Transforming Distributed Acyclic Systems into Equivalent Uniprocessors Under Preemptive and Non-Preemptive Scheduling

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

Abstract

Many scientific disciplines provide composition primitives whereby overall properties of systems are composed from those of their components. Examples include rules for block diagram reduction in control theory and laws for computing equivalent circuit impedance in circuit theory. No general composition rules exist for real-time systems whereby a distributed system is transformed to an equivalent single stage analyzable using traditional uniprocessor schedulability analysis techniques. Towards such a theory, in this paper, we extend our previous result on pipeline delay composition for preemptive and non-preemptive scheduling to the general case of distributed acyclic systems. Acyclic *systems are defined as those where the superposition of all task flows gives rise to a Directed Acyclic Graph (DAG). The new extended analysis provides a worst-case bound on the end-to-end delay of a job under both preemptive as well as non-preemptive scheduling, in the distributed system. A simple transformation is then shown of the distributed task system into an equivalent uniprocessor task-set analyzable using traditional uniprocessor schedulability analysis. Hence, using the transformation described in this paper, the wealth of theory available for uniprocessor schedulability analysis can be easily applied to a larger class of distributed systems.*

1. Introduction

Rigorous theory exists today for schedulability analysis of uniprocessors and multiprocessors, while mostly heuristics are used to analyze larger arbitrary-topology distributed systems. This raises the question of whether a formal transformation can be found that converts a given distributed system into an equivalent uniprocessor system analyzable using the wealth of existing uniprocessor schedulability theory. Such transformations are not uncommon in other contexts. For example, control theory describes transformations that reduce complex block diagrams into an equivalent single block that can be analyzed for stability and performance properties. In circuit theory, *Kirchoff Laws*, together with other reduction rules, can reduce a circuit to a single equivalent source and impedance.

In an earlier paper [6], the authors derived a delay composition rule that provided a bound on the end-to-end delay of jobs in a pipelined distributed system under preemptive scheduling. An extended (journal) version of that paper [7] provided a pipeline bound under non-preemptive scheduling as well. These rules permit a simple transformation of the pipelined system into an equivalent uniprocessor system analyzable using traditional uniprocessor schedulability analysis. Motivated by the ultimate goal of a general transformation theory, this paper extends the aforementioned results to a larger class of distributed systems; namely, those described by arbitrary Directed Acyclic Graphs (DAG). While the original results apply to priority-based resource scheduling only, we demonstrate how the framework can trivially accommodate resource partitioning (e.g., TDMA) as well.

Observe that reductions of complex systems into simpler ones (e.g., in control or circuit theory) depend on the choice of the two system end-points between which the reduction is made. For example, given two points in a complex circuit, a reduction can compute the equivalent impedance between these two specific points. In a similar manner, this paper addresses distributed system reduction as seen *from the perspective of a given distributed task*. Informally, the question addressed in this paper is as follows: given a distributed task A in a distributed task system of workload W_{dist}, can we systematically construct a uniprocessor task B and a uniprocessor workload W_{uni}, such that if B is schedulable on the uniprocessor, A is schedulable on the distributed system? We show that such a transformation is possible and that it is linear in the number of tasks on A's path. A wide range of existing schedulability analysis techniques can thus be applied to the uniprocessor task set, to analyze the distributed system under both preemptive and non-preemptive scheduling.

Earlier schedulability analysis, such as holistic analysis [13], requires global knowledge of task routes and computation times in order to predict the worst case end-to-end delay of a task. In contrast, the transformation derived in this paper depends only on the load that the analyzed task encounters along its path (i.e., it is linear in the number of tasks scheduled on the same resources as task A). We also show that the schedulability analysis technique developed in this paper

*The work reported in this paper was supported in part by the National Science Foundation under grants CNS 06-13665, CNS 06-15318, and CNS 05-53420

1068-3070/08 $25.00 © 2008 IEEE
DOI 10.1109/ECRTS.2008.31

outperforms existing techniques by a factor that grows larger with system scale. For small distributed systems, existing literature is adequate. Hence, the new results nicely complement existing literature. They are particularly useful for analysis of large data-driven systems where requests are streamed in the same direction across a large number of processing stages possibly tailored to the type of request. Distributed client-server applications that involve round trips (i.e., cyclic execution paths) are not the goal of this paper.

The remainder of this paper is organized as follows. Section 2 briefly describes the system model. Section 3, generalizes previous pipeline delay composition results to the DAG case, and provides an improved bound for the special case of periodic tasks. We show how partitioned resources can be handled in Section 4. In Section 5, we present distributed system reduction to a single stage and show how well-known single stage schedulability analysis techniques can be used to analyze acyclic distributed systems. In Section 6, we describe how the system model can be extended to include tasks whose sub-tasks themselves form a DAG. In Section 7, we compare the performance of schedulability analysis based on our delay composition theorem with holistic analysis, and show that under certain conditions non-preemptive scheduling can result in higher system utilization than preemptive scheduling. Related work is reviewed in Section 8. We discuss future work in Section 9, especially how the results derived in this paper can be extended to non-acyclic systems. We conclude in Section 10.

2. System Model

In this paper, we consider a multi-stage distributed system that serves real-time tasks. In our system model, we first assume that each task traverses a path of multiple stages of execution and must exit the system within specified end-to-end latency bounds. The combination of all such paths forms a DAG. We then extend the above results to tasks whose subtasks themselves form a DAG.

We assume that all stages are scheduled in the same priority order. If some resources are partitioned (e.g., in a TDMA fashion) with priorities applied within partitions, we consider each partition to be a slower prioritized resource and add a delay (the maximum time a task waits for its slot). For example, partitioning communication resources among senders using a TDMA or token-passing protocol is a common approach for ensuring temporal correctness in distributed real-time systems.

No assumptions are made on the periodicity of the task set. For periodic tasks, multiple invocations of the task can be present concurrently (this can occur when the end-to-end deadline is larger than the period of invocation of the task). Different invocations of the same task need not have the same priority. We henceforth call each task invocation a *job*. Thus, the delay composition rule and the corresponding transformation to an equivalent uniprocessor apply to static-priority scheduling (such as rate-monotonic), dynamic-priority scheduling (such as EDF), aperiodic task scheduling, as well as partitioned-resource systems.

