On Schedulability and Time Composability of Multisensor Data Aggregation Networks

Fatemeh Saremi
Department of Computer Science
University of Illinois
Urbana, IL 61801
Email: saremi1@illinois.edu

Praveen Jayachandran
IBM Research
India
Email: prjayach@in.ibm.com

Forrest Iandola, Yusuf Sarwar, Tarek Abdelzaher
Department of Computer Science
University of Illinois
Urbana, IL 61801
Email: {iandola1, mduddin2, zaher}@illinois.edu

***Abstract*—This paper develops a framework to analyze the latency and delay composition of workflows in a real-time networked aggregation system. These workflows are characterized by different inputs that are processed along parallel branches that eventually merge or fuse to compute the aggregation result. The results for each flow must be produced within certain end-to-end deadlines or else the information would become stale and useless. We consider an end-to-end view of the aggregation system that allows us to derive a much tighter analysis of the end-to-end delay compared to traditional analysis techniques. The framework extends results developed by the authors recently to analyze end-to-end latency of various workflow topologies. We then provide a reduction of the aggregation network system to an equivalent hypothetical uniprocessor for the purposes of schedulability analysis. Extensive simulations show that latency bound obtained from the analysis framework is significantly more accurate than that of traditional analysis techniques.**

Keywords: Data Aggregation, Delay, Composition, Schedulability.

I. Introduction

Multisensor data aggregation is a discipline concerned with collecting and processing data from multiple sensor sources in order to produce information that is accurate, specific, and relevant to the users of the system. Automated target recognition, battlefield surveillance, remote sensing, ocean surveillance, robotics, medical diagnosis, condition-based maintenance, and environmental monitoring are all examples of applications of data aggregation systems. An overarching challenge in such systems is in maintaining the *relevance* of information that is obtained by the process of aggregating together multiple sensory inputs. Oftentimes, the data collected is real-time in nature, and the processing needs to complete within certain time constraints. For instance, in defense surveillance systems, detection, tracking and identification of the threat have to complete within stringent deadline constraints in order to enable timely action to counter the threat.

The aggregation workflows we consider in this paper are characterized by different sensory inputs processed along parallel branches that merge together to produce the aggregation results. When parallel branches merge together, the aggregation results can be computed only when *all* incoming branches have completed execution. A workflow could have multiple such successive merges. The aggregation results of individual jobs of workflows need to be computed within certain end-to-end deadlines. The deadline represents the maximum delay that can be tolerated from the time the inputs arrive at the various branches till the time the final aggregation results are calculated. This system model presents a deviation from existing literature and presents a new “MERGE” primitive that is characteristic of aggregation systems.

In this paper, we investigate timing composability and schedulability of workflows in such multisensor data aggregation systems. We develop an analysis framework which extends recent work by the authors in developing a delay composition algebra [1], where distributed system stages are iteratively composed into a single stage. Well known uniprocessor analysis techniques can then be applied on this hypothetical uniprocessor, in order to infer end-to-end delay and schedulability properties of jobs in the original distributed system. The framework was designed for jobs that traverse a sequence of resource stages in a distributed system and is not directly applicable to aggregation systems characterized by the “MERGE” primitive described above.

Our framework develops theoretical rules which determine how latency in data aggregation systems is composed under preemptive and non-preemptive scheduling. Unlike traditional analysis techniques such as holistic analysis [2] [3] and network calculus [4] [5] that analyze the delay one stage at a time, delay composition framework considers an end-to-end view of the system. This allows us to develop intuition into the main system and task parameters that contribute towards increasing the end-to-end delay, and to express our end-to-end delay bound in terms of these parameters. This results in a much tighter analysis of the end-to-end delay and schedulability of jobs compared to existing analysis techniques, especially for large systems.

We evaluate our theoretical framework through extensive simulations based on two related metrics. The first is the capacity or the maximum workload due to jobs of workflows that can be guaranteed to meet their end-to-end deadlines. This is important in ensuring that the system operates at its capacity limit without wasting resources serving jobs that do not meet their end-to-end deadlines. The second metric measures the ratio of the average end-to-end delay of jobs of a workflow to the analytically computed worst-case end-to-end delay bound. This gives us a measure of the tightness of the analysis.

The rest of this paper is organized as follows. Section II reviews related work. In Section III we describe the system model and the state the main problem addressed in this paper. In Section IV, we present examples to explain why seemingly intuitive approaches to calculate the worst-case latency are not accurate for aggregation systems. These examples also serve to provide intuition into the right approach for determining the worst-case latency of individual jobs of a workflow based on the arrival and computation time characteristics of other concurrent workflows. We then proceed to formally derive the delay bound under both preemptive and non-preemptive scheduling in Section V. Section VI presents reduction rules that can be employed to transform the entire data aggregation system into a single hypothetical stage for the purposes of schedulability analysis. Then the performance of the analysis framework is extensively evaluated using simulations in section VII. Finally, Section VIII concludes the paper.

II. Related Work

Existing data aggregation schedulability literature often focuses on a specialized system model and a specific scheduling algorithm. For example, Li and Cao proposed a non-preemptive scheduling algorithm for data aggregation systems called Coordinated Workload Scheduling (CWS) [6]. They also analyze schedulability, but the schedulability results in that study only apply when a particular scheduling algorithm is used. CWS constrains the system model to contain just three workflows: sensing, communication, and computation. In the same vain, in an earlier paper we developed the real-time capacity of data aggregation systems that are scheduled under EDF [**?**]. In that study, our bound only handled data aggregation systems with a single merge.

There also exist several studies that are applicable to more general data aggregation system models. For example, the real-time systems community has developed several algorithms to perform optimal offline scheduling of workflow sets in (general) distributed systems [7] [8]. These algorithms work by constructing the complete schedule for all the jobs executing on all the stages. The schedule is then used to determine the the schedulability of the workflow set. However, these optimal algorithms require NP-hard computation, which is a major drawback especially when dealing with large real-time systems. In addition, workflow sets involving aperiodic or sporadic workflows make offline scheduling even less feasible.

