

# Real-time Capacity of Networked Data Fusion

Forrest Iandola, Fatemeh Saremi, Tarek Abdelzaher

Department of Computer Science  
University of Illinois  
Urbana, IL 61801

Email: {iandola1, saremi1, zaher}@illinois.edu

Praveen Jayachandran

IBM Research  
India

Email: prjayach@in.ibm.com

Aylin Yener

Department of Electrical Engineering  
Pennsylvania State University  
University Park, PA 16802

Email: yener@ee.psu.edu

**Abstract**—In networked data fusion systems, results must be produced by end-to-end deadlines. As such, latency is an important attribute contributing to quality-of-information (QoI) in these systems. A key question is how much work can be completed on time given some amount of distributed resources and the logical workflow topology of the data fusion graph. To answer this question, this paper presents the concept of real-time capacity; a performance metric in line with the recently introduced *operational information content capacity* (OICC) of a networked data fusion system. It extends results developed by the authors recently in the real-time systems community to analyze the capacity of various data fusion workflow topologies. A simple closed-form expression for a topology-dependent real-time capacity bound is derived and is shown to be accurate via extensive simulations.

**Keywords:** Latency, data fusion, pipelines, capacity.

## I. INTRODUCTION

This paper presents *worst-case* latency analysis of real-time data fusion systems. We characterize the data fusion system by a collection of different distributed fusion tasks, we call *workflows*. This terminology is meant to emphasize that each task constitutes a flow of distributed processing. The system is *real-time* because individual tasks have corresponding maximum tolerable end-to-end latencies, we call *end-to-end deadlines*. A deadline describes the maximum delay that can be tolerated between receiving a set of fusion inputs and computing the corresponding fusion results. For example, in an intrusion detection scenario, where sensors and fusion stages form “virtual tripwires” that detect and classify intruders, one may want to design the system such that some finite reaction time is guaranteed. In general, a key question in a real-time data fusion system is: under what workload conditions can all deadlines of all tasks in the system be met? This paper derives the answer as a closed-form expression, we call the *real-time capacity region*. The real-time capacity region quantifies the relation between load, distributed resource topology of the data fusion system, and the worst-case end-to-end latency constraints that can be met.

The paper utilizes recent results in real-time computing literature that present worst-case latency analysis for pipelines. This previous work considered how worst-case latency of tasks on individual stages of distributed multi-priority systems compose, when these stages are traversed in a pipelined fashion. It was shown that delays compose *subadditively*. In other words, if the worst-case latency of a task, executing

at some priority  $p$ , on machine  $A$  is  $L_A$  and the worst-case latency of this task, executing at the same priority, on machine  $B$  is  $L_B$ , then the worst-case latency of the task if it executes at priority  $p$  on machine  $A$  then machine  $B$  is deterministically less than  $L_A + L_B$ . The result follows as a consequence of the pipelining effect. An exact subadditive expression for delay composition in pipelines, called the *pipeline delay composition theorem* [1], was published by two of the authors. This result is extended here to derive conditions under which end-to-end deadlines of different workflows are met in data fusion systems.<sup>1</sup>

The rest of this paper is organized as follows. Section II describes the data fusion model and the real-time capacity problem statement. Section III derives the basic capacity expression for data fusion pipelines, then extends it to more general fusion topologies. We illustrate the real-time capacity calculations with an example in Section IV. A simulation-based evaluation is presented in Section V. Section VI reviews related work. The paper concludes with Section VII.

## II. MODEL AND PROBLEM STATEMENT

Consider a data fusion system, composed of one or more data workflows, such as video feeds from UAVs or telemetry from deployed sensors, each moving through several stages of processing (e.g., tracking, threat analysis, and display). A processing stage is modeled as a single processing resource, scheduled in priority order. For example, it can refer to a processor that schedules arriving tasks, or a network link that queues up incoming packets. Since each stage represents one resource, in the rest of this paper, we use the names *stage* and *resource* interchangeably. Collectively, the stages of processing of each workflow are organized in a graph called the *workflow graph*. It determines precedence constraints among the different stages. In the simplest data fusion system, the workflow graph is simply a chain representing a single processing pipeline. More complex graphs may represent trees where different such pipelines ultimately merge.

To apply real-time analysis, we further discretize each workflow,  $F_i$ , into successive chunks, called *jobs*. Each job is a distributed computation that follows the workflow graph,  $F_i$ , but processes only a portion of the input data; namely,

<sup>1</sup>Our deterministic worst-case analysis complements queuing-theoretic characterization of average latency and latency distribution, that offers tools for latency analysis in a probabilistic sense.

the set of samples, images, or video frames that arrived, as inputs to  $F_i$ , within some finite interval of time,  $P_i$ . This is appropriate, for example, when stages run iterative algorithms that update outputs upon receipt of the next batch of inputs. The processing of job  $q$  of workflow,  $F_i$ , begins at the branches and ends at the root of the data fusion tree. The processing time of job  $q$  on stage,  $j$ , of the data fusion system is denoted by  $C_{i,j}[q]$ , or simply  $C_{i,j}$  if all jobs of the same workflow on a given resource have the same processing time.<sup>2</sup> We further call  $R_i = 1/P_i$  the *job rate* of workflow  $F_i$ .