3. Delay Composition for DAGs

For the purposes of distributed system transformation, let us view the system as seen from the perspective of some job J_1 of relative deadline D_1 whose schedulability is to be analyzed. Job J_1 traverses a multistage path, $Path_1$, in the system, where each stage is a single resource (such as a processor or a network link). While the system may have other resources, we consider only those that J_1 traverses. Let there be N such resource stages, numbered $1, 2, \ldots, N$ in traversal order, constituting $Path_1$. Let the arrival time of any job J_i to stage j of $Path_1$ be denoted $A_{i,j}$. 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 pass through stage j, then $C_{i,j}$ is zero. Let $C_{i,max}$, denote J_i's largest stage execution time, on stages where both J_i and J_1 execute. Observe that a job J_i ($i \neq 1$) may meet with J_1's path and diverge several times. Let M_i be the number of times the paths of J_i and J_1 meet (for a sequence of one or more consecutive stages that ends with one job using a stage not used by the other). In Sections 3.1 and 3.2, we derive the proofs for the preemptive and non-preemptive versions of the DAG delay composition theorem, respectively. We then leverage it to present a transformation to an equivalent uniprocessor.

3.1 The Preemptive Case

In this section, we bound the maximum delay of J_1 as a function of the execution requirements of higher priority jobs that interfere with it along its path. The following theorem states the delay bound.

Preemptive 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 N stages can be composed from the execution parameters of jobs that delay it (denoted by set $\bar{S}$) as follows:*

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

Proof. The proof of the preemptive DAG delay composition theorem for job J_1 is accomplished by transforming the system to a pipelined system in which the worst case delay of J_1 is no lower than that in the original system. The pipeline delay composition theorem [6] can then be applied to derive a worst case end-to-end delay bound for job J_1.

Consider a job J_i whose path meets with the path of J_1 in the distributed system then splits from it multiple times. Every time the paths of J_i and J_1 meet for one or more consecutive stages, we consider J_i's execution on those stages to be a new job J_{i_k} as shown in Figure 1. In other words, we split each J_i into M_i independent jobs, each of which has one or more consecutive common stages of execution with J_1. The transformation effectively relaxes the precedence relations that chain together the jobs J_{i_k} in the original system. The relaxation can only decrease the schedulability of J_1 by making it possible to construct more aggressive worst-case arrival patterns of the higher-priority jobs J_{i_k}. Hence,

if J_1 is schedulable in the new system, it is schedulable in the original system. The new system, however, can be analyzed by the pipeline result in [6], which applies when all tasks traverse a chain of stages in the same direction. We prove the correctness of such an analysis more formally in the appendix.

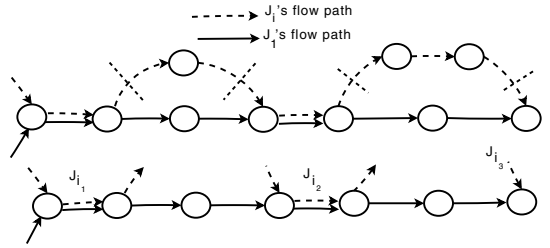

Figure 1: Figure illustrating splitting job J_i into M_i independent sub-jobs.

Let set $\bar{Q}$ denote the set of all higher priority jobs J_{i_k} over all jobs J_i and including J_1. We can now apply the pipeline delay composition theorem [6] to bound the worst case end-to-end delay of J_1. We have:

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

Since each job J_i gave rise to M_i sub-jobs J_{i_k}, the summation over all jobs J_{i_k} in set $\bar{Q}$ (in the first term of the bound above) can be rewritten as a double summation over jobs J_i in $\bar{S}$ and their M_i sub-jobs. Similarly, the maximization in the second term can also be broken into two as follows:

$$Delay(J_1) \leq \sum_{J_i \in \bar{S}} \sum_{k=1}^{M_i} 2C_{i_k,max} + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max_{J_i \in \bar{S}} (\max_{k \leq M_i}(C_{i_k,j}))$$

This is equivalent to:

$$Delay(J_1) \leq \sum_{J_i \in \bar{S}} 2C_{i,max} M_i + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max_{J_i \in \bar{S}}(C_{i,j}) \quad (3)$$

This proves the preemptive DAG delay composition theorem. □

The above theorem presents a delay bound for J_1 given any arbitrary set of higher priority jobs $\bar{S}$. For the special case where the higher priority jobs are invocations of periodic tasks, denoted by set R, an improved delay bound can be derived based on the observation that not all sub-jobs of each invocation of task $T_i \in R$ contribute to the delay of J_1. Let x_i denote the number of invocations of task T_i that can potentially contribute to the delay of J_1 (the number of invocations of T_i that belong to set $\bar{S}$). The following corollary derives this improved bound for periodic tasks.

Corollary 1. *Under preemptive scheduling, the end-to-end worst-case delay bound for a job J_1 of a lowest priority task T_1, in the presence of higher priority periodic tasks (denoted by set R) is given by:*

$$Delay(J_1) \leq \sum_{T_i \in R} 2C_{i,max}(x_i + M_i) + \sum_{\substack{j \in Path_1 \\ j \leq N-1}} \max_{T_i \in R}(C_{i,j}) \quad (4)$$

Proof. Each invocation of T_i has M_i sub-jobs, and there are x_i such invocations in set $\bar{S}$. The key observation is that not all $x_i \times M_i$ sub-jobs of T_i can delay J_1, and by removing the sub-jobs that cannot delay J_1 from set $\bar{Q}$, an improved delay bound can be obtained for periodic tasks. To see that, consider the delay of one invocation J_1 of the periodic task under consideration. This invocation makes forward progress along its path and never revisits a stage. Hence, for example, if all M_i sub-jobs of one invocation of T_i delay J_1, it implies that J_1 has already progressed past a certain stage on its path (specifically, past the last stage, say g, where the paths of T_i and T_1 meet). Therefore, sub-jobs of future invocations of T_i that may execute later at those already traversed stages (i.e., stages prior to g) will not interfere with J_1. Extending this argument, if $y_1 \leq M_i$ sub-jobs of the first invocation of T_i delay J_1, then only $y_2 \leq M_i - y_1 + 1$ sub-jobs of the second invocation can delay J_1. Likewise, only $y_3 \leq M_i - (y_1 + y_2) + 2$ sub-jobs of the third invocation can delay J_1. Therefore, the total number of sub-jobs of T_i that delay J_1 is bounded by $y_1 + y_2 + \ldots + y_{x_i} \leq x_i + M_i$. Thus, to calculate the worst-case delay for J_1, we can discard all but $x_i + M_i$ sub-jobs of T_i from set $\bar{Q}$. This new system, however, can be analyzed by the pipeline result in [6] as before. The corollary follows by grouping all sub-jobs belonging to the same periodic task together.