To reduce the computational complexity of schedulability testing, Kao et al. [9] and Zhang et al. [10] presented techniques to divide the end-to-end deadline into per-stage deadlines. Uniprocessor schedulability tests are then used to determine whether each stage is schedulable. If all the stages are schedulable, the system is deemed to be schedulable. These techniques allow for a generalized system model and are not constrained to a particular scheduling algorithm. However, they do not accurately account for the inherent parallelism in the execution of different stages of a data aggregation system. As a result, they tend to exhibit extreme pessimism when estimating schedulability boundaries for pipelined and merging workflows, especially for large systems.

Holistic analysis [2] [11] [12] and real-time calculus [13] [14] [15] comprise a "middle ground" between pessimistic bounds and NP-hard optimal calculations. Unlike some of the related work we have discussed, holistic analysis and network calculus allow for a very general system model. A study by Koubaa and Song found that holistic analysis is less pessimistic than network calculus for most system configurations [3]. For this reason, we use holistic analysis as a benchmark for our evaluation. Yet, as the holistic approach analyzes one stage at a time, the pessimism of the analysis grows with system scale. In contrast, in this paper we develop a delay bound for aggregation systems by considering an end-to-end view of the delay. This allows us to develop a bound that does not become increasingly pessimistic with system scale. Further, by reducing the problem of schedulability analysis of the distributed system to that on an equivalent uniprocessor, we significantly reduce the complexity of analysis, making it extremely suitable for large systems.

The related works have one or more of the following shortcomings: NP-hard computational complexity, pessimism, constrained aggregation system models, or constrained scheduling algorithms. In contrast to these problems, the work presented in this paper applies to a general aggregation system model, is simple to compute even for large systems, and is less pessimistic than holistic analysis for a wide range of workload configurations.

III. Data Aggregation Model and Problem Statement

In this paper, we consider an abstract model of a data aggregation system, comprising of m data workflows (e.g., audio and video data from camera and microphone sensors, speed and proximity sensor information in a next-generation automobile), each requiring several stages of processing (such as monitoring, threat analysis, actuation and display). Each processing stage is handled by a single resource that is scheduled in priority order. For instance, this could refer to processors that schedule and serve arriving tasks, or a network link on which enqueued packets are transmitted. Since each stage represents one resource, in the rest of this paper, we use the terms *stage* and *resource* interchangeably. Each workflow could potentially have multiple branches that ultimately merge forming an aggregation tree. Collectively, the stages of processing of all workflows taken together are organized in a graph called the *workflow graph*. It determines precedence constraints among the different stages. For simplicity, we only consider workflow graphs that are trees (no two task paths can split and re-merge with one another).

Consider a flow F_i that has L parallel pipelines, described by disjoint paths p_1, ..., p_L, followed by a common parent stage, P, as shown in Figure 1.

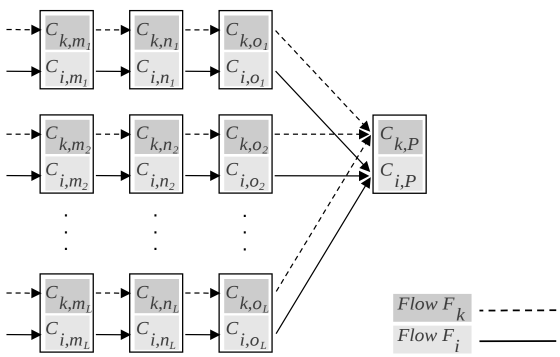

Figure 1. An example of merging workflows.

Jobs from different workflows are assumed to be aperiodic and may have different arrival offsets. Let Off_i denote the offset from time zero at which a job J_i of workflow F_i arrives at all the branches of its workflow (if different branches arrive at different offsets, Off_i can be set to the maximum of these offsets). A job does not become eligible to execute on the merge-stage (the common parent, P) until all pipelines have finished processing it. A workflow could potentially have several such merge-segments.

In a *real-time* data aggregation system, each workflow, F_i, has an end-to-end latency constraint, D_i, denoting the maximum allowable latency between the arrival of a new job of workflow F_i into the system (to all its branches), and the completion of its processing on the last stage. We call this constraint the *end-to-end deadline* of workflow F_i.

Each job of every workflow is assigned a priority and we derive results for both preemptive and non-preemptive scheduling. We also assume that the relative priority of each job remains the same across all the stages on which it executes. The main problem we address in this paper is to determine a worst-case bound on the end-to-end delay of jobs for an arbitrary workflow given the computation times of other workflows that exist concurrently with it in the system.

IV. Intuition

In this section, we develop some intuition that will assist us in deriving the delay bound of workflows in an aggregation network. We explore a simple and seemingly-intuitive approach to estimate the worst-case delay using an example and show why the approach is not accurate. This exercise points us to the right intuition into developing the worst-case delay bound in the next section.

Consider the following system. Let j_{merge} denote a merge stage in the workflow graph of job J_k which has l incoming branches from stages j^{m_1}, j^{m_2}, ..., and j^{m_l}. By semantics of merge, J_k can execute on j_{merge} only when all its subjobs arrive from the l upstream stages. So, J_k can execute on stage j_{merge} after the last of the l subjobs arrives. One way to compute the worst-case end-to-end delay of J_k, is to consider all the possible paths from aggregation input to computing the aggregation result, and picking the path that incurs the largest end-to-end delay bound obtained through the delay composition algebra. This intuition works even when the workflow graph of J_k consists of multiple successive merges, by considering all the possible end-to-end paths that comprise the workflow graph.

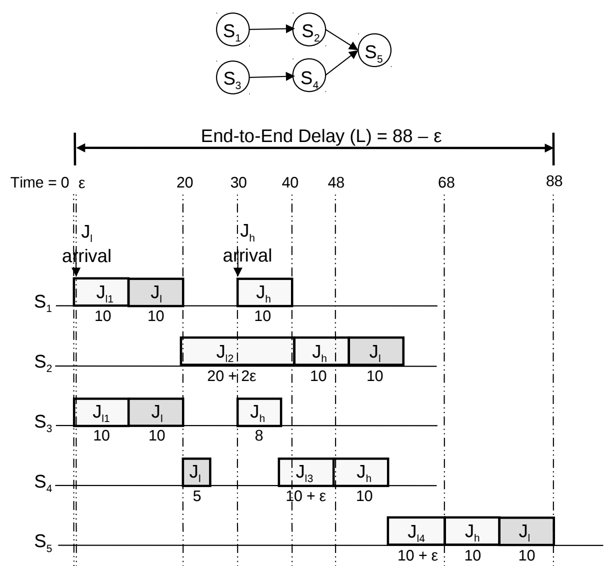

Figure 2. Workflow graph and execution trace for seemingly candidate aggregation composition approaches.

