where the maximum is over all stages where both J_{k_i} and J_1 execute ($j \in Path_{k_i} \cap Path_1$), apart from contributing to the maximum job execution times on stages 1 through j which belong to $Path_{k_i}$ (the stage-additive component).

Case 2: J_{k_i} arrived before or together with J_1 to the first common stage $j > 1$ where J_{k_i} completes execution before J_1.

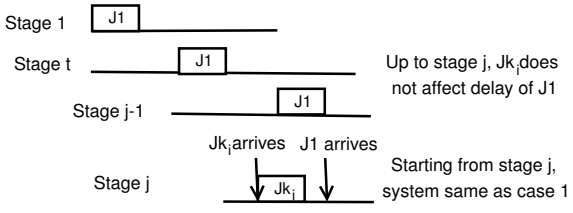

Figure 7. Figure showing the case when J_{k_i} arrived before J_1 to the first common stage j where J_{k_i} completed execution before J_1.

Up to stage j, the delay of J_1 is independent of job J_{k_i}, as J_{k_i} does not execute on these stages or executes after J_1. Starting from stage j, the system is identical to case 1,

wherein the system can be thought of as one with $N-j+1$ stages. Stage j is now the first stage in the system, and is also the first common stage where J_{k_i} and J_1 execute, and J_{k_i} arrived before or together with J_1 (shown in Fig 7). Therefore from case 1, J_{k_i} delays J_1 by at most one maximum stage execution time (the job-additive component), where the maximum is over all stages where both J_{k_i} and J_1 execute. Apart from the job-additive component, J_{k_i} contributes to the maximum job execution times on each stage (the stage-additive component) starting from stage j.

Case 3: J_{k_i} preempts J_1.

As J_{k_i} is a part of J_k which does not have any split-merges with the path of J_1, J_{k_i} preempts J_1 at most once. Until the time J_{k_i} preempts J_1, the delay of J_1 is independent of J_{k_i}. Let stage j be the first stage where J_{k_i} completes execution before J_1, and let J_{k_i} preempt J_1 at stage j. Beyond stage j, J_{k_i} arrives at each common stage before J_1. Therefore, the system beyond stage j can be thought of as one having $N-j$ stages, and J_{k_i} arriving before J_1. We can then apply the result from case 2. Thus, the delay of J_1 can be thought of as two components - the delay up to stage j, and the delay of J_1 beyond stage j.

The fact that J_{k_i} preempted some job at stage j (it is possible that J_{k_i} preempted some job, which in turn had preempted J_1), implies that there was a job executing when J_{k_i} arrived at stage j. Further, there is no idle time between the executions of J_{k_i} and J_1. Let J_{l_1}, J_{l_2}, ..., J_{l_s}, be the jobs that execute between J_{k_i} and J_1 on stage j (Figure 8). J_{l_1} is delayed by J_{k_i} up to stage j by at most $C_{k_i,j}$. Similarly, irrespective of previous stages, each of J_{l_2}, J_{l_3}, ..., J_{l_s}, and J_1 are delayed by an amount $C_{k_i,j}$ due to J_{k_i} up to stage j.

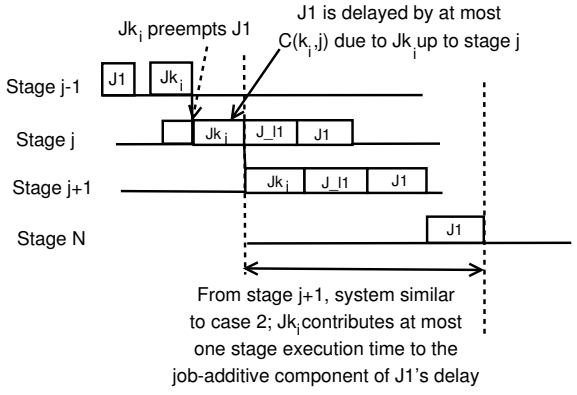

Figure 8. Figure showing the case when J_{k_i} arrived after J_1 and preempts J_1 at stage j.