$$\begin{aligned} Delay(J_1) &\leq \sum_{J_i \in \bar{Q}} 2C_{i,max} + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max_{J_i \in \bar{Q}}(C_{i,j}) \\ &\leq \sum_{T_i \in R} 2C_{i,max}(x_i + M_i) + \sum_{\substack{j \in Path_1 \\ j \leq N-1}} \max_{T_i \in R}(C_{i,j}) \end{aligned}$$

□

3.2 The Non-Preemptive Case

Next, we bound the maximum delay of J_1 under non-preemptive scheduling. Unlike the previous case, here J_1 might also be delayed by lower-priority jobs, collectively denoted by set $\underline{S}$. In particular, it may be delayed by up to one such job on each stage. The following theorem states the new delay bound.

Non-preemptive DAG Delay Composition Theorem. *Assuming a non-preemptive scheduling policy with the same priorities across all stages for each job, the end-to-end delay of a job J_1 of N stages can be composed from the execution parameters of other jobs that delay it (denoted by set S) as follows:*

$$\begin{aligned} Delay(J_1) \leq \sum_{J_i \in \bar{S}} C_{i,max} M_i + \sum_{\substack{j \in Path_1 \\ j \leq N-1}} \max_{J_i \in S}(C_{i,j}) \\ + \sum_{j \in Path_1} \max_{J_i \in \underline{S}}(C_{i,j}) \end{aligned} \quad (5)$$

Proof. To bound the worst case delay for a job J_1 under non-preemptive scheduling, we first transform the task set by removing all lower-priority jobs, and instead adding to the computation time of J_1 on each stage i the maximum blocking delay due to jobs in $\underline{S}$. Let us call the adjusted computation time, $C_{1',j}$. Hence, $C_{1',j} = C_{1,j} + \max_{J_i \in \underline{S}}(C_{i,j})$.

This results in a system of only J_1 and higher-priority jobs. Observe that if the new system is schedulable so is the original one because we extended J_1's computation time by the worst case amount of time it could have been blocked by lower priority jobs. We then cut each higher-priority job J_i into M_i sub-jobs as we did in the preemptive case, and let $\bar{Q}$ denote the set of all such sub-jobs including J_1. The resulting system is a task pipeline to which the non-preemptive pipeline delay composition theorem [7] applies. According to this theorem:

$$\begin{aligned} Delay(J_1) &\leq \sum_{J_i \in \bar{Q}} C_{i,max} + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max_{J_i \in \bar{Q}}(C_{i,j}) \\ &\leq \sum_{J_i \in \bar{S}} \sum_{k=1}^{M_i} C_{i_k,max} + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max(\max_{\substack{J_i \in \bar{S}, \\ k \leq M_i}}(C_{i_k,j}), C_{1',j}) \\ &\leq \sum_{J_i \in \bar{S}} C_{i,max} M_i + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max(\max_{J_i \in \bar{S}} C_{i,j}, C_{1',j}) \\ &\leq \sum_{J_i \in \bar{S}} C_{i,max} M_i + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max_{J_i \in \bar{S}}(C_{i,j}) + \sum_{j \in Path_1} \max_{J_i \in \underline{S}}(C_{i,j}) \quad (6) \end{aligned}$$

Inequality 6 follows by replacing $C_{1',j}$ by $C_{1,j} + \max_{J_i \in \underline{S}}(C_{i,j})$ and making the delay due to lower priority jobs a separate term. This proves the non-preemptive DAG delay composition theorem. □

For the special case of periodic tasks, an improved bound can be derived as before. Let the set of all periodic tasks be denoted by R. Let $\bar{R}$ denote the set of higher priority tasks including T_1 and $\underline{R}$ denote the set of lower priority tasks.

Corollary 2. *Under non-preemptive scheduling, the end-to-end delay bound for a job J_1 of task T_1, in the presence of other periodic tasks (denoted by set R) is given by:*

$$\begin{aligned} Delay(J_1) \quad \leq \quad & \sum_{T_i \in \bar{R}} C_{i,max}(x_i + M_i) + \sum_{\substack{j \in Path_1, \\ j \leq N-1}} \max_{T_i \in \bar{R}}(C_{i,j}) \\ & + \sum_{j \in Path_1} \max_{T_i \in \underline{R}}(C_{i,j}) \quad (7) \end{aligned}$$

Proof. The proof is similar to the preemptive case, and we do not repeat the proof in the interest of brevity. □

4. Handling Partitioned Resources

The delay composition theorem as described so far, is only applicable to systems where resources are scheduled in priority order. However, resources such as network bandwidth are often *partitioned* among jobs, for example, using a TDMA protocol. In such a partitioned resource, a job may access the resource only during its reserved time-slot or token. Multiple jobs can share a time-slot and be scheduled in priority order within it.

Consider a stage j that is a partitioned resource. Let job J_i be allocated a slice that is served for B_{slice} time units every B_{total} time units. As shown in Figure 2, this is no worse than having a dedicated resource that is slower by a factor B_{slice}/B_{total} and that introduces an access delay of at most $B_{total} - B_{slice}$.

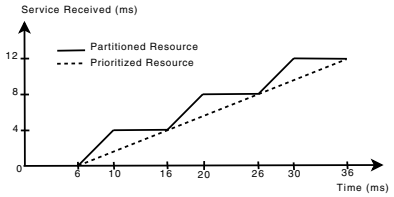

Figure 2: Illustration of conversion of a partitioned resource into a prioritized resource.

Figure 2 illustrates the service received by a set of tasks over time for the original partitioned resource and for its corresponding dedicated prioritized resource, when $B_{slice} = 4ms$ and $B_{total} = 10ms$. Note that the service received under the prioritized resource will always be less than in the partitioned resource, causing tasks to be delayed longer. Hence, this transformation is safe in that if the tasks in the transformed system are schedulable, so are the tasks in the original system.

