Latency-Rate servers & Dataflow models

Maarten Wiggers Marco Bekooij

September 28, 2006

1 Introduction

In the signal processing domain, dataflow graphs [2] [10] and their associated analysis techniques are a well-accepted modeling paradigm. The vertices of a dataflow graph represent functionality and are called actors, while the edges model which actors communicate with each other. Traditionally, these actors have been scheduled in a static order or in a fully static fashion [9] [16]. However, for firm real-time applications, run-time scheduling is required if there are tasks, that operate on different streams and share the same resource, for which either the execution time or the execution rate is unknown or impractical to bound.

Furthermore a fully static schedule is not attractive since computing a fully static schedule is NP-hard [6], storing the fully static schedule can require a large on-chip memory, it requires a global notion of time, and a schedule for each combination of applications, and when the actual execution time is smaller than the worst-case execution time then this leads to idle time.

Static order scheduling does not intrinsically require a conservative bound on the execution times. However, in order to determine the worst-case response time of a task, a conservative bound on the execution time is required to be known of all tasks that share the same resource. Static order scheduling does, however, require a known execution rate. And static order scheduling also requires a schedule for each combination of applications.

This paper presents a method to perform throughput analysis of firm real-time applications when the tasks that constitute these applications are scheduled at run-time. More precisely, we will show that it is possible to include the effects of a class of schedulers, which are called Latency-Rate (LR) servers [17], in a dataflow graph, and we further show that traditional analysis techniques can still be applied on the resulting dataflow graph.

LR servers were originally proposed as a modeling paradigm to model the effect of scheduling on traffic passing through a chain of routers in a network with real-time constraints. The analysis goal of the LR model is to be able to determine whether traffic that is injected into the network according to a specific traffic model arrives at its destination in time, and determine sufficient buffer capacities. The presented work removes a number of limitations of LR servers.

We allow dependencies between more than two LR servers, in which a request for service only arrives when requests on possibly multiple LR servers have received service. An example would be a task that combines results from two different tasks, this task is only enabled if both its inputs are present.

In a dataflow model, cyclic dependencies can be analysed. As will be explained in later sections, this allows us to analyse task graphs in which the tasks are scheduled on LR servers and in which tasks are only enabled if both data is available on the input buffers and output space is present in the output buffers. This mechanism results in back-pressure, which will not enable the task if any of its output buffers is filled with data. We will show that application of this backpressure mechanism leads to reduced buffer capacities.

This document is structured as follows. First in Section 2 we briefly present work related to worst-case analysis of task graphs that include tasks that are scheduled at run-time. Then in Section 3 we present the single-rate dataflow model that we in subsequent sections use to derive our initial results. In Section 4 we describe the traditional relation between implementation and dataflow models. A more general relation between implementation and its dataflow model is described in Section 5. In Section 6 we introduce LR servers, and in Section 7 we show a dataflow component of which we show that this component has an input-output behaviour that is equivalent to the LR model of a task that is scheduled on an LR server. In Section 8 we show how the application of LR servers improves upon our previous work. And in Section 9 we show that dataflow analysis leads to smaller buffer capacities than LR analysis. Then in Section 10 we show that dataflow subgraphs can be conservatively modelled by other dataflow subgraphs, which allows to create a hierarchical model. In Section 11 we show how to construct these abstract dataflow subgraphs. And in Section 12 we generalise the results obtained using single-rate dataflow to more expressive dataflow models.

2 Related work

Network calculus, as presented by Cruz [3] [4], is related to the LR model. However as explained in [17], since the LR model uses the concept of a busy period instead of a backlogged period, the LR model leads to tighter server latency bounds and a larger class of schedulers that can provide these latency bounds, when the offered service is linearly bounded.

Not only the LR model and Network calculus, but also other scheduling approaches that include run-time scheduling, as for instance presented by Jersak [8], Goddard [7], or Maxiaguine [12], do not allow feedback cycles that influence the temporal behaviour of the system.

Bekooij [1] previously included resource sharing in dataflow models, however, only Time Division Multiplex and Round Robin scheduling are considered, while the class of LR servers is broader. And further, the analysis approach as proposed in [1] is less accurate than the analysis approach presented in this paper. This is because [1] assumes that the worst-case response time is the same for every execution of a task, while we will show that for sequences of task executions that immediately follow each other this assumption is overly pessimistic.

3 Single-Rate Dataflow

In this section we define Single-Rate Dataflow (SRDF) graphs and summerise a number of relevant properties of SRDF graphs. An SRDF graph is a directed graph $G = (V, E, d, r)$ that consists of a finite set of actors V, and a set of directed edges, $E = \{(v_i, v_j)|v_i, v_j \in V\}$. Actors synchronise by communicating tokens over edges that represent queues. The graph G has an initial token placement $d : E \rightarrow \mathbb{N}$.

An actor is enabled to fire when a token is available on each input edge. The response time $r(v_i)$, $r : V \rightarrow \mathbb{R}$, is the difference between the finish and start time of actor v_i. When actor v_i finishes, then it produces a token on each output edge in one atomic action.

For a strongly connected and consistent SRDF graph, we can derive the period μ of the graph through Maximum Cycle Mean (MCM) analysis [16]. To determine the MCM, the maximum of the cycle means of all simple cycles in the SRDF graph needs to be determined, where the cycle mean of a cycle c is the sum of the response times of the actors on c divided by the number of initial tokens on the cycle c. The throughput of the graph relates to μ^{-1}.

3.1 Monotonic Execution

In this section we will show that SRDF graphs are temporally monotonic.

Definition 1 (temporal monotonicity) *An SRDF graph is temporally monotonic if no decrease in response time, increase in number of initial tokens, or decrease in the difference between actor enabling time and actor start time leads to a later actor start time.*

In [14] it is stated that an SRDF graph is monotonic in the response times, if the SRDF graphs maintains a FIFO ordering of tokens. An SRDF graph maintains a FIFO ordering of tokens if each actor either has a constant response time or a self-edge with one initial token [5]. This is because the queues by definition maintain a FIFO ordering.

No proof was provided in [14] to show that an SRDF graph is monotonic in the response times, only the observation that the evolution equations as given in Chapter 2 of [5] are monotonic in the response times. These evolution equations only hold if the SRDF graph maintains a FIFO ordering of tokens. This is because in the evolution equations the start time of the k'th firing of actor v_j depends on firings $k - m, 0 \leq m \leq M$ of actors v_i, where M is the maximum number of initial tokens on any edge. If FIFO ordering is not maintained, then firing k of actor v_j is enabled by firing q of actor v_k, where q can be larger than, smaller than, or equal to k.

We will first extend [14] by proving that an SRDF graph that maintains FIFO ordering of tokens is temporally monotonic. Subsequently we will show that the requirement of FIFO ordering is not necessary.

Theorem 1 *An SRDF graph that maintains a FIFO ordering of tokens is temporally monotonic.*

Proof. Since the SRDF graph maintains a FIFO ordering of tokens, the enabling time of actor v_x in iteration k, $l(v_x, k)$, is given by [15]

$$l(v_x, k) = \max(\{s(v_i, k - d(v_i, v_x)) + r(v_i, k - d(v_i, v_x))|(i, x) \in E\}) \qquad (1)$$

where $s(v_i, k)$ is the start time of actor v_i in iteration k, and $r(v_i, k - d(v_i, v_x))$ is the response time of actor v_i in iteration $k - d(v_i, v_x)$. The start time of actor v_x in iteration k is therefore given by

$$s(v_x, k) = l(v_x, k) + \epsilon(v_x, k) \qquad (2)$$

where $\epsilon(v_x, k) \geq 0$ is the difference between the start time and the enabling time of actor v_x in iteration k.

Since $a' \leq a \Rightarrow \forall b : \max(a', b) \leq \max(a, b)$, a decrease of the start times or response times of v_i, which is a predecessor of v_x, in iteration $k - d(v_i, v_x)$ cannot lead to an increase of the enabling time of actor v_x in iteration k, and therefore also cannot lead to an increase of the start time of actor v_x in iteration k.