Figure 1 shows an example of two workflows,  $F_1$  and  $F_2$ , that traverse two stages each. The processing time (of a job) of each workflow on each stage is indicated. For example, the processing time of  $F_1$  on the second stage, denoted  $C_{1,2}$ , is 1.1. As we show later, it is also useful to define  $C_{i,max}$  to refer to the maximum processing time of a job of workflow  $F_i$  on any of its stages (i.e.,  $\max_j C_{i,j}$ ), as well as  $C_{max,j}$  to refer to the longest processing time of any of the workflows on stage  $j$  (i.e.,  $\max_i C_{i,j}$ ). These quantities are also illustrated in Figure 1.

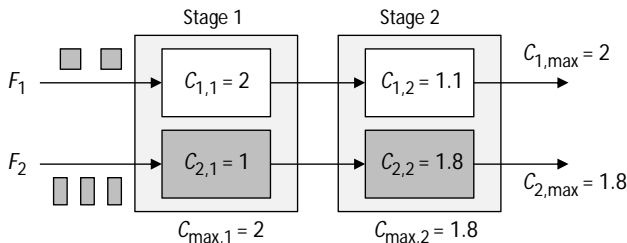


Figure 1. Example of two workflows.

In a *real-time* data fusion pipeline, 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, 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 is assigned a priority and scheduling is non-preemptive. We further assume that priority assignment is consistent across all resources. Hence, the priority of a job, once determined by the scheduler, remains the same at all stages. The problem statement is to find a general condition on incoming load under which each job traverses the system within its end-to-end deadline. This condition implicitly defines the *real-time capacity region* of the data fusion system, in that it defines how much load it can handle on time given the latency constraints on various workflows.

### III. REAL-TIME CAPACITY OF DATA FUSION PIPELINES

In this section, we first derive a general expression for the capacity within which the end-to-end deadlines of all jobs of

<sup>2</sup>Note that, in this paper, we borrow terminology from real-time computing where  $C$  denotes computation times; or an amount of processing that a task needs on a resource. This is not to be confused with terminology from information theory, where  $C$  commonly refers to capacity. The reader should interpret  $C$  as the former, not the latter.

all workflows are always met. We then apply the result to different workflow topologies.

The expression uses recent results developed in the real-time scheduling community regarding schedulability analysis of distributed systems. A delay composition theorem that computes a worst-case bound on end-to-end latency of jobs in pipelined systems under preemptive and non-preemptive scheduling was recently derived by the authors [1], [2]. This result was extended to more complex workflow graphs [3], [4], and ultimately to an algebra, called *delay composition algebra* [5] that describes how worst-case delays of subsystems (in a distributed system) compose into worst-case delays across the entire system. It allows reducing distributed systems to centralized systems that are (roughly) equivalent in terms of worst-case latency. The reduction allows a variety of existing tools to be applied to the reduced system in order to analyze the original distributed one.

The main result for pipelines has shown that each workflow  $F_i$  on the pipeline can be represented by an equivalent task on a uniprocessor that has a processing time equal to  $C_{i,max}$ , the largest of  $F_i$ 's processing times on any of the workflow stages. Each task in the resulting uniprocessor set has a worst-case latency that is approximately the same as that of the corresponding distributed workflow. Hence, the distributed system can be approximated by the uniprocessor for purposes of latency analysis. The approximation was shown to be of the order of the execution time of one job. It improves as the number of concurrent workflows increases. For instance, workflows  $F_1$  and  $F_2$  that traverse the pipeline in Figure 1 are approximated by the uniprocessor task set shown in Figure 2 below, where each workflow is replaced by one processing time,  $C_{i,max}$ .

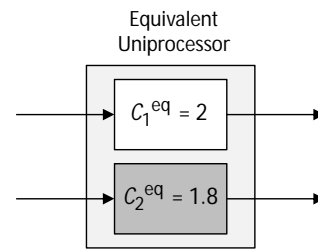


Figure 2. The (approximately) equivalent uniprocessor.

To see that the systems in Figure 1 and Figure 2 are roughly equivalent, consider a set of 10 workflows of type  $F_1$  and 10 workflows of type  $F_2$ . Figure 3 compares the worst-cast delay of the two systems when a job of each workflow arrives to the system at the same time. We further assume that  $F_1$  is higher priority than  $F_2$  (note that, the reduction result holds for any scheduling policy).

As can be seen from Figure 3, the worst-case latencies of a job of  $F_1$  and  $F_2$  are roughly the same in the original system (Figure 3-a) and in the reduced system (Figure 3-b). The difference, as shown in prior work [5], is of the order of the execution time of one job.