When analyzing the end-to-end delay of J_1, the computation time of J_1 on the new prioritized resource j can be taken as $C_{1,j} \times \frac{B_{total}}{B_{slice}} + (B_{total} - B_{slice})$ (the additional delay is subsumed in the computation time). The computation time of all other jobs in the same slice would be $C_{i,j} \times \frac{B_{total}}{B_{slice}}$.

Once this transformation is conducted for all partitioned resources that J_1 encounters in the system, the delay composition theorem can be directly applied to compute the worst case end-to-end delay of J_1.

5. Transforming Distributed Systems

The preemptive and non-preemptive DAG delay composition theorems derived in Section 3, can be used to reduce a given distributed acyclic system to an equivalent single stage system. Let S_{wc} denote the worst-case set of jobs that can potentially delay J_1. A simple, and somewhat conservative definition of S_{wc}, similar to the definition in [6] is given by:

Definition: The worst-case set S_{wc} of all jobs that delay 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, $[A_1, A_1 + Delay(J_1)]$.

Note that the above definition is recursive, in the sense that it defines the set of jobs that delay J_1 in terms of the delay experienced by J_1. This can be resolved by starting with a small estimate for the delay of J_1 and iteratively adding jobs and recomputing the delay, until all jobs J_i whose intervals $[A_i, A_i + D_i]$ overlap $[A_1, A_1 + Delay(J_1)]$ have been included. Alternatively, a quick, but more pessimistic definition of S_{wc} could include all jobs J_i whose intervals $[A_i, A_i + D_i]$ overlap $[A_1, A_1 + D_1]$.

In Sections 5.1 and 5.2, we show how an equivalent uniprocessor system can be created to analyze schedulability of the original system under preemptive and non-preemptive scheduling, respectively. When the system consists of partitioned resources, we assume that the transformation de-

scribed in Section 4 has already been performed. In Section 5.3, we present DAG schedulability expressions for deadline monotonic scheduling based on the above task set reduction.

5.1 Preemptive Scheduling Transformation

The form of the DAG delay composition theorem suggests a reduction to a uniprocessor system in which the lowest-priority uniprocessor job suffers the delay stated by the theorem. This reduction to a single stage system is conducted by (i) replacing each higher priority job J_i in $\bar{S}_{wc}$ by a single stage job J_i^* of execution time equal to $2C_{i,max}M_i$, and (ii) replacing J_1 with a lowest-priority job, J_1^* of execution time equal to $2C_{1,max} + \sum_{j \in Path_1, j \leq N-1} \max_i(C_{i,j})$ (the second term is the stage-additive component), and deadline same as that of J_1. The delay of J_1^* on the hypothetical uniprocessor adds up to the delay bound as expressed in the right hand side of Inequality 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_1^* on the uniprocessor. Thus, if J_1^* completes prior to its deadline in the uniprocessor, so will J_1 in the acyclic distributed system.

For the case of periodic tasks, we can use Corollary 1 to reduce the system to a uniprocessor system. In a uniprocessor system, the delay of the lowest priority task invocation in the presence of higher priority tasks T_i (collectively denoted by R_u) with computation times C_i is given by:

$$Delay_{uniprocessor}(T_1) = \sum_{T_i \in R_u} x_i C_i \tag{8}$$

where x_i is the number of invocations of T_i that delay T_1. The terms x_i are typically determined by a uniprocessor schedulability analysis technique. The delay of J_1 in the DAG, as per Corollary 1, can be written in the same form as Equation 8, leading to a natural reduction of the DAG system to a uniprocessor system.

$$Delay(J_1) \leq \sum_{T_i \in R} x_i \times 2C_{i,max} + 1 \times \Big(\sum_{T_i \in R} 2C_{i,max}M_i + \sum_{\substack{j \in Path_1 \\ j \leq N-1}} \max_{T_i \in R}(C_{i,j}) \Big)$$

The reduction to a single stage system for periodic tasks can then be conducted by (i) replacing each higher priority periodic task T_i by an equivalent single stage task with execution time $C_i^* = 2C_{i,max}$ and having the same period and deadline as T_i (the number of invocations x_i will be determined by the uniprocessor schedulability analysis), and (ii) replacing T_1 with lowest priority task, T_1^* with computation time $C_1^* = 2C_{1,max} + \sum_i 2C_{i,max}M_i + \sum_{j \in Path_1, j \leq N-1} \max_i(C_{i,j})$ (similar to the reduction of the pipelined system as in [6]) with same period and deadline as T_1. If task T_1^* is schedulable on a uniprocessor, so is T_1 on the original acyclic distributed system.

5.2 Non-Preemptive Scheduling Transformation

Under non-preemptive scheduling, we reduce the DAG into an equivalent single stage system that runs *preemptive* scheduling as before. This is achieved by (i) replacing each job J_i in $\bar{S}_{wc}$ by a single stage job J_i^* of execution time equal to $C_{i,max}M_i$, and (ii) replacing J_1 by a lowest-priority job, J_1^* of execution time equal to $C_{1,max} + \sum_{j \in Path_1, j \leq N-1} \max_{J_i \in \bar{S}_{wc}}(C_{i,j}) + \sum_{j \in Path_1} \max_{J_i \in \underline{S}}(C_{i,j})$ (which are the last two terms in Inequality (5)), and deadline same as that of J_1. Note that the execution time of J_1^* includes the delay due to all lower priority tasks. Further, in the above reduction, the hypothetical single stage system constructed is scheduled using preemptive scheduling, while the original DAG was scheduled using non-preemptive scheduling. This is because we only care to match the sum of the delay experiences by J_1 and J_1^* in their respective systems. By the delay composition theorem, the total delay incurred by J_1 in the acyclic distributed system under non-preemptive scheduling is no larger than the delay of J_1^* on the uniprocessor under preemptive scheduling, since the latter adds up to the delay bound expressed on the right hand of Inequality (5).

For the case of periodic tasks, the delay bound in Corollary 2 can be used. 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 T_i^* of computation time $C_i^* = C_{i,max}$ and same period and deadline as T_i, and (ii) replacing T_1 with a lowest priority task, T_1^* with computation time $C_1^* = C_{1,max} + \sum_{T_i \in \bar{R}} C_{i,max}M_i + \sum_{j \in Path_1, j \leq N-1} \max_i(C_{i,j}) + \sum_{j \in Path_1} \max_{T_i \in \underline{R}}(C_{i,j})$ with same period and deadline as T_1.