If the number of initial tokens $d(v_i, v_x)$ is increased of some edge (v_i, v_x) to become $d'(v_i, v_x)$, with $d'(v_i, v_x) > d(v_i, v_x)$, then the enabling time of v_x in iteration k becomes dependent on the start time and response time of actor v_i in iteration $k - d'(v_i, v_x)$. Since FIFO ordering is maintained we have that $k - d'(v_i, v_x) < k - d(v_i, v_x) \Rightarrow s(v_i, k - d'(v_i, v_x)) + r(v_i, k - d'(v_i, v_x)) < s(v_i, k - d(v_i, v_x)) + r(v_i, k - d(v_i, v_x)$. This means that an increase in the number of initial tokens cannot lead to an increase of the enabling time of actor v_x in iteration k, and therefore also cannot lead to an increase of the start time of actor v_x in iteration k.

Further if the difference $\epsilon(v_x, k)$ between the start time and enabling time of actor v_x in iteration k is decreased to become $\epsilon'(v_x, k), 0 \leq \epsilon'(v_x, k) < \epsilon(v_x, k)$, then this does not lead to an increase of the start time of actor v_x in iteration k.

Since for any actor v_x and any iteration k no decrease of the response time, increase in the number of initial tokens, or decrease of the difference between start time and enabling time leads to a later start time, the SRDF graph is temporally monotonic. $\square$

We will now show that an SRDF graphs are also temporally monotonic without the constraint that FIFO ordering is maintained.

Theorem 2 *An SRDF graph is temporally monotonic.*

Proof. The enabling time of actor v_x in iteration k, $l(v_x, k)$, is determined by

$$l(v_x, k) = \max(\{s(v_i, k') + r(v_i, k')|(i, x) \in E\}) \qquad (3)$$

where $s(v_i, k')$ is the start time of actor v_i and $r(v_i, k')$ is the response time of actor v_i in some iteration k'. The start time of actor v_x in iteration k is therefore given by

$$s(v_x, k) = l(v_x, k) + \epsilon(v_x, k) \qquad (4)$$

where $\epsilon(v_x, k) \geq 0$ is the difference between the enabling time and the start time of actor v_x in iteration k.

Since $a' \leq a \Rightarrow \forall b : \max(a', b) \leq \max(a, b)$, a decrease of the start times or response times of v_i, which is a predecessor of v_x, in iteration k' cannot lead to an increase of the enabling time of actor v_x in iteration k. Since the SRDF graph does not maintain FIFO ordering of tokens, it can occur that a decrease of the start time or response times of an actor v_i, which is a predessor of v_x, in iteration k'' determines the enabling time of v_x in iteration k. This can only occur if iteration k'' of v_i produces earlier than iteration k' and thus $s(v_i, k'') + r(v_i, k'') < s(v_i, k') + r(v_i, k')$. This situation can therefore not lead to a later enabling time of actor v_x in iteration k, and therefore also not to a later start time of actor v_x in iteration k.

If the number of initial tokens $d(v_i, v_x)$ is increased of some edge (v_i, v_x) to become $d'(v_i, v_x)$, with $d'(v_i, v_x) > d(v_i, v_x)$, then the enabling time of v_x in iteration k becomes dependent on the start time and response time of actor v_i in iteration $\widehat{k}, \widehat{k} \neq k'$, with $s(v_i, \widehat{k}) + r(v_i, \widehat{k}) \leq s(v_i, k) + r(v_i, k)$. This is because firing k is no longer dependent on the $k - d(v_i, v_x)$'th token to arrive on edge (v_i, v_x) but on the $k - d'(v_i, v_x)$'th token to arrive. Since $d'(v_i, v_x) > d(v_i, v_x)$, the $k - d'(v_i, v_x)$'th token arrives no later than the $k - d(v_i, v_x)$'th token.

Further if the difference $\epsilon(v_x, k)$ between the start time and enabling time of actor v_x in iteration k is decreased to become $\epsilon'(v_x, k), 0 \leq \epsilon'(v_x, k) < \epsilon(v_x, k)$, then this does not lead to an increase of the start time of actor v_x in iteration k.

Since for any actor v_x and any iteration k no decrease of the response time, increase in the number of initial tokens, or decrease of the difference between start time and enabling time leads to a later start time, the SRDF graph is temporally monotonic. □

4 Analysis model and implementation

In this section we show how traditionally an SRDF graph is constructed from the implementation of our applications, and we show that the analysis of the resulting SRDF graph gives a conservative bound on the behaviour of the implementation.

We assume the following. Our applications are implemented as a weakly connected directed task graph, of which the vertices represent tasks and the edges represent FIFO buffers with a fixed capacity. Tasks only communicate fixed sized containers over FIFO buffers, where a container can be full or empty. A task produces one container on each output FIFO and consumes one container from each input FIFO in every execution. Furthermore, the execution of a task only starts when a full container is present on every input FIFO buffer and an empty container is present on every output FIFO buffers. The finish time of each task is at most the worst case response time later than the enabling time. And further at most one instance of a task can execute at any time.

The corresponding SRDF graph can now be constructed as follows. Each task is modelled by an actor. The response time of the actor equals the worst-case response time of the task. Containers are represented by tokens. Each FIFO buffer is modelled by a pair of queues in opposite direction, where the tokens on one queue represent full containers, and on the other queue represent empty containers. The fixed capacity of a FIFO buffer is modelled by the number of initial tokens on the corresponding pair of queues. The number of containers communicated per task execution is modelled by the token production and token consumption rates of the actor, which are all equal to one in an SRDF graph. The constraint that no two instances of a task can execute simultanuously is represented by an edge from the actor to itself, a self-cycle, with one initial token.

In Section 3 we have established that an SRDF graph is temporally monotonic. This means that analysis of an SRDF graph where every actor has a worst-case response time leads to worst-case token arrival times. Furthermore, we have that the SRDF graph is constructed in such a way that there is a one-to-one correspondance between tokens and containers. And because tasks can only have a smaller response time than actors, while the number of tokens equals the FIFO buffer capacity and in the implementation tasks start their execution as soon as they are enabled, we arrive at the conclusion that tasks produce their containers no later than the corresponding actors produce their tokens.

5 Analysis of tasks on LR servers

While in Section 4 every task is modelled by one actor, we will in later sections like to model a task that is scheduled on a LR server with two actors v_y and v_z. However, since in that case there is no longer a clear correspondence between queues and tokens in the SRDF graph and FIFO buffers and containers in the task graph, we can no longer use the argumentation of Section 4. In this section, we will show that a correspondence between model and implementation and a sufficient condition exists such that worst-case container production times can still be computed with dataflow analysis techniques.

We assume that the application is implemented as a task graph G_A as described in Section 4.

We define $a_A(m, j)$ to be the arrival time of the j-th container in the input FIFO $m \in I_x$ and $a_A(n, j), n \in O_x$ to be the arrival time of the j-th container in the output FIFO n both of a task u_x in the implementation.

We assume an SRDF graph G_M that models the implementation. We define C as a partitioning of the set of actors V, $\forall C_x, C_y \in C, C_x \neq C_y \Rightarrow C_x \cap C_y = \emptyset, V = \{C_i | C_i \in C\}$. Each element in C is called a component. Components consume tokens from component input queues and produce tokens on component output queues. A component input queue of component C_x is an edge (i, j) in the SRDF graph that has an actor $v_j \in C_x$ as destination and an actor $v_i \notin C_x$ as its source. A component output queue of component C_x is an edge (i, j) in the SRDF graph that has an actor $v_i \in C_x$ as its source and an actor $v_j \notin C_x$ as its destination.

The component graph needs to match with the task graph, i.e. there should be a one-to-one correspondence between tasks and components, and between bounded FIFO buffers in the task graph and pairs of edges in the component graph. We define I_x as the set of input queues of component C_x, and O_x as the set of output queues of C_x.

We define $a_M(m, j)$ to be the arrival time of the j-th token in the component input queue $m \in I_x$ and $a_M(n, j), n \in O_x$ to be the arrival time of the j-th token in the component output queue n both of a component C_x in the SRDF graph.