Such a system with two merging branches is illustrated in Figure 2. Let us assume that scheduling is non-preemptive in nature. Job J_l is the job whose end-to-end delay we wish to bound. Job J_h is a higher priority job, and jobs J_{l1}, J_{l2}, J_{l3} and J_{l4} are lower priority jobs. Let us further suppose that J_l arrives at stages S_1 and S_3 concurrently at time $\epsilon > 0$ (ϵ can be arbitrarily small). Job J_h arrives at stages S_1 and S_3 at time 30 units. Job J_{l1} executes only on stages S_1 and S_3 and arrives at time zero. Job J_{l2} executes only on stage S_2 and arrives at time $20 - \epsilon$. Job J_{l3} executes only on stage S_4 and arrives at time $38 - \epsilon$. Job J_{l4} arrives at stage S_5 at time $58 - \epsilon$. The computation times and execution trace of all the jobs are shown in Figure 2. Now, let us use the delay composition theory developed in [1], [16] along each path in the workflow graph of J_l, and compute the maximum to obtain an estimate of the worst-case end-to-end delay bound.

The worst-case delay bound along the pipeline segment $< S_1, S_2, S_5 >$ can be computed using delay composition theory (ignoring the parallel branch along $< S_3, S_4 >$), as the sum of three terms. The first term is the sum of maximum computation times for each higher priority job across all the stages on which it executes. The second term, is the stage additive component, which is one maximum stage execution time over all higher priority jobs for each stage on the path. The third term is the blocking term, which is the sum of the maximum execution time of any lower priority job at each stage.

$$\begin{aligned} Delay_{<S_1,S_2,S_5>}(J_l) &\leq 10 + (3 \times 10) + \\ &\quad (10 + 20 + 2\epsilon + 10 + \epsilon) \\ &= 80 + 3\epsilon \end{aligned} \tag{1}$$

Similarly, the worst-case delay along the pipeline segment $< S_3, S_4, S_5 >$ can be computed using delay composition theory as follows:

$$\begin{aligned} Delay_{<S_3,S_4,S_5>}(J_l) &= 10 + (3 \times 10) + \\ &\quad (10 + 10 + \epsilon + 10 + \epsilon) \\ &= 70 + 2\epsilon \end{aligned} \tag{2}$$

The maximum delay across both paths for J_l is therefore $80+3\epsilon$. However, as the execution trace suggests, the delay of J_l can be as large as 88 time units.

$$Delay_{trace}(J_l) = 88 - \epsilon$$

So, why didn't our intuition work in this case? Delay composition theory implicitly assumes that once a higher priority job preempts or overtakes a lower priority job, it will always execute ahead of the lower priority job on all future stages. However, when there are "merge" stages, this assumption breaks down. As illustrated in the example, while J_h delays J_l on one branch ($< S_1, S_2 >$), it executes after J_l in a different busy period on a parallel branch ($< S_3, S_4 >$) and takes longer to arrive at the merge stage. We call this a revisit event and a more formal definition is provided in Section V. This is why delay composition theory is not applicable to this case. We need to account for this revisit event.

A closer look at the above example reveals that the revisit event arises due to the fact that the higher priority job J_h completes execution sooner on one branch, while the lower priority job J_l completes sooner on another. This in turn requires that J_l arrives before J_h to the system. A closer scrutiny revealed that our previous delay composition result works fine as long as J_l arrives no earlier than J_h. So, intuitively, we should be able to calculate the delay starting from the arrival time of J_h using our previous result, and add the difference in the arrival times between J_l and J_h. This is precisely how we derive our delay bound in Section V. The end-to-end delay bound is expressed as the sum of two terms. The first is the maximum offset between the arrivals of J_l and any higher priority job. The second term is the maximum delay along any end-to-end path that constitutes the aggregation graph of J_l, which is computed using our previous delay composition result, similar to Equations 1 and 2. Thus, our delay bound directly depends on the offsets between the arrival times of jobs. The bounds will be tighter when the offsets are lower.

A natural question is, how do we determine the maximum offset of any higher priority job that can potentially interfere with the job under consideration? A simple, but pessimistic, solution is as follows. To begin with, assume that all higher priority jobs in the system can delay the job under consideration J_1. Use the analysis to determine the worst-case delay bound. If the arrival time of any higher priority job is greater than the worst-case delay bound, then that higher priority job cannot possibly delay J_1. Such higher priority jobs can be removed and the analysis can be repeated (resulting in a lower worst-case bound). At each iteration, we can discard some higher priority jobs, and the process is repeated until no higher priority jobs can be discarded from the interfering set of jobs. Clearly, this approach will only overestimate the set of interfering higher priority jobs. The analysis stands to gain if the worst-case arrival offset information for higher priority jobs is provided as system input together to the execution time characteristics of jobs.

V. An End-to-End Delay Bound for Aggregation Workflows

In this section, we derive a worst-case bound on the end-to-end delay of an aggregation workflow in terms of the stage computation times of other workflows executing concurrently with it. We derive the result for non-preemptive scheduling in Section V-A, and in the interest of brevity, only state the result for preemptive scheduling in Section V-B. The analysis extends previous work by the authors in developing a theory called delay composition theory, to analyze the worst-case end-to-end delay of jobs in various workflow topologies. The worst-case end-to-end delays of jobs in a pipelined distributed system under preemptive and non-preemptive scheduling was analyzed in [17], [18]. The result was extended to directed acyclic graphs in [16] and to graphs with cycles in [19]. An algebra was developed to reduce an arbitrary distributed system to an equivalent uniprocessor for the purposes of schedulability analysis in [1]. Any uniprocessor schedulability analysis can then be used to infer end-to-end delay and schedulability properties of jobs in the original distributed system. In a deviation from the above work, this paper considers system graphs that contain aggregation nodes, with the property that execution on these nodes require the execution on *all* incoming branches to complete before it can be undertaken. As motivated in the previous section, this important distinction makes such systems difficult to analyze.

