Buffer Capacity Computation for Throughput-Constrained Modal Task Graphs

17

MAARTEN H. WIGGERS
University of Twente
MARCO J. G. BEKOOIJ
NXP Semiconductors
and
GERARD J. M. SMIT
University of Twente

Increasingly, stream-processing applications include complex control structures to better adapt to changing conditions in their environment. This adaptivity often results in task execution rates that are dependent on the processed stream. Current approaches to compute buffer capacities that are sufficient to satisfy a throughput constraint have limited applicability in case of data-dependent task execution rates.

In this article, we present a dataflow model that allows tasks to have loops with an unbounded number of iterations. For instances of this dataflow model, we present efficient checks on their validity. Furthermore, we present an efficient algorithm to compute buffer capacities that are sufficient to satisfy a throughput constraint.

This allows to guarantee satisfaction of a throughput constraint over different modes of a stream processing application, such as the synchronization and synchronized modes of a digital radio receiver.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-based Systems**]: Real-Time and Embedded Systems

General Terms: Algorithms, Languages, Performance

Additional Key Words and Phrases: Dataflow, data-dependent inter-task synchronization, multiprocessor

ACM Reference Format:

Wiggers, M.H., Bekooij, M.J.G., and Smit, G.J.M. 2010. Buffer capacity computation for throughput-constrained modal task graphs. ACM Trans. Embedd. Comput. Syst. 10, 2, Article 17 (December 2010), 59 pages.
DOI = 10.1145/1880050.1880053 http://doi.acm.org/10.1145/1880050.1880053

M. H. Wiggers is currently affiliated with Eindhoven University of Technology, Eindhoven, The Netherlands.
Authors' address: M. H. Wiggers, Eindhoven University of Technology, P.O. Box 513, 5600MB, Eindhoven, The Netherlands; email: m.h.wiggers@tue.nl.

DOI 10.1145/1880050.1880053 http://doi.acm.org/10.1145/1880050.1880053

1. INTRODUCTION

Stream-processing applications have an increasingly complex control structure to better adapt to changing conditions in the environment (for example, entropy encoding in audio/video processing and mode switching in digital radio). Often, these applications require a multiprocessor implementation for performance and power dissipation reasons. On such a multiprocessor system, these stream processing applications are typically implemented as task graphs, where tasks communicate data over fixed-capacity first-in first-out (FIFO) buffers. For stream-processing applications, we often see firm real-time performance requirements, where throughput constraints dominate over latency constraints. A firm real-time requirement means that a deadline miss results in a severe loss of quality but that no safety issues are involved. Loss of synchronization in case of digital radio reception and clicks in an audio-playback application are typically perceived as dramatic decreases in quality by the end-user.

In a task graph, tasks communicate data over FIFO buffers. On every FIFO buffer, tasks synchronize on containers, which are place-holders for data and have a fixed size. Tasks repeatedly execute a sequence of loops. Applications that adapt to their environment can include tasks of which the number of containers consumed or produced in a loop iteration or the number of loop iterations depends on the actually processed data stream. Because, on every buffer, tasks synchronize on sufficient space and data, buffer capacities determine deadlock-freedom and influence the throughput of the application.

Currently, buffer capacities can be determined for a task graph as shown in Figure 1, using variable-rate dataflow (VRDF) from Wiggers et al. [2008]. In Figure 1, the expressions at the end points of the buffers denote the consumption and production quanta, while the expression above the tasks denotes the response time. For this task graph, buffer capacities should be determined such that task w_τ can execute strictly periodically with period τ. In this task graph, task w_i has a loop with n iterations that each read one container from buffer $b_{\tau i}$. After the loop, a container is written on buffer b_{ij}. The requirement in Wiggers et al. [2008] is that task w_i waits on n containers on buffer $b_{\tau i}$ and on one container on buffer b_{ij} before it starts the loop. However, the number of loop iterations is not always known; for instance, if task w_i is a variable-length decoder, then the number of iterations depends on the processed stream and is only determined during execution of the loop. Furthermore, the required buffer capacity on buffer $b_{\tau i}$ grows with the maximum value of n, and the buffer capacity is unbounded if n is unbounded.

In this article, we present a novel dataflow model called variable-rate phased dataflow (VPDF) that has actors with phases of execution, where these phases can fire a variable number of times. This implies that we can now distinguish the individual iterations of a loop, for example, of task w_i, as shown in Figure 2. In the task graph of Figure 2, task w_i reads n times a single container and can exit the loop based on the data in this container (i.e., the value of n does not need to be known before the loop is started). Furthermore, the capacity of buffer $b_{\tau i}$ is independent of n (i.e., a bounded buffer capacity exists even if n is unbounded).

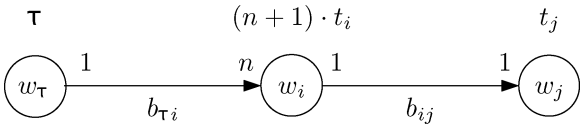


Fig. 1. Task graph that can be modeled by variable-rate dataflow.

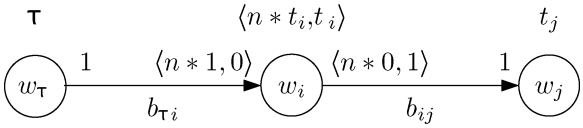


Fig. 2. Task graph that can be modeled by variable-rate phased dataflow.

More precisely, VPDF actors have a sequence of phases, and for every phase, the number of firings is parameterized. For every phase, parameterized token transfer quanta are specified for each adjacent queue. The distinction between a parameter that specifies the number of firings of a phase and parameters that specify token transfer quanta is an important novel aspect of VPDF. This distinction is required to model loops for which no upper bound on their number of iterations is known. The phases of cyclo-static dataflow actors [Bilsen et al. 1996] can, in principle, be unrolled to obtain a larger dataflow graph without phases. This is, however, not possible in any similar manner for VPDF actors because dependent on the parameter values a token is transferred by a different firing, which can be of a different phase. In other words, there is no similar unrolling possible because the dependencies between firings, and also between phases, are not fixed. Parameters of VPDF actors can attain a new value once in every iteration through all phases of the actor. In case the number of firings of a phase is parameterized in a parameter for which there is no upper bound known on the value that it can attain, then we say that this phase has an unbounded number of firings and this phase can be seen as a mode of the actor. An additional benefit of the introduction of phases is that the worst-case time between consumption and production of containers in Figure 2 is t_i instead of $(n + 1) \cdot t_i$, which enables the derivation of smaller-buffer capacities and end-to-end latency.

The VPDF dataflow model as presented in this article is a generalization of VRDF. This implies that also VPDF can model conditional execution of tasks. An example task graph is shown in Figure 3, where task w_i first produces the value of p before it produces p times a container. Task w_k first consumes the value of p before it consumes p containers. Since we allow p to have the value zero, this leads to conditional execution of task w_j.

The VPDF dataflow model is defined such that for every task graph that can be modeled by a valid VPDF dataflow graph, buffer capacities can be computed that satisfy a throughput constraint and any constraints on maximum buffer capacities. This algorithm is a generalization of the algorithm for VRDF, and has as important contribution the concept of an aggregate firing. The concept

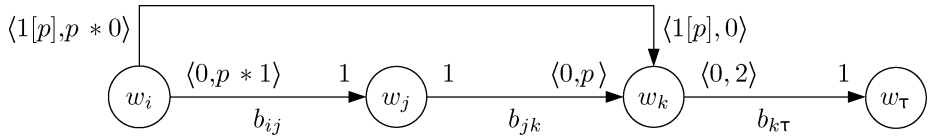


Fig. 3. Task graph with conditional execution of task w_j, because p can attain value zero.