Definition 2 *We say that the token production and consumption behaviour of component C_x is conservative with respect to the container production and consumption behaviour of task u_x, if the following holds*

$$\forall m \in I_x, a_A(m, j) \leq a_M(m, j) \Rightarrow \forall n \in O_x, a_A(n, j) \leq a_M(n, j) \tag{5}$$

Theorem 3 *If every component C_x is conservative with respect to task u_x, i.e. Equation 5 holds, then the worst-case arrival times of containers in the output FIFOs of every task u_x can be computed with dataflow analysis techniques.*

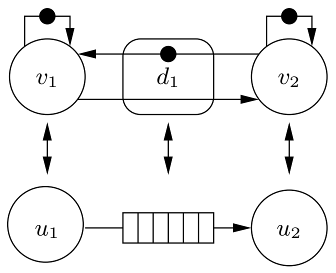
Proof. We know that (1) Equation 5 holds, (2) every SRDF graph is temporally monotonic, and (3) there is a one-to-one correspondence between the component graph and the task graph. The one-to-one correspondence combined with the fact that Equation 5 holds, means that for constant response times token arrival times form a conservative bound on container arrival times. This is because tokens are produced when the actor finishes, while containers are produced before a task finishes. Furthermore we know that an SRDF graph is temporally monotonic, which means that no reduction of the response time of an actor in the component graph leads to a later start time of an actor in the component graph. Combining this fact with the facts that the components are conservative with respect to the tasks and that there is a one-to-one correspondence between the component graph and the task graph, leads to the conclusion that token arrival times in the component graph are conservative container arrival times in the task graph, and are therefore worst-case container arrival times. □

An implication of Theorem 3 is that the execution of one task in the system can be represented by a component which can be a collection of dataflow actors as long as the component is conservative with respect to the task. In the next sections, we will use this property to model tasks that are scheduled using an LR server with a component that consists of two dataflow actors, similar to Figure 1(b), of which we will show that this component conservatively models the execution a task on an LR server. In the traditional approach as for example shown in Figure 1(a) this relationship between model and implementation was not possible.

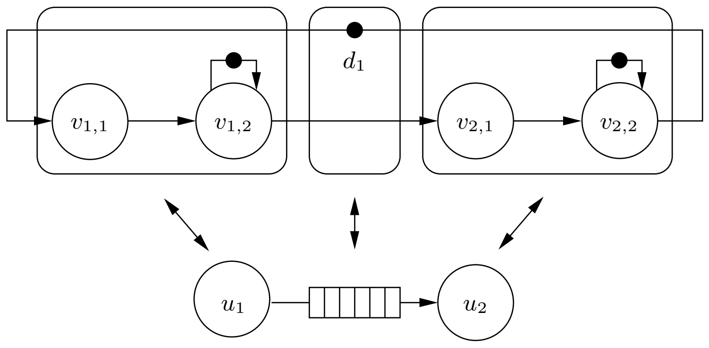
6 Latency-rate servers

In this section we introduce LR servers from a dataflow perspective. Let $e(v_x, i), e : U \times \mathbb{N} \rightarrow \mathbb{R}$, with U the set of tasks, be the time at which sufficient containers are available on all input FIFOs, such that execution i of task u_x can start, and say that execution i of task u_x is externally enabled at $e(v_x, i)$. This is because a task is only enabled if also its previous execution has finished. We define $A_x(s, t), A : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{N}$, as the number of external enablings of task u_x within the interval $(s, t]$. Further, let $f(u_x, i), f : U \times \mathbb{N} \rightarrow \mathbb{R}$, be the finish time of execution i of task u_x. Let $W_x(s, t), W : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{N}$ be the number of finishes of task u_x in the interval $(s, t]$.

A busy period is defined in [17] as a maximum interval of time $(s, t]$ for which Equation 6 holds. This means that at any time τ in a busy period the number of external enablings is at least equal to the number of finishes at the allocated rate $\rho_x \in \mathbb{R}$.



(a) Example of the traditional one-to-one relation between dataflow graph and task graph



(b) Example one-to-one relation between component graph and task graph

Figure 1: Examples of the traditional and proposed relations between dataflow graphs and task graphs.

$$\forall \tau \in (s,t] : A_x(s,\tau) \geq E(s,\tau) = \rho_x \cdot (\tau - s) \tag{6}$$

According to [17], if and only if we can show for a scheduler that a nonnegative constant $L_x \in \mathbb{R}$ can be found such that Equation 7 holds, where $(s,t]$ is a busy period of task u_x and $\tau \in (s,t]$, then this scheduler is a LR server for task u_x with rate ρ_x. The minimum nonnegative constant L_x, for which Equation 7 holds, is defined to be the latency Θ_x of the scheduler for task u_x.

$$W_x(s,\tau) \geq Q(s,\tau) = \max(0, \rho_x \cdot (\tau - s - L_x)) \tag{7}$$

Since $W_x(s,t) \in \mathbb{N}$, we obtain a tighter lower bound $\breve{Q}$ on $W_x(s,t)$, by enforcing that $\breve{Q} \in \mathbb{N}$. This is done by taking the floor of Q and we therefore require that a nonnegative constant L_x can be found such that Equation 8 holds. And again, the minimum nonnegative constant L_x, for which Equation 7 holds, is defined to be the latency Θ_x of the scheduler for task u_x.

$$W_x(s,\tau) \geq \breve{Q}(s,\tau) = \max(0, \lfloor \rho_x \cdot (\tau - s - L_x) \rfloor) \tag{8}$$

Because $W_x(s,t) \in \mathbb{N}$, using the bound $\breve{Q}(s,t)$ leads to a latency L'_x that is $\frac{1}{\rho}$ lower than the latency L''_x derived using the bound $Q(s,t)$. This is because when using $\breve{Q}(s,t)$, we have that $\breve{Q}_x(s,\tau) \leq n$ for $\tau < s + L'_x + \frac{n+1}{\rho_x}$, while when using $Q(s,t)$, we have that $Q_x(s,\tau) \leq n$ for $\tau < s + L''_x + \frac{n}{\rho_x}$. Since at the finish time of some execution m both bounds are exact, we have that $L'_x = L''_x - \frac{1}{\rho_x}$. In [17] latencies of a number of LR-servers are derived using the bound $Q(s,t)$.

Figure 2 illustrates the relation between the number of external enablings and the number of finishes.

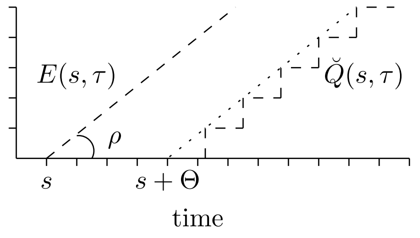


Figure 2: The lines starting at s and $s + \Theta$ both have a slope ρ. The number of external enablings should be at least $E(s,\tau)$ and the number of finishes should be at least $\breve{Q}(s,\tau)$.

Often the arrival function A and service function W are expressed in the amount of time requested from and provided by the LR server, while we have defined them in terms of number of enablings and finishes of task executions. However, in worst-case analysis these two notions are equivalent. This is because, in worst-case analysis we can associate the worst case execution time (WCET), which is the maximum time required by a task execution when the task executes in isolation, with every request or provision of service. If with every arrival and provision of service the WCET is associated, then ρ_x would be multiplied by the WCET to become $\rho'_x = \rho_x \cdot WCET_x$, while Θ_x remains the same. In this case, ρ'_x would mean the fraction of time that a requester is allowed to execute on the LR server, i.e. after Θ_x time, which corresponds with the traditional notion of rate. Therefore $\frac{1}{\rho'_x}$ would mean the fraction with which the execution time of the requested is stretched due to execution on the LR server, excluding Θ_x. And therefore $\frac{WCET}{\rho'_x}$ would be the worst case time required to finish a request for service, excluding Θ_x. However, we have that $\frac{WCET}{\rho'_x} = \frac{WCET}{\rho_x \cdot WCET} = \frac{1}{\rho_x}$, which means that the definitions as used in the present discussion are equivalent to the traditionally used definitions.

7 LR server as an SRDF Component

In this section we present an SRDF component that can be included into a larger SRDF graph that conservatively models a task that executes on an LR server by showing that the number of token releases by this dataflow construct in an interval $(s, t]$ equals $\check{Q}(s, t)$ as defined in Equation 8.

We will first show that bounding the finish time of an execution in a busy period with respect to the start of the busy period as expressed by Equation 9 is equivalent to bounding the number of finishes over a time interval that starts from the start of the busy period as expressed by Equation 8. Subsequently we show that bounding the finish time of an execution in a busy period with respect to the start of the busy period as expressed by Equation 9 is equivalent to bounding the finish time of an execution in a busy period with respect to either its external enabling time or the finish time of the previous execution as expressed by Equation 31. Subsequently we will show that Equation 31 is equivalent to an SRDF component that models a task with one input and one output FIFO that is scheduled on an LR scheduler.