A. Non-Preemptive Scheduling

For the benefit of the reader, we state the delay composition result for arbitrary directed acyclic graphs (DAGs) under non-preemptive scheduling from [16] here:

Non-preemptive Delay Composition Theorem [16]. *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_k of N stages can be composed from the execution parameters of other jobs that delay it as follows:*

$$Delay(J_k) \leq \sum_{J_i \in H} C_{i,max} + \sum_{j \leq N} (\max_{J_i \in H} C_{i,j} + \max_{J_i \in L_j} C_{i,j}) \quad (3)$$

where H is the set of higher priority jobs and L_j is the set of lower priority jobs that execute on stage j.

As priority is assigned for each job, let us order all the jobs in decreasing priority order. Let us suppose that we wish to bound the worst-case end-to-end delay of a job J_k. Let S denote the set of all jobs executing concurrently with J_k. Let Off_i denote the offset from time zero at which a job J_i arrives at all the branches of its workflow (if different branches arrive at different offsets, Off_i can be set to the maximum of these offsets). Let $Delay_i$ denote the worst-case end-to-end delay of job J_i (we wish to determine $Delay_k$). Further, let $Delay_{i,j}$ denote the worst-case delay of a job J_i up to completing its execution on stage j. Let $Paths_i$ denote the set of all paths from any source of a branch of job J_i to its sink (the union of all paths in $Paths_i$ is the workflow tree of J_i). We now state the main result of this paper:

Non-Preemptive Delay Composition Theorem for Aggregation Workflows: *Under a non-preemptive scheduling policy that assigns the same priority across all stages for each job, the worst-case end-to-end delay of a job of workflow F_k in an aggregation tree is bounded as,*

$$\begin{aligned} Delay_k \leq \quad & \max_{i \leq k}(Off_i) - Off_k + \sum_{i \leq k} C_{i,max} \\ & + \max_{p \in Paths_k} \sum_{j \in p} (\max_{i \in S} C_{i,j} + \max_{i \leq k} C_{i,j}) \quad (4) \end{aligned}$$

where Off_i is the offset of job J_i from time zero.

Proof: At each merge-stage in its workflow tree, as a job needs to wait for all incoming branches to complete execution before it can execute on the merge-stage, the following sequence of events is possible (which is otherwise not possible in the absence of merge-stages). A higher priority job J_i delays a lower priority job $J_{i'}$ and arrives ahead of it on one branch of merge-stage j_1. However, along another branch, $J_{i'}$ completes execution and arrives ahead of J_i to stage j_1. Due to this reversal in the arrival order, it is possible for J_i to again delay $J_{i'}$ at a downstream stage j_2 (can be the same as stage j_1). Let us call the instance where J_i again executes ahead of $J_{i'}$ after the reversal, as a *revisit* event (if J_i always executes after $J_{i'}$ on all remaining stages, the revisit event occurs at the last execution stage of J_i). This is illustrated in Figure 3 (J_{l1}, J_{l2}, and J_{l3} are of lower priority than job $J_{i'}$).

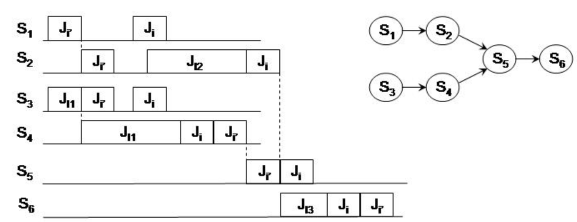

Figure 3. Example illustrating a reversal event

We shall now prove the theorem using induction on the number of such revisit events between jobs J_i and $J_{i'}$, such that $i < i' \leq k$. The basis step is when there are no such events. This means that at every merge-stage each higher priority job that was ahead of job J_k remains ahead of job J_k throughout the system. Hence, $Delay_k$ can be bounded by the maximum delay across any end-to-end pipeline path from a source to the sink. Applying the delay composition result (Equation 3) to each path $p \in Paths_k$ and picking the maximum, we can obtain a bound on $Delay_k$ as follows:

$$Delay_k \leq \sum_{i \leq k} C_{i,max} + \max_{p \in Paths_k} \sum_{j \in p} (\max_{i \in S} C_{i,j} + \max_{i \leq k} C_{i,j})$$

This proves the theorem for the basis step when there are no revisit events.

Let us assume that the result is true up to $n-1$ revisit events, $n \geq 1$. We shall now prove that the result is true when there are n such events. Among all the n revisit events, consider the one that is latest in time. Let J_i be the higher priority job involved in this last revisit event. If J_i exits the system at some stage prior to the last stage on which J_k executes, lets add executions of zero computation time for J_i on the remaining stages. This operation cannot decrease the delay of J_k. Let stage j be the last stage where there is no idle time between the executions of J_i and J_k. Let $E(i,j)$ denote the instant of time at which job J_i completes execution on stage j. We can now calculate the end-to-end delay of J_k as the sum of three terms: the offset of J_i relative to J_k, the delay of J_i up to its completion on stage j, and the delay of J_k from this instant on stage j to its sink. This is illustrated in Figure 4.