If task T_1^* is schedulable using preemptive scheduling on a uniprocessor, so is T_1 on the original acyclic distributed system under non-preemptive scheduling.

5.3 Examples of Equivalent Uniprocessor Schedulability Analysis

The reduction described in the previous subsections enables large complex acyclic distributed systems to be easily analyzed using any single stage schedulability analysis technique. For this reason, we call our solution a 'meta-schedulability test'. The only assumptions made by the reduction on the scheduling model are fixed priority preemptive scheduling, and that tasks do not block for resources on any of the stages (i.e., independent tasks). In this section, we show how the Liu and Layland bound [9] and the necessary and sufficient test based on response time analysis [1] can be applied to analyze periodic tasks in an acyclic distributed system, under both preemptive and non-preemptive scheduling. Other uniprocessor schedulability tests can be applied in a similar manner.

Define $C_{i,max}$ as the largest execution time of T_i on any stage, D_i as the end-to-end deadline, and n as the number of periodic tasks in the system. Let $M_{k,i}$ be the number of times the paths of T_k and T_i meet for one or more consecutive common stages.

For preemptive scheduling, $C_k^* = 2C_{k,max}$; $C_e^*(i) = 2C_{i,max} + \sum_{k>i} 2C_{k,max}M_{k,i} + \sum_{j \in Path_1, j \leq N-1} \max_{k \geq i}(C_{k,j})$.

For non-preemptive scheduling, $C_k^* = C_{k,max}$;

$C_e^*(i) = C_{i,max} + \sum_{k \geq i} C_{k,max} M_{k,i} + \sum_{j \in Path_i, j \leq N-1} \max_{1 \leq k \leq n}(C_{k,j}) + \sum_{j \in Path_i} \max_{k<i} C_{k,j}$.

The Liu and Layland bound [9], applied to periodic tasks in an acyclic distributed system is:

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

for each $i, 1 \leq i \leq n$.

Our meta-schedulability test, when used together with the necessary and sufficient test for schedulability of periodic tasks under fixed priority scheduling proposed in [1], will have the following recursive formula for the worst case response time R_i of task T_i:

$$R_i^{(0)} = C_e^*(i); \quad R_i^{(k)} = C_e^*(i) + \sum_{j>i} \left\lceil \frac{R_i^{(k-1)}}{P_j} \right\rceil C_j^*$$

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 should be at most D_i.

6. Handling Tasks whose Sub-Tasks Form a DAG

In the discussion so far, we have only considered tasks whose sub-tasks form a *path* in the Directed Acyclic Graph. In this section, we describe how this can be extended to tasks whose sub-tasks themselves form a DAG. We shall refer to such tasks as DAG-tasks. Figure 3(a) shows an example task, whose sub-tasks form a DAG. Edges in the DAG, as before, indicate precedence constraints between sub-tasks and each sub-task executes on a different resource. A sub-task s can execute only after all sub-tasks which have edges to sub-task s have completed execution. In the task shown in the figure, sub-task 5 can execute only after sub-tasks 2 and 3 have completed execution. We call this a 'merger' of sub-tasks. Note that a split, that is, edges from one sub-task s to two or more sub-tasks indicate that once sub-task s completes, it spawns multiple sub-tasks each executing in parallel. It can be observed from the example in Figure 3(a), that once sub-task 1 completes, it spawns sub-tasks 2 and 3 that can execute in parallel on different stages.

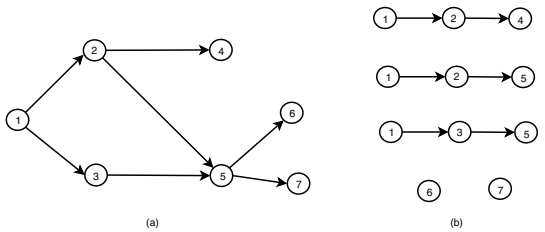

Figure 3: (a) Figure showing an example of a DAG-task (b) Different parts of the DAG-task that need to be separately analyzed to analyze schedulability of the DAG-task.

As the delay composition theorem only addresses tasks which execute in sequential stages (that is, the sub-tasks form a path in the DAG) and does not consider DAG-tasks, we need to break the DAG-task into smaller tasks which form a path of the DAG. This is carried forth as follows. Similar to traditional distributed system scheduling, artificial deadlines are introduced after each merger of sub-tasks. Each split in the DAG creates additional paths that need to be analyzed (the number of additional paths is one less than the fan-out). In the example DAG-task, an artificial deadline is imposed after sub-task 5. Sub-tasks 6 and 7 are analyzed independently using any single stage schedulability test. As there are two splits within sub-tasks 1 through 5, there are 3 paths that need to be analyzed as shown in Figure 3(b). The path 1-2-4 is analyzed independently using the meta-schedulability test and this sequence of sub-tasks need to complete within the end-to-end deadline of the DAG-task. The paths 1-2-5 and 1-3-5 can be independently analyzed using the meta-schedulability test, with their deadline set as the artificial deadline. Sub-tasks 6 and 7 need to complete in a duration at most equal to the end-to-end deadline of the DAG-task minus the artificial deadline set for sub-task 5. If all the parts of the DAG-task are determined to be schedulable, then the DAG task is deemed to be schedulable.

As observed in [6], imposing artificial deadlines add to the pessimism of the schedulability analysis. The use of the delay composition theorem reduces the need to impose artificial deadlines to only stages in the execution where two or more sub-tasks merge. This is in contrast to traditional distributed schedulability analysis, that imposes artificial deadlines after each stage of execution, causing the pessimism to quickly increase with system scale.

7. Simulation Results

In this section, we evaluate the preemptive and non-preemptive schedulability analysis techniques described in Section 5.3. A custom-built simulator that models a distributed system with directed acyclic flows is used. Due to paucity of space, we consider only periodic tasks, and further assume that partitioned resources within the system have been transformed into resources scheduled in priority order as described in Section 4, and focus this evaluation on prioritized resources. An admission controller is used to maintain real-time guarantees within the system. The admission controller is based on a single stage schedulability test for deadline monotonic scheduling, such as the Liu and Layland bound [9] or response time analysis [1], together with our reduction of the multistage distributed system to a single stage, as shown in Section 5.3. Each periodic task that arrives at the system is tentatively added to the set of all tasks. The new task is admitted if the task set is found to be schedulable by the admission controller, and dropped if not.