In the following we will only consider task u_x and we therefore use the shorthand notation $e(j)$ and $f(j)$ to mean the external enabling time of execution j of task u_x and the finish time of task u_x.

Lemma 1 *If the scheduler is a LR server with latency Θ and rate ρ, then bounding the finish time as in Equation 9 is equivalent to bounding the number of finishes as in Equation 8 in every busy period.*

Proof. Let execution k be the first execution in a busy period $(s, t]$ and let execution j occur in the same busy period, that is $e(k) = s$ and $j \geq k$ and $f(j) \leq t$, then the finish time of execution j is bounded by

$$f(j) \leq \tau_j = e(k) + \Theta + \frac{j - k + 1}{\rho} \tag{9}$$

According to Equation 9, execution j of actor v_x is guaranteed to have finished at $\tau_j = e(k) + \Theta + \frac{j-k+1}{\rho}$. For Equation 9 to equal Equation 8, it should hold that starting from τ_j the guaranteed number of finishes is at least $j - k + 1$. More formally it should hold that

$$\forall \tau' \geq \tau_j : \check{Q}(e(k), \tau') \geq j - k + 1, \text{and} \tag{10}$$

$$\forall \tau'', e(k) \leq \tau'' < \tau_j : \breve{Q}(e(k), \tau'') < j - k + 1 \tag{11}$$

Equation 10 holds, because, according to Equation 8,

$$\breve{Q}(s, \tau_j) = \max(0, \lfloor \rho \cdot (\tau_j - e(k) - \Theta) \rfloor) \tag{12}$$

$$\breve{Q}(s, \tau_j) = \max(0, \lfloor \rho \cdot (e(k) + \Theta + \frac{j-k+1}{\rho} - e(k) - \Theta) \rfloor) \tag{13}$$

$$\breve{Q}(s, \tau_j) = \max(0, \lfloor \rho \cdot (\frac{j-k+1}{\rho}) \rfloor) \tag{14}$$

$$\breve{Q}(s, \tau_j) = \max(0, \lfloor j - k + 1 \rfloor) = j - k + 1 \tag{15}$$

And Equation 11 holds, because for $\tau'' = \tau_j - \epsilon, 0 < \epsilon \leq \tau_j - e(k)$, we have that

$$\breve{Q}(s, \tau'') = \max(0, \lfloor \rho \cdot (\tau_j - \epsilon - e(k) - \Theta) \rfloor) \tag{16}$$

$$\breve{Q}(s, \tau'') = \max(0, \lfloor \rho \cdot (e(k) + \Theta + \frac{j-k+1}{\rho} - \epsilon - e(k) - \Theta) \rfloor) \tag{17}$$

$$\breve{Q}(s, \tau'') = \max(0, \lfloor \rho \cdot (\frac{j-k+1}{\rho} - \epsilon) \rfloor) \tag{18}$$

$$\breve{Q}(s, \tau'') = \max(0, \lfloor j - k + 1 - \epsilon \rfloor) < j - k + 1 \tag{19}$$

Furthermore execution k is not guaranteed to have finished at any $\dot{\tau}$ for which holds

$$e(k) < \dot{\tau} < \tau_k = e(k) + \Theta + \frac{1}{\rho} \tag{20}$$

Equation 20 is true because for $\dot{\tau} = \tau_k - \epsilon, 0 < \epsilon < e(k)$ we have that

$$\breve{Q}(e(k), \tau_k - \epsilon) \geq \max(0, \lfloor \rho \cdot (e(k) + \Theta + \frac{1}{\rho} - \epsilon - e(k) - \Theta) \rfloor) \tag{21}$$

$$\breve{Q}(e(k), \tau_k - \epsilon) \geq \max(0, \lfloor \rho \cdot (\frac{1}{\rho} - \epsilon) \rfloor) \tag{22}$$

$$\breve{Q}(e(k), \tau_k - \epsilon) \geq \max(0, \lfloor 1 - \rho \cdot \epsilon \rfloor) = 0 \tag{23}$$

□

Lemma 2 *If the scheduler is a LR server with rate ρ, then bounding the external enabling time as in Equation 24 is equivalent to bounding the number of externally enabled executions as in Equation 6 in every busy period.*

Proof. Let execution k be the first execution in a busy period $(s, t]$ and let execution j occur in the same busy period, that is $e(k) = s$ and $j \geq k$ and $f(j) \leq t$, then the external enabling time of execution j is bounded by

$$e(j) \leq \phi_j = e(k) + \frac{j-k}{\rho} \tag{24}$$

According to Equation 24, execution j of actor v_x is guaranteed to be externally enabled at $\phi_j = e(k) + \frac{j-k}{\rho}$. For the bounds, as provided by Equation 24 and Equation 6, to be equal, it should hold that starting from ϕ_j the number external enablings is at least $j - k$. More formally, it should hold that

$$\forall \phi' \geq \phi_j : E(e(k), \phi') \geq j - k, \text{and} \tag{25}$$

$$\forall \phi'' < \phi_j : E(e(k), \phi'') < j - k \tag{26}$$

Equation 25 holds, because, according to Equation 6

$$E(e(k), \phi_j) = \rho \cdot (\phi_j - e(k)) \tag{27}$$

$$E(e(k), \phi_j) = \rho \cdot (e(k) + \frac{j-k}{\rho} - e(k) = j - k \tag{28}$$

And further for $\phi'' = \phi_j - \epsilon, 0 < \epsilon \leq \phi_j - e(k)$, we have that

$$E(e(k), \phi'') = \rho \cdot (\phi_j - \epsilon - e(k)) \tag{29}$$

$$E(e(k), \phi'') = \rho \cdot (e(k) + \frac{j-k}{\rho} - \epsilon - e(k) = j - k - \rho\epsilon < j - k \tag{30}$$

□

Lemma 3 *The bound on the finish time of execution j, τ_j, as defined by Equation 9 is equivalent to the bound $g(j)$ as defined by Equation 31.*

$$g(j) = \begin{cases} \max(e(j) + \Theta, g(j-1)) + \frac{1}{\rho} & \text{if } j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

Proof. We will first show the correctness for the first busy period by induction on the number of executions, and subsequently use this result as the base step to prove this lemma by induction on the number of busy periods.

Base step – induction on the number of executions: For the first execution we have that

$$g(1) = \max(e(1) + \Theta, g(1-1)) + \frac{1}{\rho} \tag{32}$$

and because $g(1-1) = g(0) = 0$.

$$g(1) = e(1) + \Theta + \frac{1}{\rho} = \tau_1 \tag{33}$$

Inductive step – induction on the number of executions: We assume that for execution $j, j \geq 1$ holds that $g(j) = \tau_j$, and thus that

$$e(k) + \Theta + \frac{j-k+1}{\rho} = \max(e(j) + \Theta, g(j-1)) + \frac{1}{\rho} \tag{34}$$

We will now show that given this assumption also for execution $j+1$ we have $g(j+1) = \tau_{j+1}$, and thus that

$$e(k) + \Theta + \frac{(j+1) - k + 1}{\rho} = \max(e(j+1) + \Theta, g(j)) + \frac{1}{\rho} \tag{35}$$

We know from Equation 31 and Equation 24 that

$$g(j+1) = \max(e(j+1) + \Theta, g(j)) + \frac{1}{\rho} \tag{36}$$

$$e(j+1) \leq \phi_{j+1} = e(k) + \frac{(j+1) - k}{\rho} \tag{37}$$

Given the induction hypothesis from Equation 34, we know that

$$g(j) = \tau_j = e(k) + \Theta + \frac{j - k + 1}{\rho} \tag{38}$$

We therefore conclude that

$$e(j+1) + \Theta \leq \phi_{j+1} + \Theta = e(k) + \Theta + \frac{j - k + 1}{\rho} = \tau_j = g(j) \tag{39}$$

This results in $\max(e(j+1) + \Theta, g(j)) = g(j)$, and therefore Equation 36 results in

$$g(j+1) = g(j) + \frac{1}{\rho} = \tau_{j+1} \tag{40}$$

We have now shown the equivalence of $g(j)$ and τ_j for the first busy period. This forms the base step for the proof by induction that this equivalence holds for all busy periods.