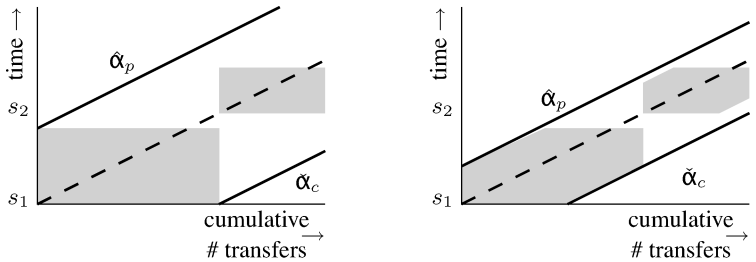


Fig. 4. Aggregate firing shapes. Schedules with two aggregate firing shapes that start at s_1 and s_2, respectively. The start times follow from the number of tokens transferred by previous aggregate firings and the allowed time per transferred token (i.e., the slope of the dashed line). The rectangular firing shapes result in a larger difference between the upper bound on production times, $\hat{\alpha}_p$, and the lower bound on consumption times, $\check{\alpha}_c$, of this actor, and will result in larger buffer capacities.

of an aggregate firing introduces a third level of scheduling. For VRDF graphs, for each individual actor, a schedule of firings was determined that satisfies the throughput constrained. Subsequently, for each actor, the start time of the first firing was determined such that on each queue tokens arrive before they are consumed in the constructed schedules. For VPDF graphs, firings are first scheduled per iteration through all phases to form an aggregate firing. Then, these aggregate firings perform the role that firings have in the algorithm for VRDF graphs.

Figure 4 shows schedules of aggregate firings together with a linear upper bound on token production times $\hat{\alpha}_p$ and a linear lower bound on token consumption times $\check{\alpha}_c$. These bounds are used to compute buffer capacities. Schedules of aggregate firings are constructed such that the left bottom corner of the aggregate firing shape is on the dashed line. As a result, the schedule of aggregate firing shapes remains between the linear bounds. The aggregate firing shapes bound the production and consumption times of the actor firings within the aggregate firings. In Section 4, we introduce aggregate firings that have a rectangular shape as in Figure 4(a). However, the size of the rectangular firing shape grows as the number of firings per iteration through all phases grows. Therefore, with rectangular aggregate firing shapes, a growing number of firings results in a growing difference between these bounds, which in turn results in growing computed buffer capacities. This means that these aggregate firings are not suitable in case of an unbounded number of firings per phase. Therefore, in Section 5, we introduce aggregate firings that have an aggregate firing shape that grows within the linear bounds along the required transfer

rate, as the number of firings increases. Because the linear bounds are not affected by an increase in firings, the computed buffer capacity does not increase with a growing number of firings.

In the next section, we discuss related work. Subsequently, in Section 3, we introduce the task graph, which is input to the buffer capacity problem, the dataflow model on which the actual computation takes place, and their relation. Section 4 presents the algorithm to compute buffer capacities and shows that these buffer capacities are indeed sufficient. While we restrict ourselves to parameters with an upper bound in Section 4, Section 5 extends this approach in order to include parameters with no upper bound. Further, we compute buffer capacities for an example application with mode switches in Section 6. We conclude and present directions for future work in Section 7.

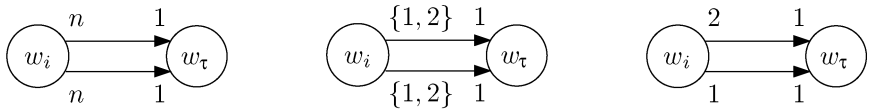
2. RELATED WORK

Related work can be split up in work that applies quasi static-order scheduling and work that applies runtime arbitration.

Approaches that construct quasi static-order schedules [Bhattacharya and Bhattacharyya 2001; Buck 1993; Girault et al. 1999; Neuendorffer and Lee 2004] require the existence of a bounded length schedule for the (sub)graph. This requires that changes in production and consumption quanta only occur every (sub)graph iteration. This is a (global) requirement on the graph. However, for instance, a variable-length decoder changes its consumption quantum dependent on the processed data stream and independent of other tasks production and consumption quanta, that is, it makes a local decision on its consumption quanta, which is independent of graph iterations.

The approach presented in Sen et al. [2005] proposes to have tasks produce and consume a constant number of data structures, where these data structures have a variable size. This approach is not applicable for the task graph of Figure 2. This is because this approach requires that a data structure of size n is produced by task w_τ, while w_τ might not know n and there might not be an upper bound for n.

Runtime arbitration is applied in a class of approaches that characterize traffic [Haid and Thiele 2007; Jersak et al. 2005;Maxiaguine et al. 2004]. These approaches characterize the traffic generated by a task and derive buffer capacities and end-to-end latency given these traffic characterizations. Suppose in the task graph of Figure 2 that task w_j is required to execute strictly periodically instead of w_τ and that n has an upper bound. In this case, the traffic of w_τ can no longer be characterized independently. This can be seen by the fact that if w_τ produces at the maximum consumption rate of w_i, then a buffer of unbounded capacity is required for lower rates in order not to lose data. If task w_τ produces data at a lower rate than the maximum consumption rate, then data will not always arrive in time in the buffer to satisfy the throughput constraint. Our approach can take into account that start times of task w_τ will be delayed dependent on the consumption rate of w_i, because task w_τ will only start as soon as there is an empty container available on buffer $b_{\tau i}$. Furthermore, these approaches do not allow to specify a coupling between the number



(a) task graph, $n = \{1, 2\}$ (b) interval specification (c) inconsistent graph

Fig. 5. With multiple paths between two tasks, the existence of bounded buffer capacities requires a coupling between variation in transfer quanta on these paths.

of containers transferred by a task on various buffers. For the task graph, as shown in Figure 5(a), it is clear that, for deadlock-free execution, a buffer capacity of two containers is sufficient on both buffers. However, if only intervals are specified per buffer, as shown in Figure 5(b), then the specification allows for an unbounded number of executions of the situation shown in Figure 5(c). Such a sequence of executions results in an unbounded accumulation of containers on the top buffer, and requires an unbounded buffer capacity for this buffer.

Cyclo-dynamic dataflow [Wauters et al. 1996] and bounded dynamic dataflow [Pankert et al. 1994] also apply runtime arbitration to allow for data-dependent execution rates, but do not provide an approach to calculate buffer capacities that guarantee satisfaction of a throughput constraint.

Another aspect that is different from related work is the following. We allow parameters to attain the value zero, which models conditional execution of tasks. Existing work that allows conditional execution of tasks, however, has its drawbacks. For boolean dataflow [Buck 1993] graphs, we know that a consistent graph can still require unbounded memory. For well-behaved dataflow [Gao et al. 1992], we know that any graph constructed using the presented rules only requires bounded memory. However, no procedure is given that decides whether any given—boolean—dataflow graph is a well-behaved dataflow graph.

In contrast to existing work, we present a simple decision procedure to check whether any given task graph is a valid input for the algorithm that computes sufficient buffer capacities that satisfy a throughput constraint.

VPDF is a generalization of both VRDF [Wiggers et al. 2008] and of cyclo-static dataflow [Bilsen et al. 1996]. The presented algorithm to compute buffer capacities that satisfy a throughput constraint is a generalization of the algorithms presented in Wiggers et al. [2008, 2007a] for VRDF and cyclo-static dataflow, respectively. Alternative approaches exist to derive buffer capacities that satisfy a throughput constraint for cyclo-static dataflow [Stuijk et al. 2008]. However, while our algorithm for VPDF has a polynomial complexity when applied to cyclo-static dataflow graphs, these alternative approaches are specific to cyclo-static dataflow and have an exponential complexity.

3. GRAPH DEFINITION

In this section, we first define a task graph and dataflow graph that only allow parameters that are local to tasks and do not support communication of parameter values between tasks. While buffer capacities are determined for the task graph, the actual computation of these capacities takes place on the corresponding dataflow graph. The main contribution of this article is in the definition and analysis of the dataflow graph, and can be understood independently of the task

graph. A discussion of the differences between task and dataflow graphs can be found at the end of Section 3.4.