In the section below, we consider a distributed data fusion

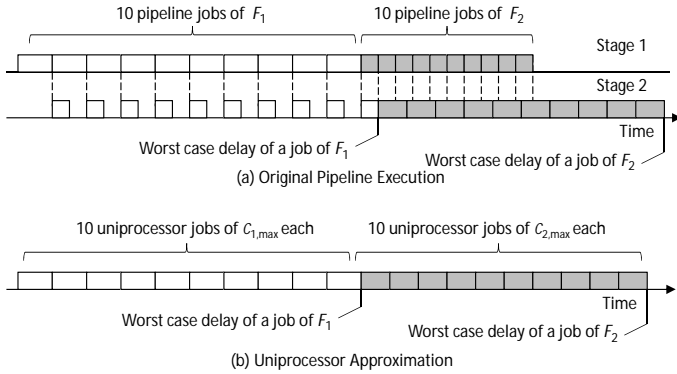


Figure 3. A comparison of original and reduced systems.

system that has been reduced to a uniprocessor, such that each workflow  $F_i$  was reduced to a uniprocessor task  $T_i$ . The reduction satisfies the condition that the end-to-end latency of a distributed job  $q$  in the original data fusion system is well approximated by the worst-case latency of the equivalent uniprocessor job  $q^{eq}$ . Real-time capacity is derived accordingly. Later, we describe in more detail the reduction from distributed workflows to uniprocessors, hence specializing the capacity expression to particular topologies of data fusion systems.

#### A. General Capacity Derivation

Consider the real-time capacity region for a data fusion pipeline. This region implicitly characterizes the conditions on input load for which all latency constraints of all jobs are *always* met. By “always,” we mean even in the worst case. Note that ability to meet deadlines depends on the scheduling policy. In this paper, we are interested in an *operational* notion of capacity. Hence, rather than providing the best possible capacity region achievable over all (known or unknown) scheduling policies, we are interested in quantifying such a region for a *given* practical scheduling policy. Of particular interest are scheduling policies with local optimality properties such as Earliest-deadline first (EDF).

EDF is the optimal dynamic-priority scheduling policy for independent tasks on uniprocessors. EDF is known to meet all task deadlines on a uniprocessor as long as the overall instantaneous utilization of all current tasks on the processor is less than 1. A task is said to be current if it arrived but its deadline has not expired. Instantaneous utilization,  $U$ , for a set of current tasks,  $T_1, \dots, T_n$  (where  $T_i$  has a computation time  $C_i$  and relative deadline,  $D_i$ ) is given by:

$$U = \sum_{i=1}^n \frac{C_i}{D_i} \quad (1)$$

Consider the end-to-end latency of job  $q$  in the distributed data fusion system. It is desired to compute the general condition under which job  $q$  meets its end-to-end deadline  $D_q$ . The system is reduced to an equivalent uniprocessor, where the equivalent job  $q^{eq}$ , shares the processor with the set of

workflows  $F_i$ , each of rate  $R_i$ , a relative deadline  $D_i$ , and an equivalent computation time  $C_i^{eq}$ . Let us refer to the above task set collectively as set  $S_{uni}$ . We call the reduction *valid* if it satisfies the condition that the worst-case delay of equivalent task  $q^{eq}$  on the uniprocessor is no less than the worst-case delay of  $q$  in the original distributed data fusion system. This ensures that when the reduced system meets deadlines, so does the original one. Later, we show how to compute a valid reduction.

Given a valid reduction, to understand whether task  $q^{eq}$  will meet its deadline on the equivalent uniprocessor, we substitute with task parameters into Equation (1). Note that, in the interval  $D_i$ , there may be at most  $\lceil D_i R_i \rceil$  current invocations of the task (by the definition of “current” stated above), assuming they meet their deadline (i.e., never stay longer than  $D_i$  in the system). Thus, the instantaneous uniprocessor utilization,  $U$  of the above task set is given by:

$$U = \frac{C_q^{eq}}{D_q} + \sum_i \frac{C_i^{eq}}{D_i} \lceil D_i R_i \rceil \quad (2)$$

The capacity region therefore becomes:

$$\frac{C_q^{eq}}{D_q} + \sum_i \frac{C_i^{eq}}{D_i} \lceil D_i R_i \rceil \leq 1 \quad (3)$$

Note that, if the reduction to the equivalent uniprocessor was a valid reduction, this condition also describes a region where the original distributed system meets deadlines.

#### B. A Fluid Approximation

While Equation (3) is a general expression, it is interesting to consider the important special case where individual job processing times are small, and where jobs arrive at a high rate compared to their end-to-end deadline. This is true of systems that must process data at a high sampling rate, such as audio or video processing pipelines. In this case,  $\lceil D_i R_i \rceil$  in the right hand side of Equation (3) could be replaced by  $D_i R_i + 1$  without a significant loss in accuracy, resulting in:

$$\frac{C_q^{eq}}{D_q} + \sum_i \frac{C_i^{eq}}{D_i} (D_i R_i + 1) \leq 1 \quad (4)$$