Beyond stage j, as mentioned earlier the system is identical to case 2 (as J_{k_i} arrived before J_1 to every subsequent common stage). From the result of case 2, the additional delay that J_{k_i} causes J_1 is one maximum stage execution time of J_{k_i} (the job-additive component), apart from J_{k_i}'s contribution to the stage-additive component $\max_i(C_{i,t})$, for $j+1 \leq t \leq N-1$ (from Inequality 7). Figure 8 shows this scenario. We showed that the delay due to J_{k_i} up to stage j is at most $C_{k,j}$. Therefore, the total job-additive delay to J_1 due to J_{k_i} is at most the sum of two maximum stage execution times of J_{k_i}, that is $2C_{k_i,max}$.

From the above three cases, each J_{k_i} adds to the delay of J_1 at most two maximum stage execution times of J_k. There are $1+SM_{k,1}$ such jobs. Therefore, the total job-additive delay that J_k causes J_1 is at most $2C_{k,max}(1+SM_{k,1})$. Each J_{k_i} is part of the stage-additive component of J_1's delay. This delay is simply the sum of one maximum execution time over all jobs on each stage. The maximum of the execution times of all J_{k_i} on each stage, is simply the execution time of job J_k on that stage. This is the contribution of J_k to the stage-additive component of J_1's delay, as in the expression of the delay composition theorem. This proves the induction step. Using this together with Lemma 2, the delay composition theorem is proved. ■

4. Schedulability and DAG Reduction

In this section, we elucidate a systematic reduction of the schedulability problem in an acyclic distributed system 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_{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_{wc}. We expect that future work can improve upon this definition using more in-depth analysis. In the absence of further information, set S_{wc} is defined as follows.

Definition: 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 which have at least one common execution stage with J_1, and whose intervals $[A_i, A_i+D_i]$ overlap the interval where J_1 was present in the system, $[A_1, A_1+delay(J_1)]$.

Observe that the above is a conservative definition. It simply excludes the impossible. A job that does not have a common execution stage with J_1 can never delay J_1. Further, 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.

$$delay(J_1) \leq \sum_{i \in S_{wc}} 2C_{i,max}(1+SM_{i,1}) + \sum_{\substack{j \in Path_1 \\ j \leq N-1}} \max_{i \in S_{wc}} C_{i,j} \quad (9)$$

The reduction to a single stage system is then conducted by (i) replacing each job J_i in S_{wc} by an equivalent single stage

job of execution time equal to $2C_{i,max}(1 + SM_{i,1})$, and (ii) adding a lowest-priority job, J_e^* of execution time equal to $\sum_{j \in Path_1, j \leq N-1} \max_i(C_{i,j})$ (which is the last term in Inequality (9)), and deadline same as that of J_1. By the delay composition theorem, the total delay incurred by J_1 in the acyclic distributed system 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 (9).

For the case of periodic tasks, the delay bound can be significantly improved based on the observation that not all invocations of a higher priority task T_i can preempt an invocation of T_1, $1 + SM_{i,1}$ times. Let us suppose that during the execution of an invocation of T_1, at most x invocations of T_i preempt T_1. If one such invocation preempts T_1 $1 + SM_{i,1}$ times, it implies that T_1 has progressed past the last split-merge between the paths of T_i and T_1, and therefore, future invocations of T_i can preempt T_1 at most once. Extending this argument, at most one invocation of T_i can preempt T_1 at each split-merge between the paths of T_i and T_1. Therefore, the maximum number of preemptions that x invocations of T_i can cause T_1 is $x + SM_{i,1}$, rather than $x(1 + SM_{i,1})$. Notice that the factor $SM_{i,1}$ now appears only once for each task, rather than once for each invocation of every task.

The reduction to a single stage system for periodic tasks can then be conducted by (i) replacing each periodic task T_i by an equivalent single stage task of execution time equal to $2C_{i,max}$, and (ii) adding a lowest priority task with computation time equal to $\sum_{j \in Path_1, j \leq N-1} \max_i(C_{i,j}) + \sum_i 2C_{i,max}SM_{i,1}$.