3.1 Task Graph

We assume that an application is implemented as a task graph. A task graph is a weakly connected directed multigraph $T = (W, B, P_T, \zeta, \eta, \kappa, \phi_T, \xi, \lambda, \theta_T, \chi_T)$. A weakly connected directed graph is a graph for which the underlying undirected graph is connected. A directed graph is a directed multigraph, if there are edges that have the same source and destination, that is, if there are at least two edges that cannot be distinguished based on their source and destination vertex. Therefore, in case of a multigraph, we distinguish between edges based on their label. In such a task graph, we have that tasks w_a and w_b, with $w_a, w_b \in W$, can communicate over a FIFO buffer $b_{ab} \in B$. Let b_{ab} denote a buffer over which task w_a sends data to task w_b. We say that tasks consume and produce containers on these buffers, where a container is a placeholder for data and all containers in a buffer have a fixed size. The capacity of a buffer b is given by $\zeta(b)$, with $\zeta : B \rightarrow \mathbb{N}$, while the number of initially filled containers is given by $\eta(b)$, with $\eta : B \rightarrow \mathbb{N}$.

We let $\mathbb{N}$ denote the set of nonnegative integer values, $\mathbb{N}^*$ denote the set of positive integer values, and we let $\mathcal{P}_f(\mathbb{N})$ denote the set of all subsets of $\mathbb{N}$ excluding the empty subset.

Tasks consist of a finite sequence of loops. Loops have a number of iterations. In every iteration of the loop, a constant number of synchronization actions is executed (i.e. there is no conditional execution of read and write calls within an iteration). The number of containers produced or consumed by an execution of a synchronization action is parameterized, but determined before every execution of the loop. A synchronization action blocks execution of the task until the required number of containers is present in the required buffer.

The code of a task is statically partitioned in code-segments, which are sequences of sequentially executed program statements. Code-segments start at every convergence and divergence of control flow and at every synchronization action. A nonblocking code-segment is a sequence of code-segments. Every synchronization action starts a nonblocking code-segment. Nonblocking code-segments can only start at synchronization actions and continue until just before the next executed synchronization action. Since the sequence of synchronization actions is data-dependent, also the sequence of code-segments that form a nonblocking code-segment is data-dependent. For every nonblocking code-segment, it is, however, possible to enumerate all possible sequences of code-segments. This is because the number of loops is finite; therefore, also the number of different subsequent synchronisation actions is finite. Code-fragments of an example task are shown in Listing 1. In this example, there are six code-segments and three nonblocking code-segments.

There is a one-to-one correspondence between synchronization actions and nonblocking code-segments. The number of nonblocking code-segments of task w_a is $\theta_T(w_a)$, with $\theta_T : W \rightarrow \mathbb{N}^*$. The number of times nonblocking code-segment $c \in \mathbb{N}^*$ of task w_a is executed is parameterized and given by $\chi_T(w_a, c)$. A parameter specifies the number of times that the synchronization action and

```
1  while(1)
2  {
3    int a = read(1,b1);
4    int bound = f(a);                          I
5    for(int i = 0; i < bound; i++)
6    {                                          II
7      ...
8      write(3,b2);
9      ...                                      III
10   }
11   int quantum = g(a);                        IV
12   while(...)
13   {                                          V
14     ...
15     write(quantum,b2);
16     ...                                      VI
17   }
18 }
```

Listing 1. A task w_i with six code-segments and three nonblocking code-segments formed by sequences of code-segments.

thereby its corresponding nonblocking code-segment are executed. The set of parameters is given by P_T. We define $\chi_T : W \times \mathbb{N}^* \rightarrow P_T$.

The number of transferred containers per execution of a nonblocking code-segment c is parameterized. If a nonblocking code-segment c of a task w_b starts with a read from buffer $b_{ab} \in B$, then the number of full containers that c requires to start is parameterized and given by $\lambda(b_{ab}, c)$, which equals the number of empty containers that are produced by c. We define $\lambda : B \times \mathbb{N}^* \rightarrow P_T$, which is the function that returns the parameterized container consumption quantum on a particular buffer for a particular nonblocking code-segment. Similarly, if code-segment c' of a task w_a starts with a write on buffer b_{ab}, then the number of empty containers that are required on this buffer in order to start c', is given by $\xi(b_{ab}, c')$, with $\xi : B \times \mathbb{N}^* \rightarrow P_T$. This equals the number of full containers that c' produces on b_{ab}. With each parameter $p \in P_T$, a set of integer values is associated by $\phi_T(p)$, where we define $\phi_T : P_T \rightarrow \mathcal{P}_f(\mathbb{N})$.

Of a task w_a, the number of iterations of different loops and the number of transferred containers of different synchronization actions are all parameterized in different parameters.

The worst-case response time of a nonblocking code-segment c is defined as the maximum difference between the time at which sufficient containers are present to enable the execution of c and the time at which this execution finishes. Note that this is the maximum over all possible sequences of code-segments of this nonblocking code-segment. The worst-case response time of nonblocking code-segment c of task w_a is denoted by $\kappa(w_a, c)$, with $\kappa : W \times \mathbb{N}^* \rightarrow \mathbb{R}^+$. As in Wiggers et al. [2007], we allow tasks to be scheduled at runtime by arbiters that can guarantee a worst-case response time given the worst-case execution times and the scheduler settings (i.e., the guarantee is independent

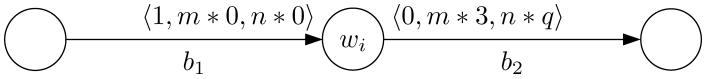


Fig. 6. A task graph with task w_i from Listing 1.

of the rate with which tasks start their execution). This class of schedulers is called Latency-Rate Servers [Stiliadis and Varma 1998; Wiggers et al. 2007b] and, for instance, includes time-division multiplex and round-robin.

Task w_i of Figure 6 corresponds with the code-fragment of Listing 1. In the first nonblocking code-segment, one container is read from buffer b_1, and no container is written on buffer b_2. The second nonblocking code-segment executes m times and each execution writes three containers on buffer b_2. The third nonblocking code-segment executes n times and each execution writes q containers on buffer b_2. In this example, the parameters m and q are required to be associated with a finite set of integer values, while the value of n can be unbounded.

3.2 Constraints

Next to the task graph, a throughput constraint and constraints on maximum buffer capacities are input to the buffer capacity computation procedure of this work.

The throughput constraint is given by specifying a task w_τ, which is a task that is required to execute wait-free and either does not have any input buffers, a source, or does not have any output buffers, a sink. A task executes wait-free, if every nonblocking code-segment of this task is enabled no later than the worst-case finish time of its previous nonblocking code-segment. Often, a sink or source of the task graph is periodically triggered by a clock, and immediately subsequent to this trigger, it produces or consumes one container. Even though tasks are enabled by the arrival of a predetermined number of containers, and not time-triggered, we can consider a time-triggered sink or source as a task if the throughput constraint is put on this sink or source. This is because satisfaction of this throughput constraint implies that we guarantee that every periodic activation the required number of containers is present, which implies that we cannot distinguish between an event- or time-based triggering of task w_τ.

Constraints on maximum buffer capacities are given by the function $\hat{\zeta} : B \rightarrow \mathbb{N}^* \cup \infty$, and constraints on the maximum number of initially filled containers on a buffer are given by the function $\hat{\eta} : B \rightarrow \mathbb{N}^* \cup \infty$. The specification of a maximum buffer capacity or maximum number of initially filled containers equal to infinity implies that there is no constraint.

3.3 Variable-Rate Phased Dataflow

A VPDF graph G is a directed multigraph given by the tuple $G = (V, E, P_D, \delta, \rho, \phi_D, \pi, \gamma, \theta_D, \chi_D)$ and consists of a finite set of actors V and a finite set of labeled queues E.

Each actor has a finite sequence of phases. The number of phases of actor v_i is given by $\theta_D(v_i)$, with $\theta_D : V \rightarrow \mathbb{N}^*$. The number of times phase h of actor v_i

is fired is parameterized and given by $\chi_D(v_i, h)$. The set of parameters is given by P_D. We define $\chi_D : V \times \mathbb{N}^* \rightarrow P_D$. Actors fire their phases in a cyclo-static fashion. The transition to the next phase (i.e., phase $(h \mod \theta_D(v_i)) + 1$) only occurs after phase h has fired $\chi_D(v_i, h)$ times.