Although the meta schedulability test 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 represents average utilization values obtained from 100 executions of the simulator, with each execution running for 80000 task invocations. When

comparing different admission controllers, each admission controller was allowed to execute on the same 100 task sets.

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 (we do not consider DAG tasks in this evaluation), with each node in the distributed system having a probability of NP (for Node Probability) of being selected as part of the route. The task's route is simply the sequence of selected nodes in increasing order of their node identifier. The default value of NP is chosen as 0.8. End-to-end deadlines (equal to the periods, unless explicitly specified otherwise) of tasks are chosen as $10^x a$ simulation seconds, where x is uniformly varying between 0 and DR (for deadline ratio parameter), 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 to the shortest task deadline to be as large as 10^{DR}. The default value for DR is 0.5. The execution time for each task on each stage was chosen based on the task resolution parameter, which is the ratio of the total computation time of a task over all stages to its end-to-end 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 $T < 1$ is the task resolution. We used a task resolution of $1/100$. The stage execution times of tasks were allowed to vary up to 10% on either side of the mean. Choosing the stage execution times to be nearly proportional to the end-to-end deadline, ensures that when tasks have similar deadlines (DR close to zero), then the execution times are also comparable. When tasks have widely different deadlines (a high value for DR), then the execution times are also widely varying. Our simulations show that non-preemptive scheduling performs better than preemptive scheduling when the task execution times are similar, and preemptive scheduling performs better than non-preemptive scheduling when the task execution times are different by more than two orders of magnitude.

Under preemptive scheduling, task preemptions are assumed to be instantaneous, that is, the task switching time is zero. The default single stage schedulability test used is the response-time analysis technique presented in [1]. 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.

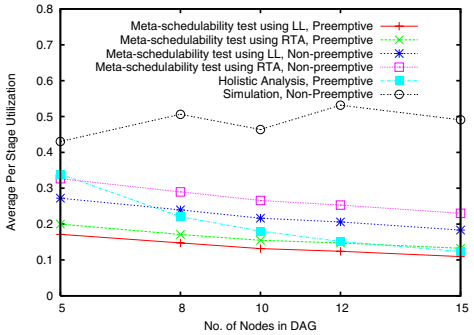

Figure 4: Meta-schedulability test vs. holistic analysis for different number of nodes in DAG

We first study the achievable utilization of our meta-schedulability test using both the Liu and Layland bound and response time analysis, for both preemptive as well as non-preemptive scheduling. We compare this with holistic analysis [13], applied to preemptive scheduling, for different number of nodes in the DAG, the results of which are shown in Figure 4. While extensions to holistic analysis have been proposed (such as [11]), we use holistic analysis as a comparison as these extensions are targeted to handle offsets (we do not consider offsets in our analysis). Further, they suffer from similar drawbacks as holistic analysis such as poor scalability and requiring global knowledge of all tasks in the system. For meta-schedulability test curves that are marked preemptive, the scheduling was preemptive and the preemptive version of the test was used in admission control. Likewise, for the meta-schedulability test curves that are marked non-preemptive, the scheduling was non-preemptive and the non-preemptive version of the test was used. We only evaluated holistic analysis applied to preemptive scheduling as presented in [13], as the non-preemptive version presented in [8] adds an extra term to account for blocking due to lower priority tasks and tends to be more pessimistic than the preemptive version, and the corresponding curve would always be lower than the curve for preemptive scheduling.

It can be observed from Figure 4, that even for an eight node DAG, non-preemptive scheduling analyzed using our meta-schedulability test significantly outperforms preemptive scheduling analyzed using both holistic analysis and our meta-schedulability test. As the utilization curve for holistic analysis applied to non-preemptive scheduling would be lower than the curve for the preemptive scheduling version of holistic analysis, non-preemptive scheduling analyzed using our meta-schedulability test would also outperform the non-preemptive version of holistic analysis. A drawback of holistic analysis is that it analyzes each stage separately assuming the response times of tasks on the previous stage to be the jitter for the next stage. It therefore assumes that every higher priority job will delay the lower priority job at every stage of its execution, ignoring possible pipelining between the executions of the higher and lower priority jobs. This causes holistic analysis to become increasingly pessimistic with system size when periods are of the order of end-to-end deadlines (as opposed to per-stage deadlines). As motivated in [7], preemption can reduce the overlap in the execution of jobs on different stages, resulting in non-preemptive scheduling performing better than preemptive scheduling in the worst case.

In order to estimate when deadlines are actually being missed, and to evaluate the pessimism of the admission controllers, we conducted simulations to identify the lowest utilization at which deadlines are missed. The curve labeled 'Simulation' in Figure 4 presents the results from simulations of the lowest utilization at which deadline misses were observed for different number of nodes in the system when non-preemptive scheduling was employed. The corresponding curve for preemptive scheduling, was within 0.02 of those of non-preemptive scheduling, and we don't show the values

here for the sake of clarity (the reader must bear in mind that task sets were generated randomly, and that the task sets do not represent worst case scenarios). Each point for the simulation curve was obtained from 500 executions of the simulator in the absence of any admission controller, with each execution considering a workload with utilization close to where deadline misses were being observed. We observe that the meta-schedulability test curves degrade only marginally with increasing scale, while the performance of holistic analysis degrades more rapidly.