For example, let us illustrate this transformation in the case of rate-monotonic scheduling of periodic tasks with periods equal to deadlines. Consider a set of periodic tasks, where each task T_i has a period P_i. As shown in Figure 9, there can be at most one invocation of each higher-priority task T_i in S_{wc} that arrives before an invocation of T_1. The number of invocations of each task T_i that arrive after the invocation of T_1 and delay it, is no larger than $\lceil \frac{delay_1}{P_i} \rceil$. 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 $C_e^* = \sum_i C_{i,max} + \sum_i 2C_{i,max}SM_{i,1} + \sum_{j \in Path_1, j \leq N-1} \max_i(C_{i,j})$. The task further has the same period and deadline as T_1 in the original set.

- Tasks T_i^*, each has the same period and deadline as one T_i in the original set, and has an execution time equal to $C_i^* = 2C_{i,max}$.

Hence, if task T_e^* is schedulable on a uniprocessor, so is T_1 on the original acyclic distributed system. The transformation is complete. In Section 5, we present DAG schedulability expressions for deadline monotonic scheduling based on the above task set reduction.

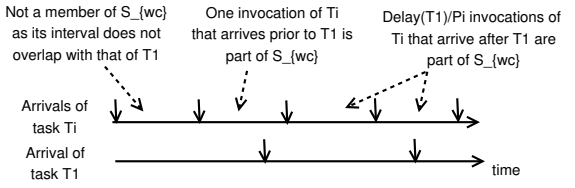

Figure 9. Invocations in S_{wc}.

5. Utility of Derived Result

The reduction described in the previous section enables large complex acyclic distributed systems to be easily analyzed using any single stage schedulability analyses technique. In this respect, our solution is indeed a 'meta-schedulability test'. The only assumptions made by the reduction on the scheduling model are fixed priority preemptive scheduling, and tasks do not block for resources on any of the stages (i.e., independent tasks). In the rest of this section, we concern ourselves with schedulability analysis for periodic tasks. We assume that task T_i has a higher priority than task T_k, if $i < k$.

As examples, we show how the Liu and Layland bound [8] and the necessary and sufficient test based on response time analysis [2] can be applied to analyze periodic tasks in an acyclic distributed system. Other uniprocessor schedulability tests can be applied in a similar manner.

The Liu and Layland bound [8], applied to periodic tasks in an acyclic distributed system is:

$$\frac{C_e^*(i)}{D_i} + \sum_{k=1}^{i-1} \frac{C_k^*}{D_k} \leq i(2^{\frac{1}{i}} - 1)$$

for each $i, 1 \leq i \leq n$, where $C_e^*(i) = \sum_{k=1}^{i} C_{k,max} + \sum_{k=1}^{i-1} 2C_{k,max}SM_{k,i} + \sum_{j \in Path_1, j \leq N-1} \max_{k \leq i}(C_{k,j})$; $C_k^* = 2C_{k,max}$. $C_{i,max}$ is the largest execution time of T_i on any stage, D_i is the end-to-end deadline, and n is the number of periodic tasks in the system.

The necessary and sufficient test for schedulability of periodic tasks under deadline monotonic scheduling proposed in [2], used together with our meta-schedulability test, will have the following recursive formula for the worst case response time R_i of task T_i:

$$R_i^{(0)} = C_e^*(i)$$

$$R_i^{(k)} = C_e^*(i) + \sum_{j<i} \left\lceil \frac{R_i^{(k-1)}}{P_j} \right\rceil C_j^*$$

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.

6. Simulation Results

In this section, we present results from simulation studies conducted using a custom-built simulator that models a

distributed system with directed acyclic flows. Each task requires processing at a fixed set of nodes in the distributed 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] or response time analysis [2], together with our reduction of the multistage distributed system to a single stage, as shown in Section 5. Each periodic task that arrives at the system is tentatively added to the set of all 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.