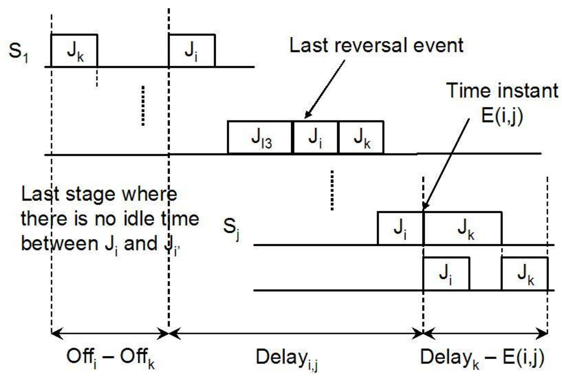

Figure 4. Example illustrating the proof

$$Delay_k \leq (Off_i - Off_k) + Delay_{i,j} + (Delay_k - E(i,j)) \tag{5}$$

The delay of job J_i up to stage j, $Delay_{i,j}$, can be obtained from induction assumption as it has only $n-1$ revisit events. Let H_1 be the set of higher priority jobs including J_i that contribute to $Delay_{i,j}$.

$$\begin{aligned} Delay_{i,j} \leq \max_{i' \leq i}(Off_{i'} - Off_i) + \sum_{i' \in H_1} C_{i',max} \\ + \max_{p \in Paths_k} \sum_{j' \leq j \in p} (\max_{i' \in S} C_{i',j'} + \max_{i' \leq i} C_{i',j'}) \end{aligned} \tag{6}$$

In the above equation, $j' \leq j$ denotes stages before stage j and $j' > j$ denotes stages after stage j in path p. The delay of job J_k starting from instant $E(i,j)$ on stage j does not encounter any revisit events and the delay can be obtained from the basis step. Let H_2 be the set of higher priority jobs that contribute to the delay of J_k from stage j to the sink.

$$\begin{aligned} Delay_k - E(i,j) \leq \sum_{i' \in H_2} C_{i',max} + \\ \max_{p \in Paths_k} \sum_{j' > j \in p} (\max_{i' \in S} C_{i',j'} + \max_{i' \leq k} C_{i',j'}) \end{aligned} \tag{7}$$

Expanding Equation 5 using Equations 6 and 7 we get,

$$\begin{aligned} Delay_k \leq \max_{i' \leq k}(Off'_i) - Off_k + \sum_{i' \in H_1} C_{i',max} + \sum_{i' \in H_2} C_{i',max} \\ + \max_{p \in Paths_k} \sum_{j' \in p} (\max_{i' \in S} C_{i',j'} + \max_{i' \leq k} C_{i',j'}) \end{aligned} \tag{8}$$

Clearly, $H_1 \cup H_2$ is a subset of the set of higher priority jobs of J_k. What remains to be shown is that $H_1 \cap H_2 = \phi$. Any higher priority job J_h in H_1 has executed ahead of J_i on some stage $j' \leq j$. If $J_h \in H_2$, it must have arrived at stage j after J_i (there is an idle time between the executions of J_i and J_k after stage j). This would be a revisit event that is after that of J_i's, as J_h has executed ahead of J_i on a stage j' prior to j, leading to a contradiction. This proves the theorem. □

B. Preemptive Scheduling

In the interest of brevity, we only state the result under preemptive scheduling. The proof is similar to the non-preemptive case, using the preemptive version of the delay composition theorem.

Preemptive Delay Composition Theorem for Aggregation Workflows: *Assuming a preemptive scheduling policy that assigns the same priority across all stages for each job, the worst-case end-to-end delay of a job of workflow F_k in an aggregation tree is bounded as,*

$$\begin{aligned} Delay_k \quad \leq \quad \max_{i \leq k}(Off_i) - Off_k + \sum_{i \leq k} C_{i,max} \\ + \max_{p \in Paths_k} \sum_{j \in p} \max_{i \leq k} C_{i,j} \end{aligned} \tag{9}$$

where Off_i is the offset of job J_i from time zero.

VI. Schedulability Analysis by Reducing the Aggregation System to an Equivalent Uniprocessor

In this section, we present a reduction of the aggregation workflow graph into an equivalent uniprocessor for the purposes of schedulability analysis. This reduction maintains the property that the worst-case delay of a job in the distributed aggregation graph is no more than the delay of the corresponding task in the equivalent uniprocessor. Thus, if the uniprocessor task completes execution within its deadline, then the corresponding distributed workflow will also complete execution within its end-to-end deadline. The hypothetical uniprocessor can be analyzed using any well-known schedulability analysis technique (e.g., Liu and Layland bound [20], response time analysis [21]).

Under non-preemptive scheduling, the schedulability of a job J_k of an aggregation workflow can be determined by analyzing the schedulability of a hypothetical uniprocessor constructed as follows:

- Each higher priority job J_i in the original distributed system is replaced with a job J_i^* on the hypothetical singe stage, with a computation time equal to $C_{i,max}$, its maximum computation time across all stages in the distributed system and a deadline D_i, same as that of J_i.
- Job J_k is replaced with job J_k^* of lowest priority in the hypothetical single stage and computation time of $\max_{i \leq k}(Off_i) - Off_k + C_{k,max} +$

$\max_{p \in Paths_k} \sum_{j \in p} (\max_{i \in S} C_{i,j} + \max_{i \leq k} C_{i,j})$ and deadline equal to D_k.

Under preemptive scheduling, the hypothetical uniprocessor is constructed as follows:

- Each higher priority job J_i in the original distributed system is replaced with a job J_i^* on the hypothetical sing stage, with a computation time equal to $2C_{i,max}$ and deadline equal to D_i.
- Job J_k is replaced with job J_k^* of lowest priority in the hypothetical single stage and computation time of $\max_{i \leq k}(Off_i) - Off_k + C_{k,max} + \max_{p \in Paths_k} \sum_{j \in p} \max_{i \leq k} C_{i,j}$ and deadline equal to D_k.

The reduction process in the delay composition framework is schematically depicted on an aggregation tree in Figure 5.

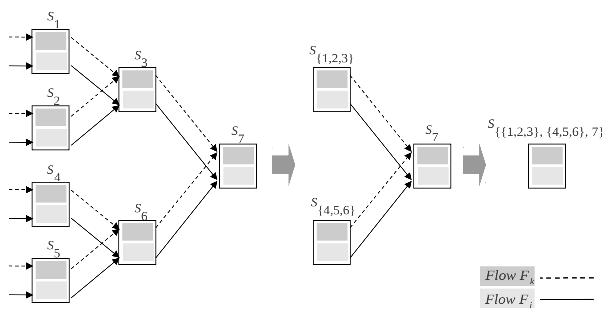

Figure 5. The reduction process on an aggregation tree in the delay composition framework

VII. Evaluation

In this section, we evaluate our approach to demonstrate the accuracy of our analysis in quantifying the end-to-end delays of workflows in an aggregation network. Specifically, we elaborate on how different system and load parameters can affect the effectiveness and accuracy of our approach. Towards this goal, two questions are explored: first, how efficiently our analysis utilizes the system resources, and second, how accurate it is in estimating the worst-case end-to-end delays.