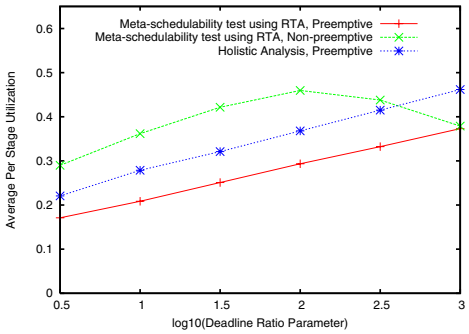

Figure 5: Meta-schedulability test vs. holistic analysis for different deadline ratio parameters

To precisely evaluate the scenarios under which non-preemptive scheduling performs better than preemptive scheduling in distributed systems, we conducted experiments varying the deadline ratio parameter (DR) while keeping the other parameters equal to their default values. Figure 5 plots a comparison of the meta-schedulability test under both preemptive as well as non-preemptive scheduling, with holistic analysis for different DR values ranging between 0.5 and 3.0. A DR value of x indicates that the end-to-end deadlines of tasks can differ by as much as 10^x. As stage execution times are chosen proportional to the end-to-end deadline, when the end-to-end deadlines of tasks are widely different, the lower priority tasks (those with large deadlines) have a large stage execution time. Initially, as DR increases, the utilization for both preemptive as well as non-preemptive scheduling increases, as lower priority tasks can execute in the background of higher priority tasks resulting in better system utilization. Up to $DR = 2$, non-preemptive scheduling (together with the non-preemptive version of the meta-schedulability test) results in better performance than preemptive scheduling (together with the preemptive version of the test). However, for values of DR greater than 2, that is, the end-to-end deadlines vary by over two orders of magnitude, preemptive scheduling performs better than non-preemptive scheduling. The achievable utilization under non-preemptive scheduling decrease beyond a DR value of 2, as higher priority tasks can now be blocked for a longer duration under non-preemptive scheduling, leading to a greater likelihood of deadline misses.

The above results have all been obtained by setting the end-to-end deadlines equal to the periods of tasks. Figure 6 plots a comparison of the meta-schedulability test under preemptive and non-preemptive scheduling with holistic

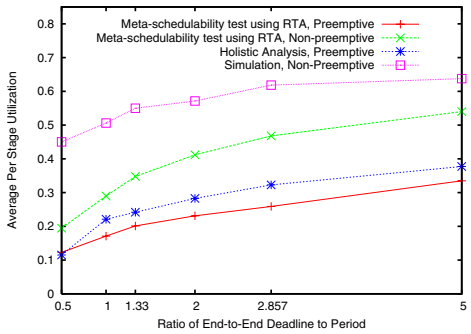

Figure 6: Meta-schedulability test vs. holistic analysis for different ratios of end-to-end deadline to task periods

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 utilization of all the three analysis techniques are high. The meta-schedulability test under non-preemptive scheduling consistently outperforms preemptive scheduling analyzed using either the meta-schedulability test or holistic analysis. As holistic analysis applied to non-preemptive scheduling (curve not shown) would perform worse than the preemptive scheduling version of holistic analysis, it would also perform worse than the meta-schedulability test applied to non-preemptive scheduling. Similar to Figure 4, the curve labeled as 'simulation' plots the lowest utilization at which deadline misses were observed obtained from simulations under non-preemptive scheduling in the absence of any admission controller. The corresponding values for preemptive scheduling were close to those obtained for non-preemptive scheduling and are not presented here for the sake of clarity. We observe that our analysis tends to be less pessimistic for larger values of the ratio between the end-to-end deadline and the period.

8. Related Work

Algorithms for statically scheduling precedence constrained tasks in distributed systems have been proposed in [4, 14]. Such algorithms construct a schedule of length equal to the least common multiple of the task periods of the set of periodic tasks. This schedule can then be used to accurately specify the time intervals during which each task will be executed. Such algorithms have a huge time complexity and are clearly unsuitable for large, complex distributed systems.

Pipelined distributed systems have been studied in the context of job-fair scheduling, where the objective is to find a feasible schedule of executing the tasks. In [2], polynomial time algorithms are presented for special cases where the problem is tractable, and heuristic solutions are developed for the general case. In contrast, we study the problem of schedulability of an arbitrary set of priority ordered tasks under a given scheduling policy.

A few offline schedulability tests have been proposed, which divide the end-to-end deadline into individual stage deadlines, and tend to ignore the overlap between the execution of different stages. In [10, 12], offset-based response time analysis techniques for EDF were proposed, which di-

vide the end-to-end deadline into individual stage deadlines.

Holistic schedulability analysis for distributed hard real-time systems [13], assumes the worst case delay at a stage as the jitter for the next stage. While this technique does not divide the end-to-end deadline into sub-deadlines for individual stages, it nevertheless does not account for the overlap in the execution of different pipeline stages.

In [5], a schedulability test based on aperiodic scheduling theory for fixed priority scheduling was derived. 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. In earlier publications [6, 7], we proved a delay composition theorem for pipelined systems under both preemptive and non-preemptive scheduling. This paper extends these results to more general distributed systems and partitioned resources.

In stark contrast to preemptive scheduling, non-preemptive scheduling has received very little attention from the real-time community. An extension to holistic analysis in distributed systems to account for blocking due to non-preemptive scheduling is presented in [8]. The paper presents a comparison of this analysis technique with network calculus [3], and concludes that the worst case response time as predicted by the holistic analysis technique tends to be superior to that of network calculus in most cases. In contrast to such techniques, we reduce the problem of analyzing a distributed acyclic system with non-preemptive scheduling to that of analyzing a single stage system using preemptive scheduling. Thus, well known uniprocessor tests can be adopted to analyze multistage systems, resulting in more efficient schedulability analysis.

9. Discussion and Future Work

A key step in deriving the DAG delay composition theorems was to split each higher priority job J_i into M_i sub-jobs J_{i_k}, each executing on one or more consecutive common stages with J_1. The precedence constraints in the arrival times of the different sub-jobs can be relaxed by assuming that each sub-job arrives independently of the others. This independence assumption can only result in a more pessimistic delay analysis for J_1. The same transformation of splitting jobs into sub-jobs and assuming independent arrivals for sub-jobs, can also be conducted for non-acyclic higher priority jobs (jobs that visit a stage more than once). Each visit to a stage can be considered as an independent arrival of a sub-job. In the case where J_1 (the job under consideration) itself has loops in its path, then J_1 can be split into sub-jobs each of which is acyclic. The DAG delay composition theorem can then be used to determine the worst-case delay for each sub-job, and the worst-case delay for J_1 can be estimated as the sum of the worst-case delays of each of its sub-jobs. Even with only one loop in the task path, there may be multiple ways in which the loop can be broken. For example, suppose that a task traverses stages 1, 2, 3, and then revisits stage 1. This loop 1-2-3-1 can be broken as either (1, 2-3-1) (one sub-job on stage 1 and another that executes along the path 2-3-1), (1-2, 3-1), or (1-2-3, 1). This choice becomes an art of design, and the choice that maximizes pipelining and minimizes the number of independent sub-jobs would typically yield the best delay bound. A better characterization of the precedence constraints between sub-jobs (instead of assuming them to be independent) could yield a more accurate delay bound for non-acyclic task systems.