Although the analysis technique derived in this paper is valid for any fixed priority scheduling algorithm, we only present results for deadline monotonic scheduling due to its widespread use. 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 utilization values obtained from 100 executions of the simulator, with each execution running for 80000 task invocations. The default number of nodes in the distributed system is assumed to be 8. Each task on arrival requests processing on a sequence of nodes, with each node in the distributed system having a probability of RP (for Route Probability) of being selected as part of the route. The task's route is simply the sequence of nodes in increasing order of their node identifier. The default value of RP is chosen as 0.8. Note that all task routes are directed and acyclic. Deadlines (equal to the periods, unless explicitly specified) of tasks are chosen as $10^x a$ simulation seconds, where x is uniformly varying between 0 and DR (for deadline ratio), and $a = 500 * N$, where N is the number of stages in the task's route. 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. The default value for DR is 3, and we refer to DR as the deadline ratio parameter. 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 $\frac{DT}{N}$, where D is the deadline of the task and T is the task resolution. The stage execution times of tasks were allowed to vary up to 10% on either side of the mean. Task preemptions are assumed to be instantaneous, that is, the task switching time is zero. We used a task resolution of $1 : 100$. The default single stage schedulability test used is the response-time analysis technique presented in [2]. The 95% confidence interval for all the utilization values presented in this section is within 0.02 of the mean value, which is not plotted for the sake of legibility.

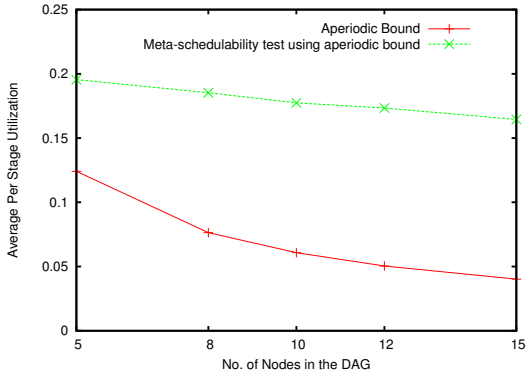

Figure 10. Comparison of meta-schedulability test with the traditional test based on aperiodic schedulability analysis

For aperiodic tasks, we used our meta-schedulability test together with the uniprocessor bound derived in [1], and compared it with the bound presented in [6] (performing one test for each task), which is based on the same aperiodic task bound. For both these tests, while keeping other simulation parameters constant, we varied the number of stages (nodes in the DAG) and measured the utilization. Figure 10 presents this comparison. We observe that the meta-schedulability test performs better than the pipeline bound presented in [6], especially for larger systems. For the rest of this section, we shall concern ourselves with only periodic tasks.

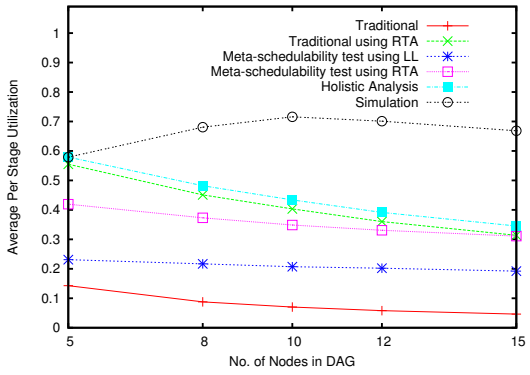

Figure 11. Comparison of meta-schedulability test with traditional tests and holistic analysis

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.

This implementation has two sources of pessimism. When the end-to-end deadlines of tasks are comparable to their periods of invocation, it does not consider the overlap in the executions of different stages. Further, it assumes that subtasks on each stage are invoked with a period equal to the per-stage deadline, thereby increasing the perceived load on each stage. To decouple the effects of these two sources of pessimism, 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. If using response time analysis, 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. Since response time analysis can handle deadlines different from the period, the only source of pessimism for this test is not considering the overlap in the executions of different stages. In holistic analysis the response time on one stage is considered as the jitter for the next change. It does not divide the end-to-end deadline into single stage deadlines. Nevertheless, by considering the previous stage response time as the jitter, it allows a job to be delayed by a higher priority job on every stage of the pipeline and does not account for the overlap in the execution of different stages of the pipeline, when the end-to-end deadlines are comparable to the task periods. We study the performance of our meta-schedulability test, using both the Liu and Layland bound and response time analysis for the single stage.