We consider aggregation trees described as full binary trees for our simulations. So, a tree of height H has $2^{H+1} - 1$ nodes and H levels of cascading merges. All workflows are assumed to execute on all nodes of the aggregation tree. Each workflow comprises of a sequence of aperiodic jobs. Each job is assigned an end-to-end deadline equal to $500 \times H \times 10^{\alpha}$ simulation time units, where H denotes the height of the aggregation tree and α is chosen uniformly across the interval $[0, DR]$. The default height of the aggregation tree is 5 and the number of stages is 63. The parameter DR, called deadline ratio parameter, is set to 2.0 in all experiments, unless otherwise specified. This choice allows the end-to-end deadlines of jobs to vary by a factor of 10^2. The computation times of jobs are chosen proportional to their deadlines based on a uniform distribution with a mean of $\frac{D}{H \times JR}$ in which D denotes the corresponding end-to-end deadline and JR denotes the job resolution parameter. Unless otherwise specified, the JR factor is set to 20. Different computation times are within 12% of the mean value. Jobs are activated with offsets randomly chosen from the interval $[0, OR \times 500 \times H]$, wherein OR denotes the offset resolution factor, set to 0.5 throughout the experiments (note that $500 \times H$ is the minimum end-to-end deadline of any job). Job priorities are assigned based on their end-to-end deadlines, and scheduling follows an earliest deadline first policy.

The delay composition algebra and holistic analysis [2] are simulated and compared in terms of different system and workload parameters such as the number of stages in the system, job resolution, deadline ratio, offset resolution, and the number of real-time jobs. Every experiment is repeated for 50 times and the 95^{th} percentile confidence level is within 1% of the plotted average value. For the sake of clarity, we do not plot the confidence values.

A. Resource Utilization

The delay composition algebra can be employed as an admission controller, which admits only the part of the workload that it deems to be feasible (all admitted workflows are guaranteed to meet their end-to-end deadline). Guaranteeing the schedulability of all admitted workflows is a vital requirement for mission-critical real-time applications. This avoids system resources from being channeled into serving jobs that may not meet their end-to-end deadline, ensuring more efficient utilization of the system resources.

For each experiment we employ the delay composition framewrok and holistic approach as admission controllers in independent executions that work on the same input. In each experiment, the same job set (which is sufficient to overload both approaches) is fed to both admission controllers and then the admitted jobs for each approach are executed through the aggregation tree. The overload region is of interest since it determines the capacity of the system under each admission controller.

In the first experiment, we varied the height of the aggregation tree (and hence the number of stages). Figure 6 presents the comparison of average per stage utilization with increasing values of the height of the tree from 1 to 5 (the number of stages varies from 3 to 63) in this experiment. The height of the tree is also the path length for the jobs. The delay composition algebra and holistic approach are denoted as DCA and Holistic, respectively. As depicted in the figure, the resource utilization of delay composition algebra remains nearly constant. This shows that the pessimism in analysis is independent of the number of stages in the system. On the other hand, the utilization of holistic approach drops significantly as system size increases. The reason is that the holistic approach analyzes one stage at a time and the pessimism grows as path length increases. Hence, our analysis is especially useful for large systems.

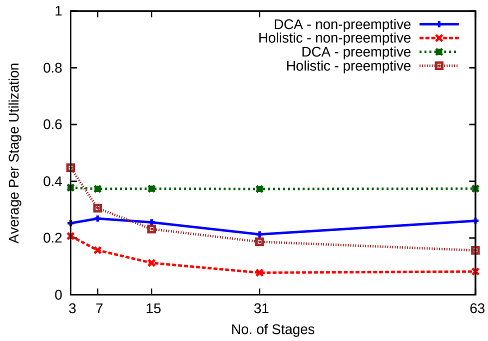

Figure 6. Comparison of utilization for different number of stages

In the next experiment, we varied the size of the real-time jobs in the system. The job resolution parameter is decreased from 80 to 5 (increasing the job sizes by a factor of 16) and the results for average per stage utilization are shown in Figure 7. A job resolution of 80 denotes a large number of small jobs, while a value of 5 denotes a small number of large jobs. Under preemptive scheduling, the system utilization of both approaches remain nearly constant. Following a non-preemptive scheduling policy, the average per stage utilization of both DCA and holistic analysis techniques strictly decreases as jobs become larger. The reason is that the blocking delay component becomes significantly large as job sizes increase. The delay composition algebra consistently outperforms holistic analysis under non-preemptive as well as preemptive scheduling.

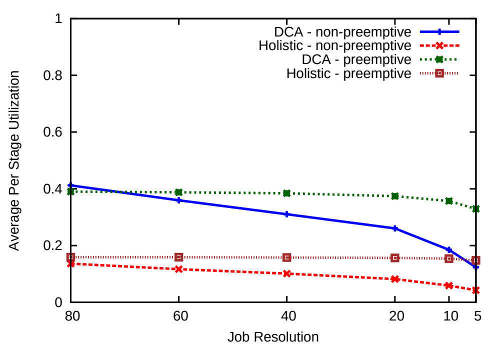

Figure 7. Comparison of utilization for different values of job resolution

Next, we changed the order of variability in job deadlines, increasing the deadline ratio parameter, DR, from 0.5 to 3.0 (recall that the deadlines vary by a factor of 10^{DR}). Figure 8 shows the average per stage utilization for this experiment. As job sizes are proportional to end-to-end deadlines, when the variability in deadlines increases, an increased number of larger jobs are introduced into the job set. When scheduling follows a non-preemptive policy, larger low priority jobs impose larger blocking delays on higher priority jobs. Therefore, as observe in the figure, under both DCA and holistic analysis, for larger deadline ratios (beyond 1) the effect of blocking delay becomes significant and the average utilization drops thereof. However, for the entire range of deadline ratios, the delay composition algebra outperforms holistic analysis.