A firing of an actor is enabled when on all input queues of the actor sufficient tokens are present. The number of tokens that are required on a queue $e \in E$ in phase h is parameterized and given by $\gamma(e, h)$. The function γ is defined as $\gamma : E \times \mathbb{N}^* \rightarrow P_D$. Similarly, the parameterized number of tokens produced in phase h (i.e., the parameterized token production quantum) is given by $\pi(e, h)$, where we define $\pi : E \times \mathbb{N}^* \rightarrow P_D$. We require that for every phase, the number of firings of this phase is parameterized in a different parameter than the parameters in which the token transfer quanta of this phase are parameterized. With each parameter $p \in P_D$, a set of integer values is associated, which is given by $\phi_D(p)$. We define $\phi_D : P_D \rightarrow \mathcal{P}(\mathbb{N})$, where $\mathcal{P}(\mathbb{N})$ denotes the set of all subsets of $\mathbb{N}$ excluding the empty subset. Every parameter $p \in P_D$ is only associated with a single phase of a single actor v_i. Further, every parameter p is only allowed to be assigned a value once per iteration through all $\theta_D(v_i)$ phases. Furthermore, the value of a parameter is required to be a function of previously consumed tokens.

The number of initial tokens on queue e is given by $\delta(e)$, with $\delta : E \rightarrow \mathbb{N}$, while the firing duration of an actor $v \in V$ in phase h is given by $\rho(v, h)$, with $\rho : V \times \mathbb{N}^* \rightarrow \mathbb{R}^+$. In any phase h of actor v, tokens are consumed in an atomic action at the start of a firing, and tokens are produced in an atomic action $\rho(v, h)$ later at the finish of the firing. An actor does not start a firing before every previous firing has finished.

We define the following convenience functions. For an actor v_i, the function $E(v_i)$ provides the set of queues adjacent to v_i. We define the parameterized cumulative token production quantum on queue e_{ij} as $\Pi(e_{ij}) = \sum_{h=1}^{\theta_D(v_i)} \chi_D(v_i, h) \cdot \pi(e_{ij}, h)$, and the parameterized cumulative token consumption quantum on e_{ij} is $\Gamma(e_{ij}) = \sum_{h=1}^{\theta_D(v_j)} \chi_D(v_j, h) \cdot \gamma(e_{ij}, h)$. For an actor v_i, the function $P_D(v_i)$ provides the set of parameters in which the cumulative token production quanta on queues from v_i and the cumulative token consumption quanta on queues to v_i are parameterized.

We require that on each queue $e \in E$ there is a parameter valuation such that $\Pi(e) > 0$ and there is a parameter valuation such that $\Gamma(e) > 0$. Furthermore, we only consider strongly connected VPDF graphs.

We will now discuss consistency of dataflow graphs and present a check on consistency of VPDF graphs. For inconsistent VPDF graphs, the throughput constraint cannot be satisfied in bounded memory.

Consistency. On each queue of a VPDF graph, the parameterized production and consumption quanta specify a relation between the execution rates of the two adjacent actors. If there are two directed paths that connect a given pair of actors and that specify inconsistent relations between the execution rates of this pair of actors, then either for any finite number of initial tokens this subgraph will deadlock or there is an unbounded accumulation of tokens. Therefore, in order to verify whether there exists a bounded number of

initial tokens that is sufficient to satisfy the throughput constraint, we need to check whether the relation between the execution rates of each pair of actors is strongly consistent [Bhattacharya and Bhattacharyya 2002; Lee 1991] over all paths between these two actors. We define strong consistency as the requirement that the constraints on execution rates are consistent for every valuation of the token transfer parameters.

On a queue e_{ij} from actor v_i to actor v_j, we have the requirement that it should hold that $z_i \cdot \Pi(e_{ij}) = z_j \cdot \Gamma(e_{ij})$, where actor v_i executes its $\theta_D(v_i)$ phases proportionally z_i times, and actor v_j executes its $\theta_D(v_j)$ phases proportionally z_j times. Similar to Lee [1991] and Bilsen et al. [1996], we collect all these requirements in matrix notation, that is, we require a nontrivial (symbolic) solution to exist for $\Psi \mathbf{z} = \mathbf{0}$, to verify whether a VPDF graph is strongly consistent. The matrix Ψ is a $|E| \times |V|$ matrix, where

$$\Psi_{ij} = \begin{cases} \Pi(e_i) & \text{if } e_i = (v_j, v_k) \\ -\Gamma(e_i) & \text{if } e_i = (v_k, v_j) \\ \Pi(e_i) - \Gamma(e_i) & \text{if } e_i = (v_j, v_j) \\ 0 & \text{otherwise} \end{cases}$$

In the matrix Ψ, each parameter p that can only attain a single value (i.e., $|\phi_D(p)| = 1$) is substituted by this value. The smallest positive integer $\mathbf{z}$ is called the (symbolic) repetition vector of the VPDF graph, and z_i is an element of this repetition vector that denotes the (symbolic) repetition rate of v_i. In the remainder of this article, we only consider strongly consistent VPDF graphs.

For example, if we substitute an actor v_k that consumes and produces a single token in every firing on all its adjacent queues for the subgraph G'_p in Figure 7, then we obtain the following topology matrix and system of linear equations for this VPDF graph.

$$\begin{bmatrix} 1 & -2 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 \\ 0 & p & -1 & 0 \\ 0 & -p & 1 & 0 \\ 0 & 0 & 1 & -p \\ 0 & 0 & -1 & p \end{bmatrix} \begin{bmatrix} z_\tau \\ z_i \\ z_k \\ z_j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The repetition vector is given by the repetition rates $z_\tau = 2$, $z_i = 1$, $z_k = p$, and $z_j = 1$.

The functional behavior of a VPDF graph is deterministic in the sense that it is schedule independent because the firing rule equals the token consumption quanta in that firing and the production rule of a firing equals the token production quanta in that firing. Because the parameters that specify the token consumption and production quanta are a function of previously consumed tokens, we have that the firing rules are sequential [Lee and Parks 1995] and that the firings are functional. These are sufficient conditions for the VPDF graph to be functionally deterministic.

Definition 1 (*Monotonic execution in the start times*). A dataflow graph executes monotonically in the start times of actor firings if no decrease $\Delta \geq 0$ in the start time of any firing can lead to an increase in the start time of any other firing.

For any schedule σ in which on every queue tokens are consumed not before they are produced, a VPDF graph executes monotonically in the start times. This is because the firing rules and token production rules of a firing are independent of the start time of the firing and independent of the arrival times of tokens. Therefore, a Δ earlier start can only lead to a Δ earlier production of the same tokens, which implies that still on every queue, tokens are not consumed before they are produced.

Definition 2 (*Linear execution in the start times*). A dataflow graph has linear temporal behavior if a delay $\Delta \geq 0$ in the start times of actor firings cannot lead to delay larger than Δ for any start time of any firing.

For any schedule σ, in which on every queue tokens are not consumed before they are produced, a VPDF graph has linear temporal behavior. This is because the firing rules are independent of the arrival times of tokens, and the production rules are independent of the start time of that firing. Therefore, let, at time t in σ, a start time be delayed by Δ. By delaying all start times at times $t' \geq t$ in σ, with Δ, again a schedule is obtained in which on every queue tokens are not consumed before they are produced.

3.4 Construction of Analysis Model

We construct a VPDF graph $G = (V, E, P_D, \delta, \rho, \phi_D, \pi, \gamma, \theta_D, \chi_D)$ from a task graph $T = (W, B, P_T, \zeta, \eta, \kappa, \phi_T, \xi, \lambda, \theta_T, \chi_T)$ as follows. Every task $w \in W$ is modeled by an actor $v \in V$, where the number of phases of v equals the number of nonblocking code-segments of w (i.e., $\theta_D(v_v) = \theta_T(w_w)$). Furthermore, the parameterized number of firings of phase h equals the parameterized number of executions of the corresponding nonblocking code-segment c (i.e., $\chi_D(v_v, h) = \chi_T(w_w, c)$).