The above modified capacity expression errs on the safe side in that if a system is designed to operate in the capacity region identified by Inequality (4), it will also satisfy Inequality (3). Hence, all deadlines will be met. We can further ignore the contribution of the single job,  $q$ , to the utilization, since the large number of current jobs in the system entails that the summation term over these jobs dwarfs the contribution of  $q$  alone. Hence, we get:

$$\sum_i \frac{C_i^{eq}}{D_i} (D_i R_i + 1) \leq 1 \quad (5)$$

or:

$$\sum_i (C_i^{eq} R_i + \frac{C_i^{eq}}{D_i}) \leq 1 \quad (6)$$

Finally, note that  $C_i^{eq} R_i$  is the effective utilization due to task  $T_i$  on the equivalent uniprocessor. For brevity, let us refer to this term as *effective utilization*,  $u_i^{effect}$ :

$$u_i^{effect} = C_i^{eq} R_i \quad (7)$$

Substituting for  $u_i^{effect}$  from Equation (7) into Equation (6) leads to the following simple expression for the real-time capacity region (under EDF scheduling):

$$\sum_i (u_i^{effect} (1 + \frac{1}{R_i D_i})) \leq 1 \quad (8)$$

We summarize the above result as the following theorem:

**Theorem 1: The real-time capacity theorem:** *In a system with a set,  $S$ , of processing workflows, where each workflow  $F_i \in S$  incurs an effective utilization  $u_i^{effect}$  on an equivalent uniprocessor and has a job rate  $R_i$  and a per-job end-to-end maximum latency constraint,  $D_i$ , all jobs meet their end-to-end deadlines if:*

$$\sum_{F_i \in S} (u_i^{effect} (1 + \frac{1}{R_i D_i})) \leq 1 \quad (9)$$

where  $u_i^{effect}$  is given by Equation (7).

The theorem is trivially true from Equation (8). To use this theorem for analyzing a data fusion system, it is thus enough to reduce it to an equivalent uniprocessor. In the rest of this paper, such reductions are presented for pipelines and for aggregation trees.

It is interesting to note that for workflows with very large deadlines, the second term on the left-hand-side of Equation (9) becomes negligible, reducing the capacity region to:

$$\sum_{F_i \in S} u_i^{effect} \leq 1 \quad (10)$$

This result serves as a sanity check. It simply says that if deadlines are not an issue, only a resource capacity constraint must be met.

It remains only to quantify the values of  $u_i^{effect}$  on the equivalent uniprocessor for all workflows  $F_i$  in the original data fusion system. From Equation (7), this means  $C_i^{eq}$ . We first describe how to do so for pipelines. We then generalize the analysis to data fusion trees.

### C. Real-time Capacity of Pipelines

A theorem of particular relevance in the pipeline scenario is the *delay composition theorem* [4]. Consider a set of workflows traversing a pipeline,  $p$  (a single chain of processing stages), where the processing time of a job of workflow,  $F_i$ , on pipeline stage,  $j$ , is denoted by  $C_{i,j}$ . Let the priority of job  $q$  be denoted  $J$ . Let the function  $HP(J)$  denote all other jobs of priority equal to or higher than  $J$ . Let  $C_{max,j}$  be the maximum processing time  $C_{i,j}$  (on stage  $j$ ) of all jobs flowing through stage  $j$ , and  $C_{max,j}^{LP}$  be the maximum

processing time  $C_{i,j}$  (on stage  $j$ ) of all lower priority jobs flowing through stage  $j$  (i.e., all those *excluding* packets in  $HP(J)$ ). Finally, let  $C_{i,max}$  be the maximum  $C_{i,j}$  over all stages of flow  $F_i$ . The delay composition theorem is re-stated here as follows:

**Theorem 2:** [4] *For a non-preemptive scheduling policy, the end-to-end delay of a job  $q$  of priority  $J$  following path  $p$  can be composed using the execution parameters of jobs that delay it as follows<sup>3</sup>:*

$$\begin{aligned} Delay_{pipe} \leq & \sum_{k \in HP(J)} C_{i,max|k \in F_i} + \sum_{j \in p} C_{max,j} \\ & + \sum_{j \in p} C_{max,j}^{LP} \end{aligned} \quad (11)$$

In [5], it is accordingly shown that the latency of job  $q$  in the original pipeline has the same worst-case as the latency of an equivalent packet  $q^{eq}$  on a uniprocessor, if the following equivalent uniprocessor task set, denoted  $S_{uni}$ , is used:

- Job  $q^{eq}$ , of priority  $J$ , has a (uniprocessor) processing time equal to:

$$C^{eq} = \sum_{j \in p} (C_{max,j} + C_{max,j}^{LP}) \quad (12)$$

- For each equal or higher-priority job  $k \in HP(J)$  of workflow  $F_i$  that delays or preempts  $q$  in the original system, there is a job of the same priority as  $k$ , that delays or preempts  $q^{eq}$  on the uniprocessor and has a processing time equal to:

$$C_i^{eq} = C_{i,max} \quad (13)$$

To see why the equivalent task set,  $S_{uni}$ , leads to the same worst case latency for job  $q^{eq}$ , note that the worst-case latency of job  $q^{eq}$  on the uniprocessor is simply the summation of processing times of all jobs that delay or preempt its execution in  $S_{uni}$ :

$$Delay_{uni}^{eq} = C^{eq} + \sum_{k \in HP(J)} C_i^{eq|k \in F_i} \quad (14)$$

Substituting in Equation (14) from Equation (12) and Equation (13), we get:

$$\begin{aligned} Delay_{uni}^{eq} = & \sum_{k \in HP(J)} C_{i,max|k \in F_i} + \sum_{j \in p} C_{max,j} \\ & + \sum_{j \in p} C_{max,j}^{LP} \end{aligned} \quad (15)$$