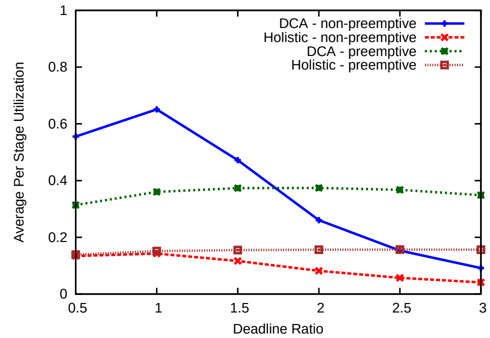

Figure 8. Comparison of utilization for different values of deadline ratio

Figure 9 plots the effect of increasing the offset resolution parameter. We increased the offset resolution value form 0.1 to 100 which denotes increasing the maximum offset in job arrivals from one-tenth of the minimum job deadline to 100 times of that. Note that the x axis is drawn in logarithmic scale. While the average per stage utilization under holistic analysis is independent of arrival offsets, that of the DCA is a function of offset values and drops beyond the point where offset values become larger than the minimum job deadline. The results report that when offset resolution remains below 100 times of the minimum job deadline, the delay composition algebra remains superior to the holistic approach.

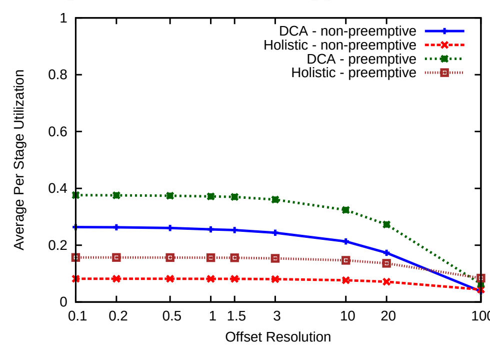

Figure 9. Comparison of utilization for different values of offset resolution

B. *End-to-End Delay Bound Accuracy*

In this section, we evaluate the tightness of the delay composition algebra by comparing the actual delay of jobs obtained through simulation to the analytically calculated worst-case delay bounds. As we are only interested in delay and not schedulability, we do not perform admission control. We compare our analysis framework with holistic analysis, and in each experiment retain the given workload constant in order to reflect merely the impact of parameters under study.

Figure 10 compares the average ratio of end-to-end delays obtained from simulation to the analytically computed delay bounds with respect to increase in the system size. As the results show, under preemptive scheduling, the holistic approach calculates tighter bounds than DCA for small-scale systems. However, as system scales, the delay bounds of DCA become significantly more accurate than those of holistic (achieving 6% and 24% improvement in accuracy respectively at 7 and 63 stages). Under non-preemptive scheduling, the delay composition algebra again outperforms the holistic approach with $14\% - 34\%$ difference in the level of tightness.

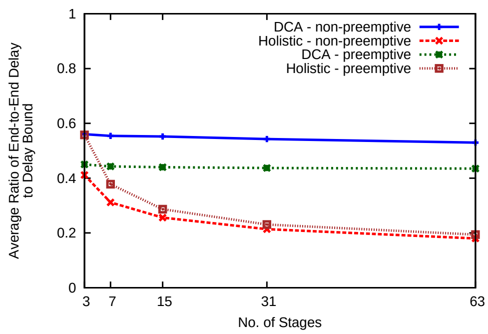

Figure 10. Comparison of delay bounds of the delay composition algebra and holistic approach for different number of stages

We then conducted an experiment to evaluate the effect of input workload on delay bounds. Figure 11 shows the corresponding results for job sets of size 10, 20, 40, 80, 160, and 320. As depicted in the figure, the DCA delay bound accuracy is superior to that of the holistic analysis for all workload values. The reason is that the holistic approach performs on a per stage basis and reflects the contribution of each higher priority job to the end-to-end delay at all stages along its path, while the delay composition algebra takes a per job approach. The latter leads to more accurate estimation of the worst-case delay for DCA. Furthermore, under non-preemptive scheduling policy, while the tightness of holistic delay bound decreases, that of the DCA improves. This is because for DCA, the difference between the delay bound and the actual delay remains more or less constant with increase in the number of jobs. Therefore, the delay ratio improves for DCA. On the other hand, for holistic analysis, the difference between the delay bound and the actual delay grows significantly with load and hence the delay ratio drops.

Under preemptive scheduling, the delay ratio of both analysis techniques decreases with increase in workload. For DCA, the reason lies in the fact that it accounts for a delay of two stage execution times for the contribution of each higher priority job in delay of other lower priority ones in the worst case (rather than one in non-preemptive scheduling). We encourage the reader to review our prior work [1], [16] for an explanation of this source of pessimism in the analysis. Under both non-preemptive and preemptive scheduling, DCA consistently has a higher ratio of end-to-end delay to the computed delay bound as compared to holistic analysis (more than 20% when the number of jobs is over 80).

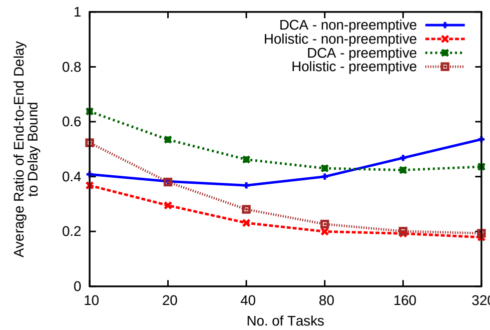

Figure 11. Comparison of delay bounds of the delay composition algebra and holistic approach for different number of jobs

Figure 12 plots the results for the effect of job resolution on delay bounds. As depicted in the figures, when changing jobs sizes, the delay composition algebra is superior to the holistic approach under both preemptive and non-preemptive scheduling policies (up to 25% and 45% improvement in tightness, under preemptive and non-preemptive policies, respectively).