Inductive step – induction on the number of busy periods: We assume that $g(j) = \tau_j$ for all executions up to and including busy period β_l starting at $e(h)$, we now show that given this assumption $g(j) = \tau_j$ is also true for busy period β_{l+1} starting at $e(k)$.

Let k be the first execution in the busy period β_{l+1}, then execution $k-1$ belonged to the previous busy period β_l starting at $e(h)$. From Equation 24, we have that

$$e(k-1) \leq \phi_{k-1} = e(h) + \frac{(k-1) - h}{\rho} \tag{41}$$

And since execution k does not belong to busy period β_h, we have that

$$e(k) > e(h) + \frac{k - h}{\rho} \tag{42}$$

Furthermore we know from Equation 9 that

$$\tau_{k-1} = e(h) + \Theta + \frac{(k-1) - h + 1}{\rho} \tag{43}$$

$$\tau_{k-1} = e(h) + \Theta + \frac{k-h}{\rho} \tag{44}$$

Using Equation 42

$$\tau_{k-1} < e(k) + \Theta \tag{45}$$

Using the induction hypothesis we obtain $\tau_{k-1} = g(k-1)$, which leads to

$$g(k) = \max(e(k) + \Theta, g(k-1)) + \frac{1}{\rho} \tag{46}$$

$$g(k) = e(k) + \Theta + \frac{1}{\rho} = \tau_k \tag{47}$$

The earlier part of this proof, *Inductive step – induction on the number of executions*, that showed that $g(j+1) = \tau_{j+1}$, if $g(j) = \tau_j$, for the first busy period, did not assume that the executions j and $j+1$ were in the first busy period and therefore holds for any busy period. □

We will now present the most important result of this section.

Theorem 4 *The dataflow construct shown in Figure 3 models a task u_x with one input FIFO and one output FIFO that executes on a latency-rate server with latency Θ_x and allocated rate ρ_x.*

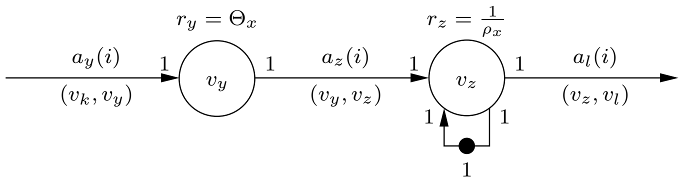


Figure 3: A dataflow component that models the effect of a LR server.

Proof. Let $a_y(i)$ be the arrival time of token i on edge (v_k, v_y), let $a_z(i)$ be the arrival time of token i on the edge (v_y, v_z), and let $a_l(i)$ be the arrival time of token i on the edge (v_z, v_l), see Figure 3(b).

From Figure 3(b) we can derive that

$$a_z(i) = a_y(i) + \Theta_x \tag{48}$$

$$a_l(i) = \max(a_z(i), a_l(i-1)) + \frac{1}{\rho_x} \tag{49}$$

Substition of Equation 48 in Equation 49 results in

$$a_l(i) = \max(a(i) + \Theta_x, a_l(i-1)) + \frac{1}{\rho_x} \tag{50}$$

Since every arrival of a token on the edge (v_k, v_y) leads to an external enabling of v_y, we have that $e(v_y, i) = a_y(i)$. And since every firing of v_z produces one token on the edge (v_z, v_l), we have that a finish of firing i of actor v_z corresponds with the release of a token on the edge (v_z, v_l): $f(v_z, i) = a_l(i)$. This means that Equation 50 becomes

$$f(v_z, i) = \max(e(v_y, i) + \Theta_x, f(v_z, i-1)) + \frac{1}{\rho_x} \tag{51}$$

By defining $f(v_z, i) = 0$ for $i \leq 0$, we have that substitution of $g(j) = f(v_z, i)$ and $e(j) = e(v_y, i)$ results in an equality of Equations 51 and 31. Lemma 3 and Lemma 1 tells us that the bound on the finish time of actor v_x as obtained by Equation 31 is equivalent to the bound on the number of finishes of actor v_x as obtained by Equation 8.

This means that in any busy period $(s, t]$ the number of releases by actor v_z in the interval $(s, \tau], s < \tau \leq t$ equals $\check{Q}$ as defined by Equation 8. □

8 Example

In this section we will show that TDM scheduling is a LR server, and that this insight leads to more a more accurate analysis then previously applied in [1].

8.1 TDM is a LR server

Let P be the TDM period, S_x be the time slice allocated to task u_x, $E_{x,i}$ be the execution time of execution i of task u_x, and $\overline{E}_x$ be the worst-case execution time of task u_x. We will derive in this section an expression for the latency and rate of a TDM scheduler in terms of the period, time-slice, and worst-case execution time, such that Equation 7 holds.

8.1.1 The rate of a TDM scheduler

The difference between subsequent task finishes is $f(u_x, i) - f(u_x, i-1) = E_{x,i}\frac{P}{S_x}$ in a busy period. This is because, in a busy period, without resource sharing execution i of task u_x will finish $E_{x,i}$ time later than the finish of firing $i-1$. However, with TDM scheduling we have that in every period P, there is only a time interval of length S_x time allocated to task u_x.

The guaranteed rate ρ_x at which task u_x finishes in a busy period with TDM scheduling is given by Equation 52. This is because $\forall i \in \mathbb{N} : f(u_x, i) - f(u_x, i-1) \leq \overline{E}_x\frac{P}{S_x}$, with $\overline{E}_x$ the worst case execution time of u_x. The guaranteed rate ρ_x is therefore $\frac{1}{\overline{E}_x\frac{P}{S_x}}$ which can be rewritten to obtain Equation 52.

$$\rho_x = \frac{1}{\overline{E}_x}\frac{S_x}{P} \tag{52}$$

8.1.2 The latency of a TDM scheduler

To derive the latency Θ_x we use a result from [1] to obtain that an exact upper bound on the response time of the first execution in a busy period is given by Equation 53.

$$\overline{r}_x = \overline{E}_x + (P - S_x) \left\lceil \frac{\overline{E}_x}{S_x} \right\rceil \tag{53}$$

We know that the response time of the first execution in a busy period is smaller than or equal to $\Theta_x + \frac{1}{\rho_x}$, and that this is the tightest bound. Therefore it should hold that

$$\Theta_x + \frac{1}{\rho_x} = \overline{E}_x + (P - S_x) \left\lceil \frac{\overline{E}_x}{S_x} \right\rceil \tag{54}$$

We will now rewrite $\frac{1}{\rho_x}$ to obtain an expression for the latency in terms of the period, time-slice and worst-case execution time. First we rewrite Equation 52 to obtain

$$\frac{1}{\rho_x} = P \frac{\overline{E}_x}{S_x} \tag{55}$$

which we can rewrite into

$$\frac{1}{\rho_x} = (P - S_x + S_x) \frac{\overline{E}_x}{S_x} \tag{56}$$

which also means that

$$\frac{1}{\rho_x} = (P - S_x) \frac{\overline{E}_x}{S_x} + S_x \frac{\overline{E}_x}{S_x} \tag{57}$$

which can be reduced to

$$\frac{1}{\rho_x} = (P - S_x) \frac{\overline{E}_x}{S_x} + \overline{E}_x \tag{58}$$

And since Equation 54 states that

$$\Theta_x = \overline{E}_x + (P - S_x) \left\lceil \frac{\overline{E}_x}{S_x} \right\rceil - \frac{1}{\rho_x} \tag{59}$$

which can be rewritten into

$$\Theta_x = \overline{E}_x + (P - S_x) \left\lceil \frac{\overline{E}_x}{S_x} \right\rceil - (P - S_x) \frac{\overline{E}_x}{S_x} - \overline{E}_x \tag{60}$$

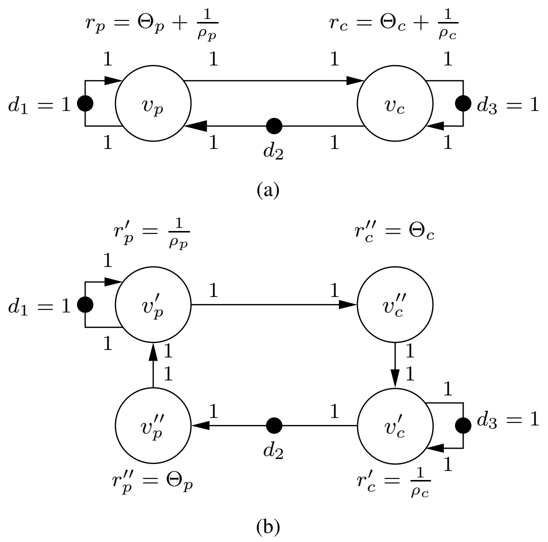
we obtain

$$\Theta_x = (P - S_x)(\left\lceil \frac{\overline{E}_x}{S_x} \right\rceil - \frac{\overline{E}_x}{S_x}) \tag{61}$$

The fact that expressions exist for the latency and rate show that TDM is an LR server.