We conducted experiments to measure the average per-stage utilization for different number of nodes in the directed acyclic graph, when using admission controllers based on each of these five tests. Figure 11 plots this comparison. The values referred to as 'simulation' are the lowest utilization values at which deadline misses were observed in the absence of any admission controller (for the same task parameters). Note that the utilization values presented for the admission controllers are average values, and therefore it is possible for such an average value to be higher than the lowest utilization value at which deadline misses were observed.

We note that holistic analysis performs the best in terms of average utilization. The reason for the meta-schedulability test to perform worse than holistic analysis is its pessimistic assumption that whenever the routes of two tasks merge, the higher priority task inflicts a worst case delay on the lower priority task. Future work can try to decrease this pessimism. Nevertheless, as the system size increases, the rate at which utilization decreases is least for our meta-schedulability test. Beyond 10 stages, the performance of our meta-schedulability test using RTA is comparable to holistic analysis and the traditional test using RTA. The reader should bear in mind that the meta-schedulability test when applied to a particular task, only assumes knowledge of tasks in the route of the task under consideration. In contrast, both the holistic analysis technique and the traditional test using response time analysis require global knowledge of all tasks and nodes in the system to predict the schedulability of a particular task. To the best of our knowledge, there is no existing analysis technique to predict the schedulability of a task based on local route information only. The meta-schedulability test is the first analysis technique to work purely based on local route information. Further, the execution complexities of holistic analysis and the traditional test using RTA are higher than that of our meta-schedulability test.

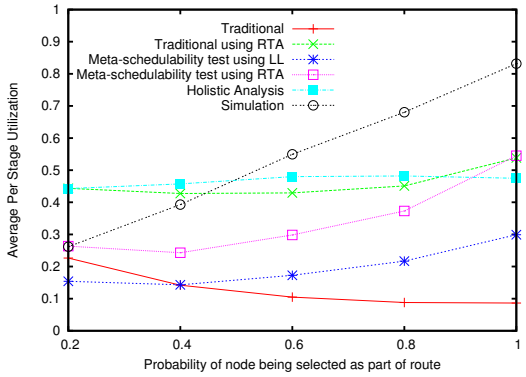

Figure 12. Comparison of meta-schedulability test with traditional tests and holistic analysis for different route probabilities

We conducted a similar comparison of the five admission controllers as in the previous experiment, but for different values of the Route Probability (RP) parameter, which is the probability with which each node in the system is chosen as part of the route of each task. The RP parameter was varied from 0.2 to 1.0 in steps of 0.2. Note that the RP parameter of 1.0 denotes a perfectly pipelined system, where each task executes sequentially on all the nodes in the distributed system. Here again, we observe that our meta-schedulability test does not perform as well as holistic analysis and the traditional test using response time analysis, due to its pessimistic assumption that when the routes of tasks merge, the higher priority task inflicts a worst case delay on each lower priority task.

The above results have all been obtained by setting the end-to-end deadlines equal to the periods of tasks. Figure 13 plots a comparison of the meta-schedulability test using response time analysis with holistic analysis for different ratios of the end-to-end deadlines to the periods. 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. When end-to-end deadlines are lesser than task periods (for example, ratio of 0.5), the utilization achieved by the meta-schedulability test is com-

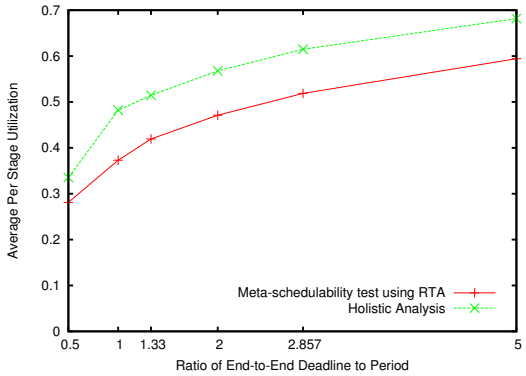

Figure 13. Comparison of meta-schedulability test with holistic analysis for different ratios of end-to-end deadline to task periods

parable to that of holistic analysis. Holistic analysis tends to be pessimistic when the deadlines are short, as it assumes that a lower priority task would be delayed by a higher priority task invocation at every stage in the system.