Comparing this result to Equation (11), we get:

$$Delay_{uni}^{eq} = \max(Delay_{pipe}) \quad (16)$$

<sup>3</sup>The terminology of the original theorem [4] is somewhat simplified above to remove notations not of relevance to this paper.

Hence, job  $q$  on the pipeline and job  $q^{eq}$  on the uniprocessor have the same worst-case latency. The above defines the equivalent task set,  $S_{uni}$ , on a uniprocessor, that represents a *valid reduction* of data fusion pipelines. Note that, for pipelines that run a large number of tasks, the computation time  $C_{eq}$  can be neglected, leaving us only with the set of tasks  $C_{eq}^i$ . Substituting from Equation (13) into Equation (7), we get  $u_i^{effect} = C_{i,max}R_i$ . Hence, we have the following corollary to Theorem 1:

**Corollary 1: The pipeline real-time capacity theorem:** *In a system with a set,  $S$ , of processing workflows, where each workflow  $F_i \in S$  is a pipeline that has a job rate  $R_i$  and a per-job end-to-end maximum latency constraint,  $D_i$ , all jobs meet their end-to-end deadlines if:*

$$\sum_i (u_i^{effect} (1 + \frac{1}{R_i D_i})) \leq 1 \quad (17)$$

where:

$$u_i^{effect} = C_{i,max}R_i \quad (18)$$

$C_{i,max}$  being the maximum stage processing time of flow  $F_i$ .

#### D. Extensions to Merging Flows

In the previous section, we derived a general capacity expression and applied it to compute the basic real-time capacity of a pipeline. It is relatively straightforward to extend this result to the case of data fusion systems where different data flows merge at selected stages forming a tree. The key is to develop a valid reduction of this tree to an equivalent uniprocessor.

We consider the problem of reducing merging workflows to a task set on a uniprocessor that has the same worst-case end-to-end delay as the original workflow. First, we analyze the basic case of merging of  $H$  parallel pipelines, described by disjoint paths  $p_1, \dots, p_H$ , followed by a common parent stage,  $O$ , as shown in Figure 4-a.

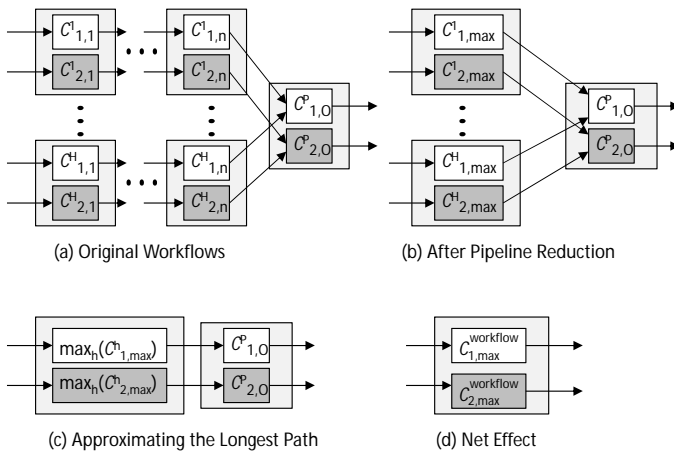


Figure 4. An example of merging flows.

A job,  $q$ , of workflow  $F_i$ , does not become eligible to execute on the last stage (the common parent,  $O$ ) until all pipelines have finished processing it. Using Equation (13), we can represent each child pipeline,  $p_h$ , by an equivalent uniprocessor node that represents  $F_i$  by an equivalent task of execution time  $C_{i,max}^h$  (denoting its maximum per-stage execution time over path  $p_h$ ). This reduction is shown in Figure 4-b. Note that the longest of the delays of the children of  $O$  is  $\max_h(\sum_i C_{i,max}^h)$ , where we know that  $\max_h \sum_i(x) \leq \sum_i(\max_h x)$ . Hence:

$$\max_h(\sum_i C_{i,max}^h) \leq \sum_i(\max_h C_{i,max}^h) \quad (19)$$

Hence, the children can be collectively approximated by a single node, where  $F_i$  is represented by a task of processing time  $\max_h C_{i,max}^h$ , as shown in Figure 4-c. According to Inequality (19), the reduced system over-estimates the latency. Therefore, the reduction is valid.

The reduced system is a two stage pipeline. Using Equation (13) again, we can reduce it to an equivalent single node. Observe that each workflow  $F_i$  is represented on that node by a task whose computation time is the maximum processing time over all stages of the workflow in the original distributed system, denoted  $C_{i,max}^{workflow}$ , as shown in Figure 4-d.