8.2 Improved dataflow analysis results

Figure 4(a) shows how the effects of TDM arbitration are included with the approach from [1], while Figure 4(b) shows how the effects of TDM arbitration are included with the approach presented in this paper.



$r_p = \Theta_p + \frac{1}{\rho_p}$
$r_c = \Theta_c + \frac{1}{\rho_c}$
$d_1 = 1$
v_p
v_c
d_2
$d_3 = 1$
1
(a)
$r'_p = \frac{1}{\rho_p}$
$r''_c = \Theta_c$
v'_p
v''_c
v''_p
v'_c
$r''_p = \Theta_p$
$r'_c = \frac{1}{\rho_c}$
(b)

Figure 4: Inclusion of TDM arbitration according to [1] (a) and proposed inclusion of TDM arbitration (b)

The MCM of the SRDF graph in Figure 4(a) is

$$\mu_a = \max(\{\Theta_p + \frac{1}{\rho_p}, \frac{\Theta_p + \frac{1}{\rho_p} + \Theta_c + \frac{1}{\rho_c}}{d_2}, \Theta_c + \frac{1}{\rho_c}\}) \tag{62}$$

And the MCM of the SRDF graph in Figure 4(b) is

$$\mu_b = \max(\{\frac{1}{\rho_p}, \frac{\Theta_p + \frac{1}{\rho_p} + \Theta_c + \frac{1}{\rho_c}}{d_2}, \frac{1}{\rho_c}\}) \tag{63}$$

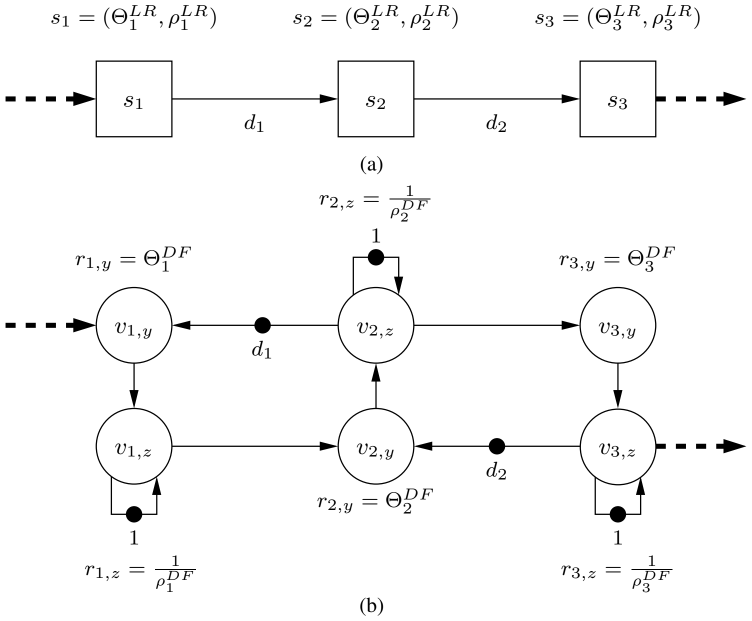
Since by definition Θ_p and Θ_c are non-negative, we have that $\mu_a \geq \mu_b$. The more accurate model can thus lead to a lower MCM, which corresponds to a higher throughput, and can therefore guarantee the satisfaction of more stringent throughput constraints for the same resource requirements in comparison with the model proposed in [1].

9 Improved buffer capacity bounds compared to LR analysis

In this section we will show an example in which an implementation where tasks only execute when there is output space available, as our applications are implemented, leads to smaller buffer capacities than an implementation in which tasks execute as soon as input data is available as the LR analysis model assumes.

Figure 5(a) shows the LR analysis model and SRDF model of a chain of three tasks that operate on a single stream. Let us assume that inputs arrive strictly periodic, e.g. from an Analog-to-Digital Converter, which is, for the sake of simplicity, left out of this example. In the LR analysis model such a stream is modelled with a (σ, ρ) model, where σ equals the maximum burst size and ρ equals the rate. Let us assume that $\sigma = 1$ and $\rho = 1$ accurately model the input stream. Let us further assume that the worst case execution time $\overline{E}_x$ of each task u_x is 1.

In the dataflow model as shown in Figure 5(b), we need to guarantee that the throughput of the graph equals the throughput of the input stream. Since $\rho = 1$, which means that one token arrives per time unit, the MCM of the graph needs to be maximally 1. If we further assume that $\Theta_i^{DF} = \rho_i^{DF} = 1$, for $i = 1, 2, 3$, then it can be verified that $d_1 = 4$ and $d_2 = 4$ are sufficient buffer capacities to let the SRDF graph have an MCM of 1.



$s_1 = (\Theta_1^{LR}, \rho_1^{LR})$
$s_2 = (\Theta_2^{LR}, \rho_2^{LR})$
$s_3 = (\Theta_3^{LR}, \rho_3^{LR})$
s_1
s_2
s_3
d_1
d_2
(a)
$r_{2,z} = \frac{1}{\rho_2^{DF}}$
1
$r_{1,y} = \Theta_1^{DF}$
$r_{3,y} = \Theta_3^{DF}$
$v_{1,y}$
$v_{2,z}$
$v_{3,y}$
d_1
$v_{1,z}$
$v_{2,y}$
$v_{3,z}$
d_2
$r_{2,y} = \Theta_2^{DF}$
1
1
$r_{1,z} = \frac{1}{\rho_1^{DF}}$
$r_{3,z} = \frac{1}{\rho_3^{DF}}$
(b)

Figure 5: A chain of tasks that operates on a stream, in which all tasks are scheduled on LR servers, the dashed arrows do not belong to the models, but show the input and output streams. (a) shows the LR analysis model (b) shows the SRDF model.

In the LR analysis model, the maximum number of tokens in the input queue of the k'th server equals $\sigma + \rho^{LR} \sum_{j=1}^{k} \Theta_j^{LR}$. This expression is derived using the bound $Q(s,t)$, which therefore means that $\Theta^{LR} = \Theta^{DF} + \frac{1}{\rho^{LR}}$, with $\rho_i^{LR} = \rho_i^{DF} \overline{E}_i$ this means that $\Theta_i^{LR} = 2$ for $i = 1, 2, 3$. The maximum number of tokens in the input queue of the server s_2 is therefore $d_1 = 1 + 1 * 4 = 5$ and the maximum number of tokens in the input queue of the server s_3 is therefore $d_2 = 1 + 1 * 6 = 7$. However, the LR analysis assumes that no space in the output buffer is required until the task has produced a token. While in some embedded streaming applications and architectures the input and output buffers can share the same memory range, and therefore re-use the memory space occupied by the input token to produce the resulting data in, this is not in general true. To deal with the general case, we need to increase each buffer capacity with one token. This results in $d_1 = 6$ and $d_2 = 8$.

From the expression for the maximum backlog it becomes clear that, in the LR analysis, the required buffer capacity scales with the length of the chain. This does not occur in the dataflow analysis. In the example the buffer capacities derived using LR analysis are larger than the buffer capacities as derived using dataflow analysis.

If one memory range can be used for both buffers, then this is modelled in the dataflow model as follows. From the model that is shown in Figure 5(b), we remove edges $(v_{2,z}, v_{1,y})$ and $(v_{3,z}, v_{2,y})$ and add an edge $(v_{3,z}, v_{1,y})$. One can verify that 6 tokens on edge $(v_{3,z}, v_{1,y})$ suffice to obtain an MCM of 1, which is smaller than the total required buffer capacity $d_1 + d_2 = 12$ that results from the LR analysis.