Further, we have that the firing duration of each phase h of actor v equals the worst-case response time of the corresponding nonblocking code-segment c of the corresponding task w (i.e., $\rho(v, h) = \kappa(w, c)$). A buffer $b_{ab} \in B$ from task w_a to task w_b is modeled by two queues in opposite direction between the actors that model the tasks (i.e., queues $e_{ab}, e_{ba} \in E$ are added if v_a models w_a and v_b models w_b). The number of initial tokens on queue e_{ab} corresponds with the number of initially filled containers on buffer b_{ab} (i.e., $\delta(e_{ab}) = \eta(b_{ab})$). The number of tokens on queue e_{ba} corresponds with the remaining initial containers on b_{ab} (i.e., $\delta(e_{ba}) = \zeta(b_{ab}) - \eta(b_{ab})$).

Every parameter $p_T \in P_T$ is modeled by a parameter $p_D \in P_D$ that has the same set of values associated with it (i.e., $\phi_D(p_D) = \phi_T(p_T)$). Given that e_{ab} and e_{ba} together model buffer b_{ab}. If the number of containers produced on buffer b_{ab} equals p_T, then the number of tokens produced on queue e_{ab} in the VPDF graph equals p_D, where p_D models p_T. In every phase h, the number of tokens consumed from queue e_{ba} equals the number of tokens produced on

queue e_{ab} (i.e., $\gamma(e_{ba}, h) = \pi(e_{ab}, h)$). The number of tokens consumed per firing from queue e_{ba} models the number of empty containers that are required for this execution of task w_a to start. Further, we have that if the number of containers consumed from buffer b_{ab} equals p'_T, then the number of tokens consumed from queue e_{ab} equals p'_D, where p'_D models p'_T. In every phase h, the number of tokens produced on queue e_{ba} equals the number of tokens consumed from queue e_{ab} (i.e., $\pi(e_{ba}, h) = \gamma(e_{ab}, h)$). Throughout this article, actor v_τ models the task w_τ that is required to execute wait-free.

Important differences between the task graph and the dataflow graph are that (i) tasks have a variable response time, while actors have a fixed firing duration, and that (ii) tasks produce and consume containers *between* start and finish of a nonblocking code-segment, while actors produce and consume tokens *at* the start and finish of a phase. These two properties enable the model to have the attractive properties of temporal monotonicity and linearity in the start times, which are the pillars on which our analysis rests.

Furthermore, a buffer is modeled by two dataflow queues in opposite direction that both can have initial tokens, where on one queue these tokens model initially empty containers, and on the other queue, these tokens model initially full containers. In the dataflow graph, it is no longer known (i) which queues together model a buffer and (ii) in which direction the data flows between the tasks. Therefore, given only the dataflow graph there is insufficient information on how to configure the buffers.

3.5 Parameter Distribution

In Section 3.3, we required that every parameter $p \in P_D$ is only associated with a single actor v_i. In this section, we relax this constraint and allow for every parameter p one queue e_p to exist over which the values of this parameter are communicated. Let e_p be from v_i to v_j, then the values of this parameter p are determined in firings of v_i. Actor v_i is required to produce a value on e_p in a phase previous to the phases that have their number of firings or token transfer quanta parameterized in p. Further, actor v_j is required to consume a value from e_p in a phase previous to the phases that have their number of firings or token transfer quanta parameterized in p. Furthermore, we require that on this queue e_p, we have $\delta(e_p) = 0$ and that $\Pi(e_p) = \Gamma(e_p) = 1$. We extend the dataflow graph with the function φ, which returns the parameter value that is communicated over a queue. The function φ is defined as $\varphi : E \rightarrow P_D \cup \{\epsilon\}$, where ϵ is an undefined parameter value. In the dataflow graph of Figure 7, the annotation at the end points of a queue denote the parameterized token transfer quanta per phase. In case the value of a parameter p is transferred, then this is denoted by $1[p]$.

Next to the already mentioned restrictions, we have three additional restrictions. The first additional restriction is a restriction on the topology of the graph and results in a scoping of this parameter p. For example, for the VPDF graph from Figure 7, this restriction does not allow v_i to use the parameter p on edges toward v_τ. Let $\Pi(e_1) \approx p$ mean that $\Pi(e_1)$ is parameterized in p, which means, with v_i producing tokens on e_1, that v_i has a phase h such that $\chi_D(v_i, h) = p$ or

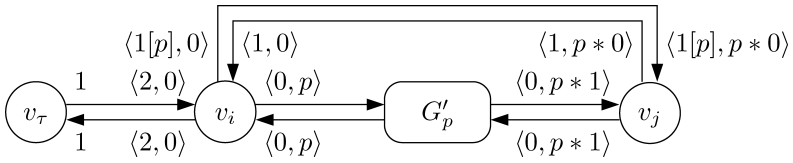


Fig. 7. VPDF graph with shared parameter p. Actors v_i and v_j have two phases, where the second phase of v_j is fired p times. The value of p is determined by v_i and send to v_j.

a phase k such that $\pi(e_1, k) = p$. Let $\Gamma(e_2) \approx p$ have the analogous meaning. Let parameter p be communicated from v_i to v_j. Then, intuitively, we require that every simple directed path that starts with an output queue e_1 of v_i, with $\Pi(e_1) \approx p$, includes an input queue e_2 of v_j, with $\Gamma(e_2) \approx p$, and vice versa from v_j to v_i. This can be verified as follows. Given a VPDF graph G, we create graph G_p^- by removing all output queues e_o of actors v_i and v_j for which $\Pi(e_o)$ is not parameterized in p and by removing all input queues e_i of actors v_i and v_j for which $\Gamma(e_i)$ is not parameterized in p. We require that in G_p^- the same set of actors V'_p is reachable from v_i as is reachable from v_j.

We define G_p to be the subgraph formed by the set of actors V'_p together with the actors v_i and v_j and by all queues from G that have both source and destination actor in G_p. The second additional restriction is that there is a positive integer repetition vector $\mathbf{z}$ of G_p, in which we have that the repetition rates of v_i and v_j are equal to 1 (i.e., $z_i = z_j = 1$). This restriction implies that for every value of p communicated from v_i to v_j there is one iteration of G_p. Therefore, this restriction implies that a strongly consistent VPDF graph executes in bounded memory, while, in general, a strongly consistent dataflow graph does not need to execute in bounded memory [Buck 1993].

The third additional restriction is that actor v_τ, which models the task on which the throughput constraint is specified, is not allowed to be part of a subgraph G_p. This means that there does not exist a parameter p that is communicated from v_i to v_j for which v_τ is on a simple directed path from v_i to v_j that starts and ends with cumulative transfer quanta that are parameterized in p. Furthermore, it is required that $v_\tau \neq v_i$ and $v_\tau \neq v_j$.

4. BUFFER CAPACITIES

In this section, we use the VPDF graph to compute a number of initial tokens that directly corresponds with sufficient capacities for the buffers in the task graph. Our approach is explained through various examples and can be understood independently of the formalized reasoning, which is included to show that indeed our approach is correct for all cases.

The required number of initial tokens depends on the firing durations of the actors. Firings of dataflow actors have a constant firing duration. At the end of this section, Theorem 5 shows that the computed number of tokens is a sufficient buffer capacity, if actor firing durations are upper bounds on task response times. This result is based on temporal monotonicity of the dataflow graph and the required one-to-one function from the task graph to the dataflow graph. In this section, we restrict ourselves to parameters with which a finite

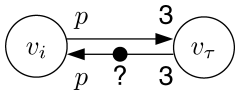


Fig. 8. Example VPDF graph, with $\phi_D(p) = \{2, 3\}$, v_i has firing duration of 2 and v_τ has a firing duration of 3.