In the presence of successive pipeline merges at multiple stages, the reduction above can be applied recursively as merged branches of a workflow merge again. Hence, it is easy to show that  $C_i^{eq} = C_{i,max}^{workflow}$  applies for any number of merges in the tree. Substituting for  $C_i^{eq}$  into Equation (7), we get  $u_i^{bottle} = C_{i,max}^{workflow}R_i$ . Together with Theorem 1, this results in the following corollary.

**Corollary 2: The tree real-time capacity theorem:** *In a system with a set,  $S$ , of processing workflows, where each workflow  $F_i \in S$  is represented by a tree of stages merging in a single parent, and where the job rate is  $R_i$  and the end-to-end maximum latency constraint is  $D_i$ , all jobs meet their end-to-end deadlines if:*

$$\sum_i (u_i^{effect} (1 + \frac{1}{R_i D_i})) \leq 1 \quad (20)$$

where:

$$u_i^{effect} = C_{i,max}^{workflow}R_i \quad (21)$$

The above results applies to any tree topology and a fluid task model.

#### IV. ILLUSTRATIVE EXAMPLE

In this section, we work out a numerical example to illustrate how the real-time capacity is calculated and used for a system comprising of resources arranged as a workflow tree. We consider three workflows  $F_1, F_2$ , and  $F_3$ , in decreasing priority order traversing the tree with 7 stages as shown in Figure 5-a. The computation times of the three workflows at each stage are shown. Let us further assume that  $R_1 = 0.1$ ,

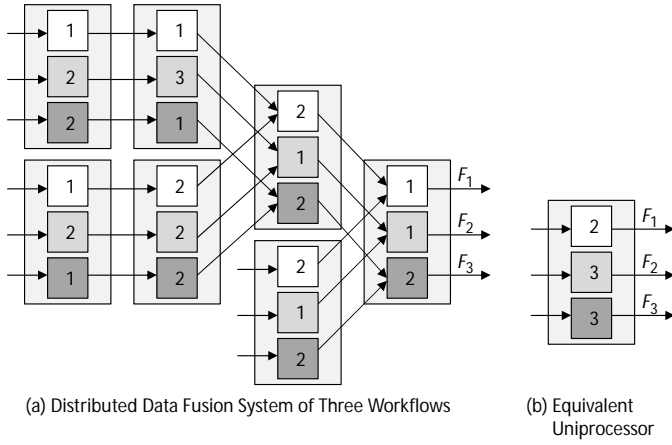


Figure 5. Figure illustrating the steps in the reduction of the system to a single node.

$R_2 = 0.05$ , and  $R_3 = 0.02$ . Also, let us choose  $D_1 = 100$ ,  $D_2 = 200$ , and  $D_3 = 250$  time units.

To compute the schedulability of this system, we reduce the system to an equivalent uniprocessor. Note that the computation times of the three workflows on the final equivalent uniprocessor are simply their maximum computation time across all the stages in the original distributed system, as observed in Corollary 2.

To determine whether the workflow set falls within the schedulable capacity region, we use Equation (20):

$$\sum_i (w_i^{effect} (1 + \frac{1}{R_i D_i})) \leq 1$$

$$0.2(1 + \frac{1}{10}) + 0.15(1 + \frac{1}{10}) + 0.06(1 + \frac{1}{5}) \leq 1$$

$$0.457 < 1$$

As the left hand side is less than 1, we declare that all the workflows will meet their respective end-to-end deadlines.

## V. EVALUATION

In this section, we evaluate our approach to determine how accurately our worst-case analysis predicts the onset of deadline misses in practice. We construct a simulator that models a distributed data fusion system. In this simulator, we construct an admission controller which admits as many data fusion flows as it can deem feasible based on the capacity expression developed in this paper. Systems where workload is limited to what the admission controller deems feasible represent “well-designed” systems that are guaranteed to meet deadlines in the worst-case. Since designing for the worst-case may be pessimistic, we experiment with increasing the load beyond the capacity region, exploring what happens to the ability of the fusion system to deliver packets on time. This helps us to quantify the distance between the boundary predicted by the analytically derived capacity region and the real onset of deadline misses in practice.

We compare the results of simulation for when the admission controller is on (denoted as AC) and when it is off (denoted as w/o AC). In the AC approach, we generate flows one at a time until the admission controller cannot accept more, hence operating at the analytically derived capacity boundary. In the absence of the admission controller, a fixed number of flows are simulated. We then compute the amount of “timely work” done. Two methods are employed for calculating timely work. In the first method, which we denote as “w/o AC - Discounting job misses,” each job that misses its deadline is dropped and discounted from calculations of useful work. In our second method, “w/o AC - Discounting flow misses,” if a job  $q \in F_i$  misses its deadline, we do not include any job belonging to workflow  $F_i$  in our calculations of useful work.