The delay composition rule derived in this paper could aid the study of obtaining optimal rate control, routing and scheduling policies in distributed systems and large networks. Extensions to allow both preemptive and non-preemptive scheduling to be employed within the same system, will also be useful.

10. Conclusion

In this paper, we present a delay composition theorem that bounds the worst-case delay of jobs for preemptive and non-preemptive scheduling in distributed systems, where the routes of tasks form a directed acyclic graph. We consider systems where resources can be either partitioned or scheduled in priority order. We also show a simple extension of the results to non-acyclic task sets. The composition rule leads to the reduction of the distributed system to an equivalent single stage system, which then enables any single stage schedulability test to analyze distributed systems. We show that under certain conditions, non-preemptive scheduling can perform better than preemptive scheduling for distributed systems. We believe the results derived in this paper will serve as a first step towards a general transformation theory for distributed systems.

References

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

[2] R. Bettati and J. W. Liu. Algorithms for end-to-end scheduling to meet deadlines. In *IEEE Symposium on Parallel and Distributed Processing*, pages 62–67, December 1990.

[3] R. Cruz. A calculus for network delay, part ii: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

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

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

[6] P. Jayachandran and T. Abdelzaher. A delay composition theorem for real-time pipelines. In *ECRTS*, pages 29–38, July 2007.

[7] P. Jayachandran and T. Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. *Invited to Real-Time Systems Journal: Special Issue on ECRTS'07 (to appear)*, 2008.

[8] A. Koubaa and Y.-Q. Song. Evaluation and improvement of response time bounds for real-time applications under non-preemptive fixed priority scheduling. *International Journal of Production and Research*, 42(14):2899–2913, July 2004.

[9] 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.

[10] 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.

[11] J. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *IEEE Real-Time Systems Symposium*, pages 26–37, December 1998.

[12] R. Pellizzoni and G. Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *RTAS*, pages 66–75, March 2005.

[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.

A Derivation of Delay Bound Using the Pipeline Result

In the proof of the preemptive DAG delay composition theorem in Section 3.1, we used the pipeline delay composition theorem [6] to derive the delay bound. The pipeline result was proved assuming that all jobs follow the same sequence of stages. However, in the system under consideration, each sub-job J_{i_k} of J_i executes only on a certain consecutive sequence of stages j through j' (say) and does not execute on the other stages. We now show that the delay bound of a job J_1 as per the pipeline delay composition theorem also bounds its delay in the presence of higher priority jobs that execute at an arbitrary sequence of stages j through j' ($1 \leq j \leq j' \leq N$) before exiting the system.

For notational simplicity, let us renumber all higher-priority jobs J_{i_k} so they are given a single index increasing in priority order, and let $\bar{Q}$ denote the set of all such jobs including J_1. Further (also for notational simplicity), let us assume that each job has a unique priority. Ties are broken arbitrarily (e.g., in a FIFO manner).

Lemma 1. *The pipeline delay composition theorem [6] provides a worst-case delay bound for job J_1 in the presence of higher priority jobs (denoted by set $\bar{Q}$ with the inclusion of J_1), each executing on some arbitrary consecutive sequence of stages in the path of J_1.*

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

Proof. The proof is by induction on task priority. While carrying out the induction, we also successively transform each added task, so that it executes on all stages 1 through N with zero execution times on stages on which it did not execute previously. We show that this transformation does not invalidate the delay bound as per the lemma.

The basis step of the lemma is when only J_1 is present in the system. In this case,

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

which is trivially true.

Now, assume that the lemma is true for $k-1$ jobs, $k \geq 2$. We shall prove the lemma when a k^{th} job J_k of highest priority is added. To do so, we need to show that the additional delay due to the presence of J_k is at most $2C_{k,max}$, in addition to J_k's contribution to the stage additive component of the delay (the sum of maximum computation times over all jobs at each stage).

Let J_k execute between stages j and j' in the path of J_1. By adding a zero execution time requirement for J_k on each stage beyond j' in the path of J_1, we do not change the execution intervals or the end-to-end delay of J_1. Now in the system with only $k-1$ jobs, in the absence of J_k, let the delay of J_1 from the time of its arrival till the time it arrives at stage j be x, and the delay from the time it arrives at stage j till the time it completes its execution in the system be y. The end-to-end delay of J_1 is thus $x+y$, when the system has $k-1$ jobs. In the system with job J_k, let the delay of J_1 from the time it arrives at stage j till the time it completes execution on all stages be $y+\Delta$ (Δ is the additional delay caused by J_k).

Consider the system starting from stage j and including all subsequent stages in the path of J_1. All k jobs execute on all the stages (the transformation has been performed for the other $k-1$ jobs), and the system is a pipelined system. We can now apply the pipeline delay composition theorem [6] to this system. From the pipeline delay composition theorem, the worst case delay that J_k can induce J_1, that is the maximum value for Δ, is $2C_{k,max}$, in addition to J_k contributing to the stage additive component of the delay, which is one maximum stage execution time over all jobs for each stage. Thus, Δ is bounded regardless of the value of x and y and the arrival times of the other jobs. Now, add a zero execution time requirement for J_k on each stage prior to stage j on the path of J_1. As J_k is the highest priority job in the system, as soon as it arrives to the system it would complete its zero execution time requirement on each stage and arrive at stage j instantaneously. Thus, when zero execution time requirements have been added for J_k on stages prior to stage j and beyond stage j', the delay that J_k causes J_1 is still Δ, which is bounded as described above regardless of the arrival times of the other jobs. This proves the induction step and each higher priority job J_k inflicts a delay of at most $2C_{k,max}$ in addition to contributing to the stage additive component.

The lemma is precisely Inequality 2 in Section 3.1. The same proof applies even for the case of non-preemptive scheduling, except for invoking the non-preemptive pipeline delay composition theorem [7] instead of the preemptive version of the theorem. Thus, under non-preemptive scheduling, Δ is bounded by $C_{k,max}$ (one maximum stage execution time for each higher priority job instead of two), and an additional blocking term determines the delay due to lower priority jobs as shown in Section 3.2. □