An important reason for the difference in buffer capacities is that our implementations include a local flow-control mechanism, because tasks only start when sufficient space is available in their output FIFOs, i.e. if a task rapidly produces a burst of containers it will eventually be slowed down because there will be no more empty containers left. LR analysis cannot model this behaviour, because this is a cyclic dependency between two tasks.

LR analysis cannot leverage the fact that traffic shapers might be part of the implementation, as e.g. a (σ, ρ)-regulator [3] or a rate- [19] or credit- [13] based scheduler. This is in contrast with Network Calculus [4] where application of traffic shapers can lead to reduced buffer capacities. In dataflow analysis traffic shapers are not required to reduce buffer capacities, but they might be required to enable the local analysis of priority based schedulers.

10 Hierarchical Dataflow Graphs

For various reasons, it is attractive to abstract parts of a dataflow graph, e.g. the dataflow graph shows confidential implementation details or the dataflow graph is too large or complex to enable efficient analysis. In this section we will show that abstraction of parts of the dataflow graph is possible while retaining the property that worst-case container arrival times can be determined through dataflow analysis techniques..

We define a block graph G_B that consists of blocks B and queues D, where B is a partitioning of the set of components C, and therefore the following holds

$$\forall B_x, B_y \in B \cdot B_x \neq B_y \Rightarrow B_x \cap B_y = \emptyset \wedge C = \bigcup_i B_i \tag{64}$$

Components are as defined in Section 5. The set of queues D is a subset of the set of queues in the component graph and formed by all queues in the component graph that connect blocks, and therefore the following holds for the set D

$$\forall (B_i, B_j) \in D \cdot B_i \in B \wedge B_j \in B \tag{65}$$

We define $I_x = \{(B_i, B_j) | (B_i, B_j) \in D \wedge B_j = B_x\}$ as the set of input edges of block B_x and $O_x = \{(B_i, B_j) | (B_i, B_j) \in D \wedge B_i = B_x\}$ as the set of output edges of block B_x.

In the following definition we assume two block graphs G_{B1} and G_{B2}. Let us define $a_{B1}(m, j)$ to be the arrival time of the j-th token in a queue m of G_{B1}, and $a_{B2}(m, j)$ to be the arrival time of the j-th token in a queue m of G_{B2}.

Definition 3 *If there is a one-to-one relation between blocks and queues in block graphs G_{B1} and G_{B2}, then we say that the token production and consumption behaviour of a block B_{x2} in block graph G_{B2} is conservative with respect to the token production and consumption behaviour of the corresponding block B_{x1} in block graph G_{B1}, if the following holds*

$$\forall m \in I_x, a_{B1}(m, j) \leq a_{B2}(m, j) \Rightarrow \forall n \in O_x, a_{B1}(n, j) \leq a_{B2}(n, j) \tag{66}$$

where the sets of input and output edges of blocks B_{x1} and B_{x2} are both called I_x and O_x to ease the notation. This is possible because of the one-to-one correspondence.

Theorem 5 *If every block in a block graph G_{B2} is conservative with respect to its corresponding block in block graph G_{B1}, i.e. Equation 5 holds for all blocks, then the worst-case arrival times of tokens in the output queues of every block B_{x1} can be computed with dataflow analysis techniques applied on block graph G_{B2}.*

Proof. This immediately follows from the fact that all blocks in G_{B2} are conservative with respect to their corresponding blocks in G_{B1}, the existense of the one-to-one relation between both block graphs, and the fact that every SRDF graph is monotonic. $\square$

Theorem 6 *If every block in a block graph G_{B2} is conservative with respect to its corresponding block in block graph G_{B1}, i.e. Equation 5 holds for all blocks, then the worst-case arrival times of containers in the FIFO buffers that correspond with the pairs of queues in the block graph G_{B1} can be computed with dataflow analysis techniques that are applied on block graph G_{B2}.*

Proof. This follows from the one-to-one relation between FIFO buffers in the implementation and pairs of queues in the component graph, the fact that the set of queues in a block graph is a subset of the queues in the component graph, and Theorems 3 and 5. □

In the next section we will show how to construct a block graph that abstracts a block from the original block graph.

11 Construction of Hierarchical Dataflow Graphs

In this section we will conservatively model a block B_x that contains an SRDF subgraph and has one input and one output queue with a block B_y. In later sections we will generalise this.

For an SRDF (sub)graph G_S a strictly periodic schedule with a period T can be constructed where T equals the MCM of G_S [15]. In such a strictly periodic schedule the start time of actor v_x in the k'th iteration is given by

$$s(v_x, k) = s(v_x, 0) + kT \tag{67}$$

If actor $v_a \in B_x$ consumes tokens from the input queue, then the $k + 1$'th token, with $k \geq 0$, is consumed at $s(v_a, k) = s(v_a, 0) + kT$. If actor $v_b \in B_x$ produces tokens on the output queue, then the $k + 1$'th token is produced at $s(v_b, k) + r(v_b) = s(v_b, 0) + kT + r(v_b)$. We restrict ourselves to blocks in which there is no iteration difference between actor v_a and actor v_b, i.e. there is a path from v_a to v_b without initial tokens.

The difference L_{ab} between the token production time and the token consumption time in any iteration therefore equals $L_{ab} = s(v_b, 0) - s(v_a, 0) + r(v_b)$, while the difference between subsequent token production times equals T.

With $c(i)$ the token consumption time in iteration i by v_a and $p(i)$ the token production time in iteration i by v_b, we have that

$$p(i) = \max(c(i) + L_{ab}, p(i - 1) + T) \tag{68}$$

Theorem 7 *If $L_{ab} \geq 0$, then the dataflow component as shown in Figure 6 models Equation 68.*

Proof. In the dataflow component that is shown in Figure 6, token $i + 1$, with $i \geq 0$, is produced on the output edge (v_z, v_l) at $a_{zl}(i + 1)$ which equals the finish time of actor v_z in iteration i, which is $s(v_z, i) + 0 = s(v_z, i)$. The start time of v_z in iteration i is given by

$$s(v_z, i) = \max(a_{xz}(i - 1), a_{yz}(i)) \tag{69}$$

which equals

$$s(v_z, i) = \max(a_{zx}(i - 1) + T, a_{yz}(i)) \tag{70}$$

which equals

$$s(v_z, i) = \max(s(v_z, i - 1) + T, s(v_y, i) + L_{ab}) \tag{71}$$

which equals

$$s(v_z, i) = \max(s(v_z, i - 1) + T, s(v_y, i) + L_{ab}) \tag{72}$$

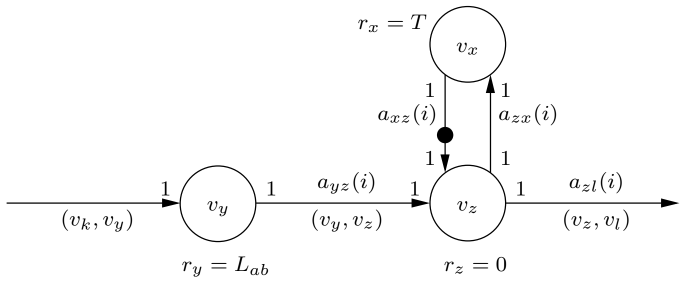


Figure 6: A dataflow component that conservatively models a block.

therefore

$$a_{zl}(i) = s(v_z, i) = \max(s(v_y, i) + L_{ab}, s(v_z, i-1) + T,) \tag{73}$$

By letting $p(i)$ equal $a_{zl}(i)$ and $c(i)$ equal $s(v_y, i)$ we have shown equivalence. $\square$

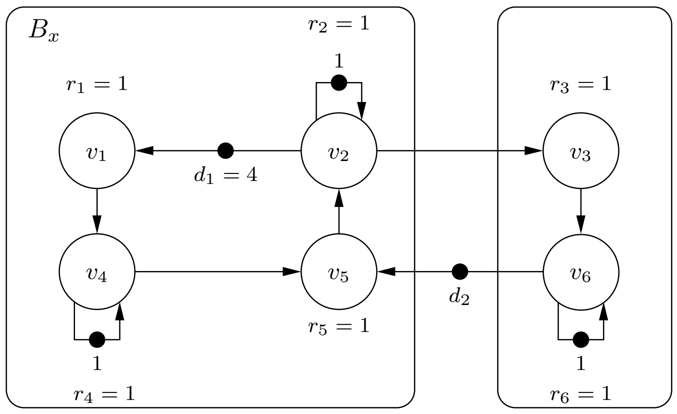
Intuitively one might have thought that since the token production time is determined by a latency term and a throughput term, also in this case the dataflow component as shown in Figure 3 could have been applied with $r_z = T$ and $r_y = L_{ab} - T$ for actors v_z and v_z from Figure 3. However in that case it is required that $L_{ab} \geq T$ in order to have a non-negative actor response time, which is in general not true.