### A. Evaluation of Pipeline Capacity

In the first set of results, we explore the capacity boundary of pipelines. We run a simulated networked pipeline for a duration approximately 1000 times the longest deadline. Each data point in the results represents the average across 50 such runs. The periods of data workflows (time between successive job invocations) are chosen proportionally to  $(10^\alpha \times N)$  wherein  $N$  denotes the number of stages in the pipeline and  $\alpha$  is uniformly distributed over  $[0, 2]$ , giving two orders of variability in periods. The deadline of each workflow is then assigned such that the ratio of deadline to workflow period is uniformly selected from  $[0.3, 3.0]$ . The processing time of each job on each stage is then given a value from a uniform distribution with an average that is significantly smaller than the period.

We begin by quantifying how the number of workflows in the pipeline affects the average per-stage utilization attributed to *timely work* in the pipeline. Toward this goal, we vary the number of workflows in a simulated 8-stage pipeline. As shown in Figure 6, the utilization of the networked system initially increases in proportion to the number of flows. However, when the number of flows becomes large, the utilization w/o AC drops significantly, while the utilization with AC remains nearly constant. This is because jobs in large workflow sets miss deadlines and are discarded in the absence of admission control. This results in a decrease in the amount of timely work completed.

We now turn to evaluating the relationship between the pipeline length amount of timely work completed when the capacity bound is used for admission control. We demonstrate that the capacity bound is scalable—the bound remains accurate as the number of stages is increased. In Figure 7, we vary the number of stages in a pipeline while holding other parameters constant. Each workflow has a deadline uniformly chosen from between 0.3 and 3 of its period, and a total of 100 workflows were used.

The results of this experiment show that, as the pipeline grows, the per-stage utilization attributed to timely work decreases. We also find that deadlines are missed when the load is increased moderately beyond the admission controller’s capacity region. These results demonstrate that the capacity

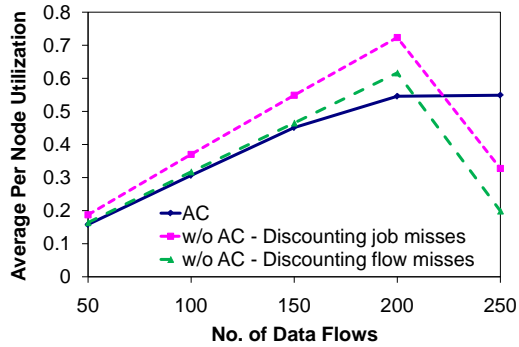


Figure 6. Comparison of AC and w/o AC with respect to the number of flows in the networked data fusion pipeline.

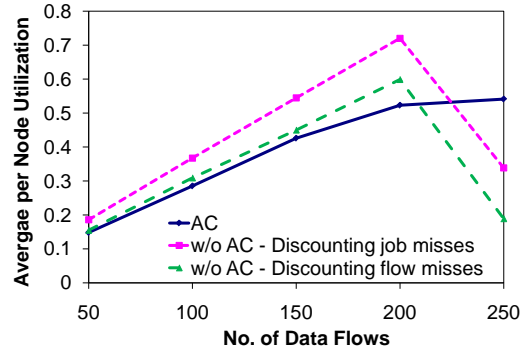


Figure 8. Comparison of AC and w/o AC with respect to the number of flows in the networked data fusion tree.

bound maintains its accuracy as the number of stages is increased. One cannot increase the amount of timely work significantly beyond what is predicted by the analysis.

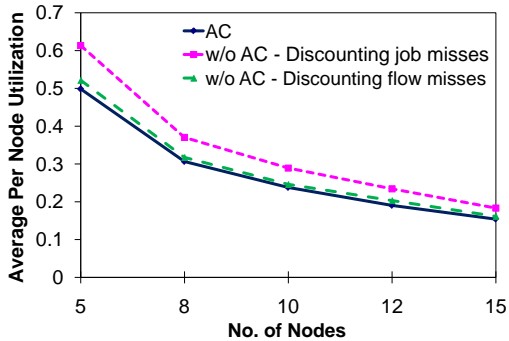


Figure 7. Comparison of AC and w/o AC with respect to the number of stages in the networked data fusion pipeline.

### B. Evaluation of Merging Pipelines

We further present results for a data fusion tree, where multiple pipelines ultimately merge into one aggregation stage. As in our evaluation of single-path pipelines, we begin by considering how the number of flows influences the measure average per-stage utilization attributed to timely work (Figure 8). We also evaluate the effect of the number of stages in the longest path on timely work (Figure 9). The key observation is that the results for merging flows are very similar to the corresponding trends for single-path pipelines (Figures 6 and 7). This is expected, since the performance of a merging flow is dominated by that of the longest pipeline path.