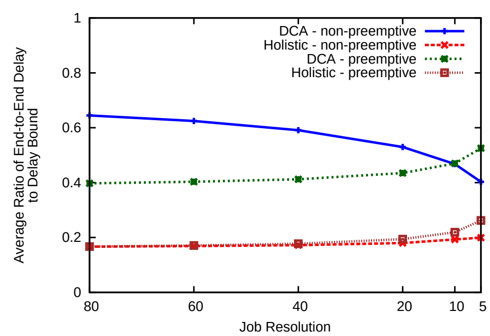

Figure 12. Comparison of delay bounds of the delay composition algebra and holistic approach for different values of job resolution

The impact of deadline ratio on the tightness is reported in Figure 13. Recall that a value of x for deadline ratio implies a 10^x variability in jobs deadlines. As indicated before, the decrease in delay ratio under a non-preemptive policy is related to the blocking delay component becoming more significant. Both analysis approaches predict a larger worst-case blocking factor leading to a reduced value for the delay ratio for both analysis techniques. Here again, the delay composition algebra outperforms the holistic approach by about 25% and 45% under preemptive and non-preemptive scheduling schemes, respectively.

The impact of offset in job arrivals is presented in Figure 14. Note that the x axis is in logarithmic scale. The worst-case delay bound estimation of the DCA is a function of offset values, while that of the holistic remains

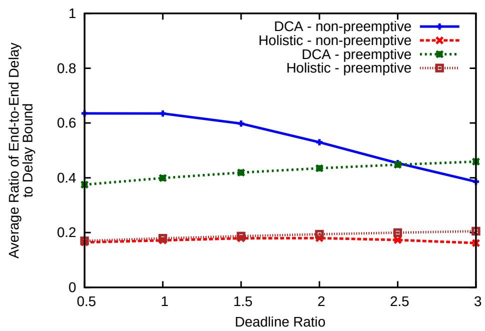

Figure 13. Comparison of delay bounds of the delay composition algebra and holistic approach for different values of deadline ratio

constant. The slight reduction in delay ratios of the holistic is due to the fact that increase in offset values reduces the interference in jobs execution which leads to decrease in the value of actual end-to-end delay. We observe that up to an offset parameter value of nearly 100, DCA outperforms holistic analysis in terms of the achieved average ratio of the end-to-end delay of jobs to the computed delay bound.

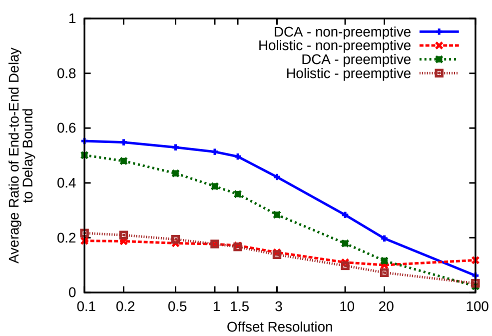

Figure 14. Comparison of delay bounds of the delay composition algebra and holistic approach for different values of offset resolution

VIII. Conclusions

In this paper, we investigate timing properties and delay composability of jobs in multisensor data aggregation systems. We propose a theoretical framework which extends previously proposed delay composition algebra for a new class of systems characterized by a "MERGE" primitive. We provide intuition as to why analyzing such systems can be difficult, and develop a framework to determine offline schedulability of multi-criticality distributed workload in data aggregation systems. The framework assists to prevent system resources from being channeled into serving jobs which may not complete its end-to-end execution within prespecified deadlines. We provide delay composition rules for aggregation workflows under preemptive as well as non-preemptive scheduling policies based on characteristics of concurrent workflows and their corresponding arrival offsets. We extensively evaluate our framework through simulations and show that our theoretical framework is significantly more accurate than traditional analysis techniques and effectively utilizes distributed resources. Our framework is especially beneficial for large systems.

Acknowledgements

This research was sponsored in part by ONR grant N00014-10-1-0172 and NSF grant CNS 07-20513 and in part by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] P. Jayachandran and T. Abdelzaher, "Delay composition algebra: A reduction-based schedulability algebra for distributed real-time systems," in *RTSS*, December 2008, pp. 259–269.

[2] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Elsevier Microprocessing and Microprogramming*, vol. 40, no. 2-3, pp. 117–134, 1994.

[3] 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*, vol. 42, no. 14, pp. 2899–2913, July 2004.

[4] R. Cruz, "A calculus for network delay, part i: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, January 1991.

[5] ——, "A calculus for network delay, part ii: Network analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132–141, January 1991.

[6] X. Li and J. Cao., "Coordinated workload scheduling in hierarchical sensor networks for data fusion applications," *Journal of Computer Science and Technology*, vol. 23, May 2008.

[7] J. Xu and D. Parnas, "On satisfying timing constraints in hard real-time systems," *IEEE Transactions on Software Engineering*, vol. 19, no. 1, pp. 70–84, January 1993.

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

[9] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1268–1274, 1997.

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

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

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

[13] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *IEEE International Symposium on Circuits and Systems*, vol. 4, May 2000, pp. 101–104.

[14] B. Jonsson, S. Perathoner, L. Thiele, and W. Yi, "Cyclic dependencies in modular performance analysis," in *ACM EMSOFT*, Oct. 2008, pp. 179–188.

[15] E. Wandeler, A. Maxiaguine, and L. Thiele, "Quantitative characterization of event streams in analysis of hard real-time applications," in *IEEE RTAS*, May 2004, pp. 450–459.

[16] P. Jayachandran and T. Abdelzaher, "Transforming acyclic distributed systems into equivalent uniprocessors under preemptive and non-preemptive scheduling," in *ECRTS*, July 2008, pp. 233–242.

[17] ——, "A delay composition theorem for real-time pipelines," in *ECRTS*, July 2007, pp. 29–38.
[18] 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*, vol. 40, no. 3, pp. 290–320, December 2008.
[19] ——, "End-to-end delay analysis of distributed systems with cycles in the task graph (to appear)," in *ECRTS*, July 2009.
[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[21] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering*, vol. 8, no. 5, pp. 284–292, 1993.