11.1 Example

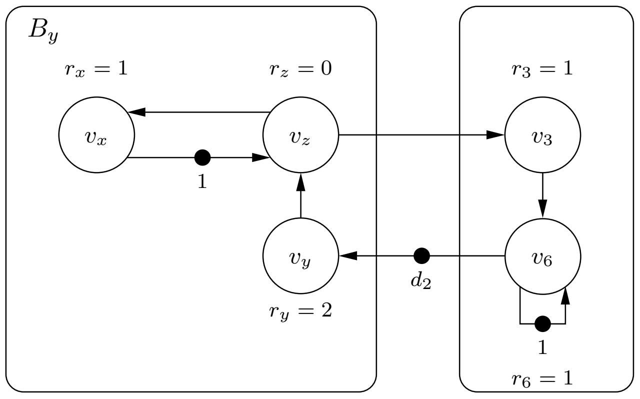
Block B_y as shown in Figure 7(b) conservatively models block B_x as shown in Figure 7(a). This is because with 4 tokens on the cycle v_1, v_4, v_5, v_2 the MCM of block B_x is 1. And further in a strictly periodic schedule with $T = 1$ every actor starts immediately when it is enabled. Therefore, the difference in start times between actors v_2 and v_5 is 1, which results in $L_{52} = 1 + 1 = 2$. Using the dataflow graph as shown in Figure 7(b) we can derive that 4 tokens on the cycle v_z, v_3, v_6, v_y are sufficient to obtain an MCM of 1. One can verify in Figure 7(a) that this is indeed sufficient.

12 Generalisation

In this section we will generalise the results obtained until now to include tasks with multiple input and output FIFO buffers and with less constrained container consumption and production behaviour than the behaviour as defined in Section 4. We will generalise three results, (1) an SRDF graph is temporally monotonic, (2) a task with one input and output FIFO buffer that consumes and produces one container per execution and is executed on an LR server can be modeled with a dataflow component, and (3) parts of an SRDF graph can be abstracted.



(a)



(b)

Figure 7: An example of abstraction, in which block B_y abstracts block B_x.

12.1 Monotonicity

We can generalise the result from Theorem 2 to conclude that deterministic Dynamic Dataflow (DDF) graphs as defined in [11] are temporally monotonic. The start time of a firing of a deterministic DDF actor depends on the token production times on specific edges, and therefore on finishes of specific firings of specific DDF actors. For every firing of a DDF actor v_x this can be formulated like Equations 3 and 4, where however the set of actors that determines the enabling time of v_x can change from firing to firing. The proof of Theorem 2 does not depend on a fixed set of actors that determines the enabling time and is therefore also applicable to deterministic DDF graphs. This means that deterministic DDF graphs are temporally monotonic.

Also for a deterministic DDF graph we can define a component graph that partitions the deterministic DDF graph. If there is a one-to-one relation between these components and the task graph, and each component is conservative with respect to the corresponding task, then since deterministic DDF graphs are temporally monotonic we have that token arrival times in the deterministic DDF graph are worst-case container arrival times in the implementation.

12.2 Execution on LR servers

The result of Theorem 4 can be generalised to include any task that can be modelled with a deterministic DDF actor. In Figure 3, $a_y(i)$ corresponded with the external enabling of the i'th execution of a task u_x on an LR server. And $a_l(i)$ was a conservative finish time of the i'th execution of a task u_x on an LR server. If a task starts when a specific number of tokens is available on a specific subset of input FIFO buffers, which only depends on the tasks state, and produces a certain number of containers on a certain set of output FIFO buffers, then this can be modelled with deterministic dataflow actors that have firing rules and production rules. The enabling rule can be modelled by an actor v_k that produces a token on edge (v_k, v_l) as soon as the firing rule is satisfied, excluding the self-cycle. The production rule can be modelled by an actor v_l that produces tokens on various edges as soon as a token is produced on edge (v_z, v_l). Since both actor v_k and v_l have a response time that equals zero, it is equivalent to let v_y execute its i'th firing as soon as the firing rule of the i'th firing is satisfied, and let firing i of actor v_z produce tokens on various edges according to the production rates of firing i.

Also the worst case execution time of different firings of a dataflow actor can be different. Since with our definitions Θ_x and ρ_x are dependent on the worst case execution time, this can be modelled by parameterising Θ_x and ρ_x, resulting in $\Theta_x(i)$ and $\rho_x(i)$ in order to make them dependent on the worst case execution time of firing i.

12.3 Hierarchy

A straightforward extension to include blocks that consume from and produce on multiple edges is to model the relation between each pair of input and output actors seperately with the dataflow component as shown in Figure 6. In this way any SRDF (sub)graph can be abstracted. Even though the number of required dataflow components can be quadratic in the number of input and output actors of the block, we do not expect this too be problematic. This is based on the task graph topologies that we have encountered until now, in which the number of different paths is limited.

Also for Multi-Rate Dataflow [10] (MRDF) it is possible to create a strictly periodic schedule as done in [18]. From this strictly periodic schedule it is again straightforward to derive the difference in start times between input and output actors of a block and the rate at which the output actor fires. Currently, in a document, which is under interval review, we create a strictly periodic schedule of a Cyclo-Static Dataflow [2] graph. It seems probable that a conservative strictly periodic schedule can be constructed for dataflow graphs of which upper bounds on the response times, and upper and lower bounds on the token production and consumption rates – and therefore also upper bounds on the firing rates – are known.

References

[1] M. Bekooij *et al.* *Dataflow Analysis for Real-Time Embedded Multiprocessor System Design*, chapter 15. Dynamic and Robust Streaming Between Connected CE Devices. Kluwer Academic Publishers, 2005.

[2] G. Bilsen *et al.* Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.

[3] R. L. Cruz. A calculus for network delay, part i: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

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

[5] F. Bacelli, G. Cohen, G.J. Olsder, and J-P. Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley, 1992.

[6] M. Garey and D. Johnson. *Computers and intractability*. W.H. Freeman and company, 1979.

[7] S. Goddard and K. Jeffay. Managing latency and buffer requirements in processing graph chains. *The Computer Journal*, 44(6), 2001.

[8] M. Jersak *et al.* Performance analysis of complex embedded systems. *International Journal of Embedded Systems*, 1(1-2):33–49, 2005.

[9] E. Lee and S. Ha. Scheduling strategies for multi-processor real-time dsp. In *Proceedings of IEEE Global Telecommunications Conference and Exhibition*, volume 2, pages 1279–1283, Dallas, TX, USA, November 1989.

[10] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[11] M. Bekooij, S. Parmar, and J. van Meerbergen. Performance guarantees by simulation of process networks. In *9th International Workshop on Software and Compilers for Embedded Systems (SCOPES'05)*, Sept. 2005.

[12] A. Maxiaguine *et al.* Tuning soc platforms for multimedia processing: Identifying limits and tradeoffs. In *Proceedings of CODES+ISSS'04*, pages 128–133, Stockholm, September 2004.

[13] C. Otero Pérez *et al.* *Dynamic and Robust Streaming between Connected CE Devices*, chapter Resource Reservations in Shared Memory Multiprocessor SoCs. Kluwer Academic Publishers, 2005.

[14] P. Poplavko *et al.* Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *Proceedings of CASES'03*, November 2003.

[15] R. Reiter. Scheduling parallel computations. *Journal of the ACM*, 15(4):590–599, October 1968.

[16] S. Sriram and S.S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker Inc., 2000.

[17] D. Stiliadis and A. Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.

[18] M. Wiggers *et al.* Efficient computation of buffer capacities for multi-rate real-time systems with back-pressure. In *Proceedings of CODES+ISSS'06*, October 2006.

[19] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal of High-Speed Networks*, 3(4), 1994.