## VI. RELATED WORK

This paper defines real-time capacity of data fusion systems. Deterministic worst-case analysis is carried out that identifies the boundaries of a region where all workflow deadlines are guaranteed to be met. While a significant amount of work is done on capacity analysis in various settings, we focus below

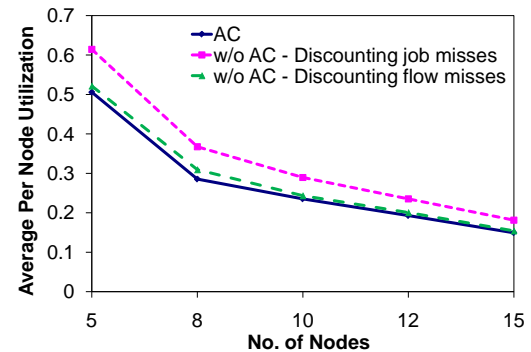


Figure 9. Comparison of AC and w/o AC with respect to the number of stages in the networked data fusion tree.

only on work that (i) provides deterministic guarantees (as opposed to probabilistic), (ii) involves deadlines as a main constraint, and (iii) performs worst-case analysis.

Algorithms such as [6] and [7] have been proposed to statically schedule precedence-constrained tasks in distributed systems. A schedule of length equal to the least common multiple of the task periods is constructed, that would precisely define the time intervals of execution of each job. These algorithms have exponential time complexity. While they ultimately define the boundaries of what is feasible (in terms of meeting deadlines) they do not provide simple intuitive expressions and involve NP-hard computation.

Offline schedulability tests have been proposed that divide the end-to-end deadlines of tasks into per-stage deadlines. The end-to-end task is then considered as several independent sub-tasks, each executing on a single stage in the system. Uniprocessor schedulability tests are then used to determine if each stage is schedulable. If all the stages are schedulable, the system is deemed to be schedulable. For instance, Kao et al. [8] and Zhang et al. [9] present techniques to divide the end-to-end deadline into per-stage deadlines. This technique tends to be extremely pessimistic in estimating the boundaries of schedulability and does not accurately account for the inherent parallelism in the execution of different stages, which is a problem in data fusion systems where performance is

enhanced due to data pipelining.

Holistic analysis was first proposed in [10], and has since had several extensions such as [11] and [12] that propose offset-based response time analysis techniques for EDF. In addition to the computation time and period, tasks are characterized by two other parameters, namely the jitter and the offset. The fundamental principle behind holistic analysis and its extensions is that, given the jitter and offset information of jobs arriving at a stage one can compute (in a worst-case manner) the jitter and offset for jobs leaving the stage, which in turn becomes the arrival pattern for jobs to a subsequent stage. By successively applying this process to each stage in the system, one can compute a worst-case bound on the end-to-end delay of jobs. An iterative procedure is described in [11] and [12] which is shown to converge. This solution technique, however, becomes tedious, complicated and quite pessimistic for large task graphs with tens of nodes, as was shown in prior work [4].

From the networking perspective, network calculus [13], [14] was proposed to analyze the end-to-end delay of packets of flows. This analysis was adapted for the context of real-time systems, and the new analysis was named *real-time calculus* when first presented in [15]. Real-time calculus has since been extended to handle different system models such as [16], [17]. In approaches based on network calculus, the arrival pattern of jobs of flows is characterized by an arrival curve. Given a service curve for a node based on the scheduling policy used, one can determine the rate at which jobs leave the node after completing execution, which in turn serve as the arrival curve for the next stage in the flow's path. Needless to say, there is no means by which the solution can be efficiently automated for arbitrary task graphs. A comparison of holistic analysis and network calculus was conducted in [18], where holistic analysis was found to be less pessimistic than network calculus in general.

Our work is built on results by two of the authors that bound the worst-case end-to-end delay of jobs in pipelined systems under preemptive and non-preemptive scheduling [1], [2], as well as their extensions to DAGs [3] and graphs with cycles [4]. In this paper, we build on these results to derive a simple closed-form expression that directly defines the capacity boundary. This is the first real-time capacity expression of its kind for data fusion pipelines. Simulation shows that the boundary is predicted fairly accurately in that deadlines are missed soon after system load exceeds the capacity region.

## VII. CONCLUSIONS

This paper presents real-time capacity expressions for data fusion pipelines. These expressions derive the amount of load that a pipeline can handle while meeting the end-to-end deadlines of all data processing workflows. A simple capacity region is derived for pipelines using recent results in real-time scheduling theory. The expression is then extended to data fusion systems involving merging flows. Evaluation results show that our capacity expressions are accurate in their ability

to delineate load regions where deadlines are met from those where they are not.

## 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, "A delay composition theorem for real-time pipelines," in *ECRTS*, July 2007, pp. 29–38.
- [2] 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.
- [3] —, "Transforming acyclic distributed systems into equivalent uniprocessors under preemptive and non-preemptive scheduling," in *ECRTS*, July 2008, pp. 233–242.
- [4] P. Jayachandran and T. Abdelzaher.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [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] 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.
- [14] —, "A calculus for network delay, part ii: Network analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132–141, January 1991.
- [15] 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.
- [16] B. Jonsson, S. Perathoner, L. Thiele, and W. Yi, "Cyclic dependencies in modular performance analysis," in *ACM EMSOFT*, Oct. 2008, pp. 179–188.
- [17] 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.
- [18] 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.