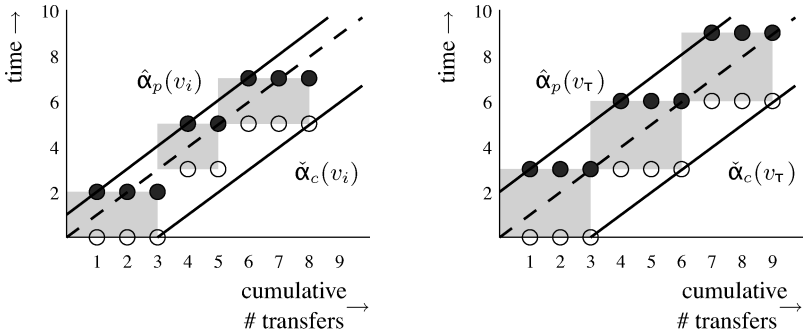
set of parameter values is associated (i.e., for which a maximum value exists). In Section 5, we remove this restriction and discuss our adaptations to the approach that we now discuss.

4.1 Outline of the Approach

The computation of the required number of initial tokens occurs in four steps, (i) computation of the maximum transfer rate per queue, (ii) derivation of actor-schedules per queue, (iii) derivation of minimum distances between the start times of adjacent actors, and (iv) deriving the required number of tokens.

Step 1. Given that actor v_τ is required to execute wait-free, we compute the maximally required token transfer rate on each queue of the dataflow graph. This token transfer rate is maximum in the sense that no possible parameter value can require a larger rate in order to let v_τ execute wait-free. This step is discussed in Section 4.3. In the VPDF graph of Figure 8, the maximum required transfer rate is one token per time unit.

Step 2. On each queue, we define a linear upper bound on token production times, $\hat{\alpha}_p$ and a linear lower bound on token consumption times, $\check{\alpha}_c$. The inverse of the maximum required rate on a queue is taken as the slope of linear bounds on the token production and consumption bounds on this queue. We will show the existence of a schedule of actor firings for every sequence of parameter values such that these bounds are valid. This schedule is shown to exist by creating aggregate firings. Instead of a parameterized number of firings per phase and a parameterized number of transferred tokens per firing, an aggregate firing only has one phase. An aggregate firing has both a parameterized firing duration and parameterized token transfer quanta. It is again monotonicity that tells us that if a schedule of aggregate firings exists, the schedule of actor firings will not lead to later token production times, since actor firings can only start earlier. A sufficient offset of the linear bounds relative to the start time of the first firing in the schedule of aggregate firings is determined such that the bound is conservative. This step is discussed in Section 4.4. The schedules derived for actors v_i and v_τ, from Figure 8, for particular sequences of parameter values are shown in Figures 9(a) and 9(b), respectively. In these figures, the dots denote token transfer times and the shaded rectangles are aggregate firing shapes. Every aggregate firing produces tokens at its finish, denoted by filled dots, and consumes tokens at its start, denoted by open dots. In these figures, the feasibility of an actor corresponds with the requirement that for every possible parameter value the top right corner of the aggregate firing shape is under the dashed line. If the top right corner of the aggregate firing shape is under the dashed line, then we can always delay the start of the next aggregate firing such that the next aggregate firing starts on the dashed



(a) Example derived schedule actor v_i. (b) Example derived schedule actor v_τ.

Fig. 9. Schedules of aggregate firings, which, in this case, are schedules of firings. The filled dots are token productions, the open dots are token consumptions, with productions at the finish of an aggregate firing and consumptions at the start of an aggregate firing. By construction, the left bottom corner of the aggregate firing shapes is on the dashed line that denotes the required transfer rate. The example sequence of parameter values for parameter p is $\langle 3, 2, 3 \rangle$. The first aggregate firing of v_i has a larger than required transfer rate. This results in the constructed schedule in a postponement of the start of the second aggregate firing to start again on the required transfer rate line.

line. This is our procedure to construct a schedule per queue that satisfies the maximum required transfer rate.

Step 3. On every queue, the slopes of the linear bounds on productions and consumption are equal. On each queue, we have determined a difference between the first start time of the actors and their linear bounds on token transfer times. Since tokens can only be consumed after they are produced, this leads to a constraint on the minimum distance between the first start times of these two actors. If there is a constraint on the maximum buffer capacity, which implies a constraint on the maximum number of initial tokens on a queue, then this leads to a constraint on the maximum distance between the first start times of the consumer and the producer. This is modeled by a constraint on the minimum distance in the opposite direction (i.e., between the first start times of the producer and the consumer). Together with the objective to minimize the distances between adjacent actors, the set of constraints on minimum distances between start times leads to a network flow problem that can be solved to obtain start times. These start times satisfy both the constraint that token consumption can only take place after token production and the constraint on maximum number of initial tokens. This step is discussed in Section 4.5. For our example VPDF graph from Figure 8, we have, with maximally zero initial tokens on the queue from v_i to v_τ and no constraint on the maximum number of initial tokens on the queue from v_τ to v_i, that v_τ should start $1^2/3$ later than v_i. The resulting situation is shown in Figure 10. In a VPDF graph as shown in Figure 16, there are multiple paths from v_i to v_j and the minimum difference in start times between these two actors is the maximum over all these paths.

Step 4. The just derived start times together with the bounds on number of token transfers enable the derivation of the required number of initial tokens.

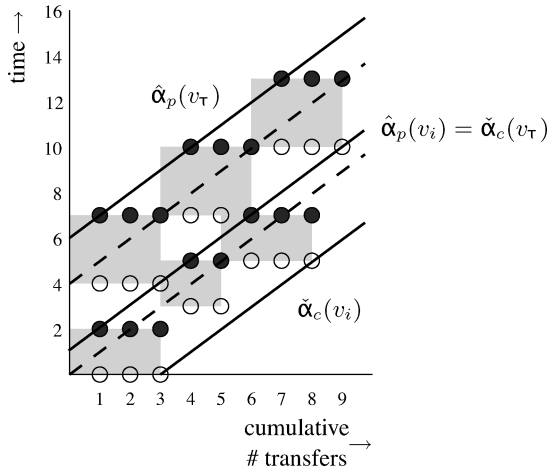
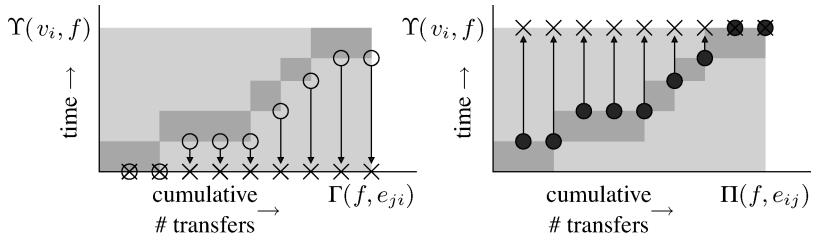


Fig. 10. Resulting example schedule for graph of Figure 8.

This is because, on every queue, the linear bounds have the same slope and the difference between the upper bound on number of consumed tokens and the lower bound on number of produced tokens is an upper bound on the number of initial tokens that is required to enable the schedules discussed in Step 3. This step is discussed in Section 4.6. The required number of initial tokens on the queue from v_τ to v_i for the graph from Figure 8 can be derived from Figure 10. We know that $\hat{\alpha}(v_\tau) - \check{\alpha}(v_i) = 9$ and that the slope equals 1, which means that the maximum number of tokens consumed by v_i, but not yet produced by v_τ according to these schedules, equals 9 tokens. For a graph as shown in Figure 16, the analysis will take into account that the number of tokens on the queue from v_j to v_i needs to compensate for any additional latency of the paths through subgraph G'_p.

The constraints on minimum distance between start times are determined using schedules for actors that are defined per queue in isolation. In Section 4.8, we show that the derived number of tokens is still sufficient if these schedules per queue are coupled to obtain a schedule per actor in isolation.

Furthermore, the required transfer rate on a queue depends on the parameter values. We will show that for every sequence of parameter values, the derived number of initial tokens is sufficient to let v_τ fire wait-free. There are two cases to be considered, (i) parameters that are used by only one actor and (ii) parameters that are used by two actors. For the first type of parameter, we will show that other values can only lead to delayed start times of actors that are further away from actor v_τ. The fact that a firing of v_τ cannot start before every previous firing has finished leads to a delay of the third firing, and also of subsequent firings, in the schedule of v_τ shown in Figure 9(b). This means that this third firing will produce its tokens later on the queue to v_i, thereby potentially delaying v_i. However, since also subsequent firings of v_τ are delayed, tokens will still arrive on time from v_i. This is discussed in detail for any VPDF graph for which each parameter is only used by a single actor in Section 4.8.1.