7. Related Work

The seminal work of Liu and Layland [8], was the first study of feasible regions in real-time systems. Under certain specific restrictions, they presented utilization bounds for uniprocessor systems. These utilization bounds were extended to multiprocessor systems in [3]. Resource constraints were considered and a single-stage utilization bound which was less pessimistic that the Liu and Layland bound was presented. While these utilization bounds were sufficient conditions for schedulability, exact tests such as [2, 7] were also proposed.

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 huge 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. These techniques divide the end-to-end deadline into individual per-stage deadlines, and tend to ignore the overlap that exists between the execution of different pipeline stages. A distributed pipeline framework was presented in [4]. Offset-based response time analysis techniques for 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]. Here, the worst case delay at a stage is taken as the jitter for the next stage. While this technique does not divide the end-to-end delay into sub-deadlines for individual stages, it nevertheless does not account for the overlap in the execution of different pipeline stages.

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 system. Our simulation results have shown that the meta-schedulability test outperforms this test in terms of average per-stage utilization.

8. Conclusions and Future Work

This paper presents a delay composition rule for distributed systems, where the routes of tasks form a directed acyclic graph. 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. The rule works purely based on information available along the route followed by the task under consideration, and does not require global knowledge of other parts of the distributed system, unlike traditional analysis techniques such as holistic analysis. This composition rule leads to the reduction of the 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 distributed systems. We envision the main application of this test in large distributed systems, where it is infeasible to apply traditional schedulability analysis.

This work opens the door for schedulability theory research in distributed systems in multiple directions. There is no evidence that the proposed composition rule is optimal. More efficient composition rules, if identified, can help reduce the pessimism further. Extending the delay composition theorem and the meta-schedulability test to non-preemptive scheduling and resource blocking can help widen the applicability of the result. The delay composition rule could aid the study of obtaining optimal rate control,

routing and scheduling policies in distributed systems and large networks. The current work addresses only directed acyclic systems and does not account for loops. Accounting for loops can enable the result to be applied to semiconductor chip manufacturing plants, where chips revisit the same service center multiple times before exiting the system.

References

[1] T. Abdelzaher and C. Lu. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers*, 53(3), March 2004.

[2] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, pages 284–292, 1993.

[3] E. Bini, G. Buttazzo, and G. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *13th Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.

[4] S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: End-to-end analysis of heterogeneous multi-resource real-time sytems. In *IEEE International Conference on Distributed Computing Systems*, pages 204–211, May 1995.

[5] G. Fohler and K. Ramamritham. Static scheduling of pipelined periodic tasks in distributed real-time systems. In *9th Euromicro Workshop on Real-Time Systems*, pages 128–135, June 1997.

[6] W. Hawkins and T. Abdelzaher. Towards feasible region calculus: An end-to-end schedulability analysis of real-time multistage execution. In *IEEE Real-Time Systems Symposium*, December 2005.

[7] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[9] J. Palencia and M. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *Euromicro Conference on Real-Time Systems*, pages 3–12, July 2003.

[10] R. Pellizzoni and G. Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*, pages 66–75, March 2005.

[11] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *IEEE International Conference on Distributed Computing Systems*, pages 108–115, 1990.

[12] J. Schneider. Cache and pipeline sensitive fixed priority scheduling for preemptive real-time systems. In *IEEE Real-Time Systems Symposium*, pages 195–204, November 2000.

[13] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Elsevier Microprocessing and Microprogramming*, 40(2-3):117–134, 1994.

[14] J. Xu and D. Parnas. On satisfying timing constraints in hard real-time systems. *IEEE Transactions on Software Engineering*, 19(1):70–84, January 1993.

[15] N. Zhang, A. Burns, and M. Nicholson. Pipelined processors and worst case execution times. *Real-Time Systems*, 5(4):319–343, 1993.

[16] Y. Zhang, C. Lu, C. Gill, P. Lardieri, and G. Thaker. End-to-end scheduling strategies for aperiodic tasks in middleware. Technical Report WUCSE-2005-57, University of Washington at St. Louis, December 2005.