(a) Changed consumption times (b) Changed production times

Fig. 11. An aggregate firing is a conservative approximation of its firings.

For parameters that are shared by two actors, other values also do not affect the schedule of actor v_τ. This is because, by construction of VPDF graphs, there is always an actor on the path to v_τ of which the schedule is unaffected by the change in parameter value. This is discussed in detail in Section 4.8.2.

We start this section by introducing aggregate firings. Aggregating firings allow for a closed expression for the schedules that we construct per queue in isolation. This drastically reduces the complexity of comparing schedules and reasoning about which schedule has later start times.

4.2 Aggregate Firings

In the reasoning in the following sections, the following additional conservative approximation is made. For each actor v_i, we aggregate all phases to a single phase. On any output queue e_{ij} of v_i, this aggregate phase has a parameterized token production quantum $\Pi(e_{ij})$, and on any input queue e_{hi} of v_i, this aggregate phase has a parameterized token consumption quantum $\Gamma(e_{hi})$. The parameterized firing duration of this aggregate phase is equal to the cumulative firing duration $\Upsilon(v_i) = \sum_{h=1}^{\theta_D(v_i)} \chi_D(v_i, h) \cdot \rho(v_i, h)$. We call a firing of this aggregated phase an aggregated firing. In aggregate firing f, the token production quantum on e_{ij} is given by $\Pi(f, e_{ij})$, the token consumption quantum on e_{hi} is given by $\Gamma(f, e_{hi})$, and the firing duration is given by $\Upsilon(v_i, f)$. Furthermore, we use $\hat{\Gamma}(e_{ij})$ to denote the maximum consumption quantum of an aggregate firing on queue e_{ij} and $\hat{\Upsilon}(v_i)$ to denote the maximum firing duration of an aggregate firing. Both these maximum values are obtained by taking the maximum values of the parameters on which $\Gamma(e_{ij})$ and $\Upsilon(v_i)$ depend.

In Figure 11, an example schedule of firings is denoted by the darker firing shapes. The corresponding aggregate firing shape is denoted by the lighter color. This aggregate firing consumes all $\Gamma(f, e_{ji})$ tokens at its start, see Figure 11(a), and produces all $\Pi(f, e_{ij})$ tokens at its finish, which is $\Upsilon(v_i, f)$ later than its start, see Figure 11(b). As illustrated in Figure 11, an aggregate firing requires the same number of tokens to be present earlier than a normal firing and delays token production times. This abstraction of multiple firings to a single aggregate firing leads to a reduced accuracy of the analysis. If there exists a schedule of aggregate firings, then this implies the existence of a schedule of normal firings. This is because normal firings can only start earlier and produce their tokens earlier than the aggregate firings. By monotonicity of VPDF graphs, we know that earlier start times and earlier token production

times cannot lead to later token production times anywhere in the VPDF graph. This implies that if the number of initial tokens allows v_τ to fire wait-free in case of aggregate firings by all actors, then v_τ can still fire wait-free in case all actors have normal firings.

4.3 Maximum Token Transfer Rates

In this section, we derive the maximum required token transfer rate of each queue that is required to guarantee that given a bounded number of initial tokens still sufficient tokens are available to let actor v_τ fire wait-free. This means that for each queue, we need to find the values for all parameters of the VPDF graph that maximize the number of tokens transferred on this queue, given that v_τ fires wait-free. This section explains Step 1 of the approach that is outlined in Section 4.1.

The maximum required token transfer rate on queue e_{ij} is defined by Equation (1) that specifies that $\hat{r}(e_{ij})$ is the maximum over all possible parameter values of a certain ratio. This ratio includes z_i/z_τ, which is the ratio of the repetition rate of v_i and the repetition rate of v_τ. If each actor v_x in a graph fires z_x times, then on each queue of that graph the number of produced tokens equals the number of consumed tokens. This implies that z_i/z_τ specifies how often v_i needs to fire relative to a firing of v_τ to satisfy the requirement of a bounded number of initial tokens. Given that v_τ fires every τ, this implies that v_i is required to fire z_i/z_τ times every τ (i.e., with firing rate $z_i/z_\tau \cdot \tau$). The parameterized token production rate on e_{ij} is given by the parameterized cumulative token production quantum, that is, $\Pi(e_{ij})$, times the just derived firing rate, which equals $z_i/z_\tau \cdot \tau$. Equation (1) defines the maximum required token transfer rate per queue. This is a maximum rate per queue because we only consider graphs that can execute in bounded memory, which implies that the maximum required consumption rate on a queue equals the maximum required production rate on that queue.

Definition 3. The maximum required token transfer rate on queue e_{ij} is given by the maximum value that Equation (1) can attain, where $\phi_D(P_D)$ denotes the set of assignments of values to the parameters of G.

$$\hat{r}(e_{ij}) = \max_{\phi_D(P_D)} \frac{\Pi(e_{ij}) \cdot z_i}{z_\tau \cdot \tau} \tag{1}$$

Determining the maximum required transfer rate on a queue requires to find the parameter values that maximize the ratio $\Pi(e_{ij}) \cdot z_i/z_\tau \cdot \tau$. Theorem 1 shows that to determine these parameter values, instead of considering all possible combinations of parameter values, only all combinations of the extreme values that the parameters can attain need to be considered. This is because Lemma 2 shows this ratio is monotone in every parameter. Lemma 1 is used in the proof of Lemma 2 to rewrite this ratio.

LEMMA 1. *Given a directed path of length n from an actor v_i to an actor v_z that consists of queues e_1 through e_n, then*

$$\frac{z_i}{z_z} = \frac{\Gamma(e_1) \cdot \ \ldots \ \cdot \Gamma(e_n)}{\Pi(e_1) \cdot \ \ldots \ \cdot \Pi(e_n)}. \tag{2}$$

PROOF. The proof is by induction over the queues of such a path.

Base step. Let e_{ij} be the first queue on the path. We have that $z_i \cdot \Pi(e_{ij}) = z_j \cdot \Gamma(e_{ij})$, which implies $z_i/z_j = \Gamma(e_{ij})/\Pi(e_{ij})$.

Induction step. Given that for queues e_1 to e_k, we have that $\frac{z_i}{z_x} = \frac{\Gamma(e_1)\cdot\ldots\cdot\Gamma(e_k)}{\Pi(e_1)\cdot\ldots\cdot\Pi(e_k)}$. Let e_{k+1} be queue e_{xy}. Then, for this queue, we have that $z_x/z_y = \Gamma(e_{k+1})/\Pi(e_{k+1})$. This implies that $\frac{z_i}{z_y} = \frac{\Gamma(e_1)\cdot\ldots\cdot\Gamma(e_{k+1})}{\Pi(e_1)\cdot\ldots\cdot\Pi(e_{k+1})}$. □

Lemma 2 shows that the parameterized ratio in Equation (1) is monotone in every parameter. This enables a straightforward procedure as presented in Theorem 1 to find the parameter values that lead to the maximal value that this ratio can attain.

LEMMA 2. *For any parameter $p \in P_D$, the ratio as given by Equation* (3) *is monotone in p.*

$$\frac{\Pi(e_{ij}) \cdot z_i}{z_\tau \cdot \tau} \tag{3}$$

PROOF. From Lemma 1, we know that if there is a directed path of length n from v_i to v_τ consisting of queues e_1 through e_n, then Equation (3) can be rewritten into Equation (4). There is always such a directed path because we only consider strongly connected VPDF graphs.

$$\frac{\Pi(e_{ij}) \cdot \Gamma(e_1) \cdot \ldots \cdot \Gamma(e_n)}{\Pi(e_1) \cdot \ldots \cdot \Pi(e_n) \cdot \tau} \tag{4}$$

This expression can be factored into Equation (5), where each factor is a ratio of cumulative transfer quanta of a single actor on this path from v_i to v_τ.

$$\frac{\Pi(e_{ij})}{\Pi(e_1)} \cdot \frac{\Gamma(e_1)}{\Pi(e_2)} \cdot \ldots \frac{\Gamma(e_{n-1})}{\Pi(e_n)} \cdot \frac{\Gamma(e_n)}{\tau} \tag{5}$$

A cumulative token transfer quantum is a sum of products, where the number of summands equals the number of phases and each product has as factors a firing parameter and a transfer parameter. A cumulative firing duration such as τ is also a sum of products, where again the number of summands equals the number of phases and each product now has as factors a firing parameter and a firing duration.

Let us first consider the case of parameters that are not shared between two actors. These parameters can only be present in one factor of Equation (5). Say that x/y is that factor from Equation (5) in case of parameter p. Here, x stands for the numerator of this factor, which is either a cumulative token production or consumption quantum, and y stands for the denominator of this factor, which is either a cumulative token production or consumption quantum or a cumulative firing duration. Because p is only allowed to be used in a single phase of an actor, we can find α and q such that $x = \alpha p + q$, and we can find β and r such that $y = \beta p + r$, where q and r are expressions that do not include p. Because it is not allowed that a parameter is used both to parameterize the number of firings of a phase as well as a token transfer quantum of that same phase, also α and β can be found that are independent of p. This implies that the numerator x and the denominator y are linear in p. Consequently, the

derivative of this factor x/y with respect to p is $\alpha\beta p+\alpha r-\alpha\beta p-\beta q/(\beta p+r)^2 = \alpha r-\beta q/(\beta p+r)^2$. In this derivative, the numerator is independent of p, while the denominator is always positive. This implies that the sign of the derivative is independent of p and, therefore, that this factor is monotone in p.

For the case that parameter p is shared by actors v_a and v_b, we have that $z_a/z_b = 1$. This implies by Lemma 1 that if a directed path from v_a to v_b is part of the directed path from v_i to v_τ that the cumulative token transfer quanta that include p cancel each other out. In other words, Equation (5) will not include parameters that are shared by two actors that are on a path from v_i to v_τ, and every parameter will only occur in one factor of Equation (5). This implies that the previous reasoning for parameters that are not shared between two actors covers all parameters that are relevant for this proof. □

Lemma 2 tells that Equation (3) is monotone in every parameter. This means that when considering a single parameter and taking constant values for all other parameters, then this expression is monotonically increasing or decreasing in this parameter. Note, however, that whether the expression is increasing or decreasing in this parameter depends on the values chosen for the other parameters. This means that to determine the maximum required transfer rate, all combinations of extreme values of the parameters need to be considered.

THEOREM 1. *For a queue e_{ij}, the maximum required rate $\hat{r}(e_{ij})$ is found by considering all combinations of the extreme values of the parameters in which the right-hand side of Equation* (1) *is parameterized.*

PROOF. This follows immediately from Lemma 2 that says that the ratio over which is maximized is monotone in every parameter. □

Theorem 2 establishes a necessary and sufficient condition on the cumulative firing duration of an actor v_i in order for a schedule of v_i to exist that has the maximum required rate.

THEOREM 2. *For every actor $v_i \in V$, there exists a schedule for all parameter values such that this schedule has the maximum required rate if Equation* (6) *holds.*

$$\max_{\phi_D(P_D)} \frac{\Upsilon(v_i) \cdot z_i}{\tau \cdot z_\tau} \leq 1 \tag{6}$$

PROOF. If all actors v_j in the graph fire z_j times, then, on each queue, the number of tokens produced equals the number of tokens consumed. The left part of Equation (6) finds the maximum time that it takes v_i to fire z_i times relative to the time it takes v_τ to fire z_τ times. Since v_τ should fire wait-free, it is required that, for all parameter values, v_i requires not more time to fire z_i times than v_τ requires to fire z_τ times. □

The cumulative firing duration is a sum of products, where the number of summands equals the number of phases. Each product has as factors a firing duration and a parameter that specifies the number of firings of this phase. The cumulative production quantum on a queue is an expression with the same structure, but has transfer parameters instead of firing durations. Therefore,

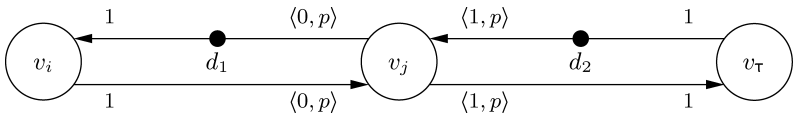


Fig. 12. Example VPDF graph, with $p \in \{1, 2, 3\}$.

since Equation (3) is monotone in every parameter, the expression over which is maximized in Equation (6) is also monotone in every parameter, and the necessary condition for feasibility of the graph is verified by considering all combinations of extreme parameter values.

For example, consider the VPDF graph as shown in Figure 12, where v_i has a firing duration of $4t$, v_j has a firing duration of t for both phases, and v_τ has a firing duration of $3t$. The repetition rates of the actors in this graph are $z_i = p$, $z_j = 1$, and $z_\tau = p + 1$. The feasibility check for v_i amounts to determining the value of p that results in the maximum value of $(4t \cdot p)/(3t \cdot (p+1))$. For $p = 1$, we have $(4t \cdot p)/(3t \cdot (p+1)) = 4/6$, while for $p = 3$, we have $(4t \cdot p)/(3t \cdot (p+1)) = 1$. Therefore, $p = 3$ maximizes this ratio, and we conclude that the firing durations of v_i allow to satisfy the throughput constraint of the graph. For v_j, we maximize $(2t \cdot 1)/(3t \cdot (p+1))$. For $p = 1$, we have $(2t \cdot 1)/(3t \cdot (p+1)) = 2/6$, while for $p = 3$, we have $(2t \cdot 1)/(3t \cdot (p+1)) = 2/12$. Therefore, $p = 1$ maximizes this ratio and we conclude that the firing durations of v_j allow to satisfy the throughput constraint of the graph. The firing durations of actor v_τ impose the throughput constraint on the graph.

In the VPDF graph shown in Figure 12, we have that the maximal value of p determines whether v_i can satisfy the throughput constraint, while the minimal value of p determines whether v_j can satisfy the throughput constraint. This is different in a VRDF graph [Wiggers et al. 2008], in which we have that for every parameter there is a single extreme value that triggers the worst-case behavior.

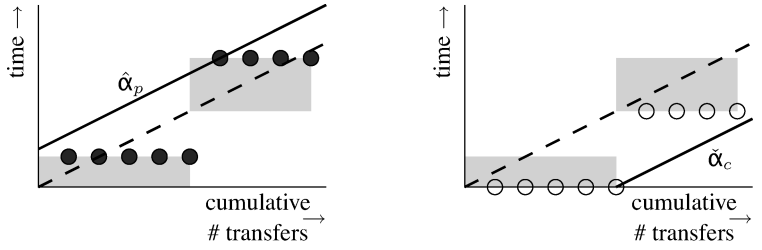
4.4 Schedules per Queue

In this section, we show how to compute a sufficient difference between a linear upper bound on production times and a linear lower bound on consumption times such that for every sequence of parameter values there exists a schedule between these bounds. This section explains Step 2 of the approach that is outlined in Section 4.1.

Given two actors v_i and v_j, and a queue e_{ij}. The linear upper bound on the production time of token x on e_{ij} under schedule $\sigma(v_i, e_{ij})$ is given by $\hat{\alpha}_p(x, e_{ij}, \sigma(v_i, e_{ij}))$, while $\check{\alpha}_c(x, e_{ij}, \sigma(v_j, e_{ij}))$ is the linear lower bound on the consumption time of token x on e_{ij} under schedule $\sigma(v_j, e_{ij})$. The schedules considered in this section are schedules of aggregate firings.

Let $\Xi(f, e_{ij})$ be the cumulative number of tokens produced on queue e_{ij} in aggregate firings one up to and including firing f, with $f \in \mathbb{N}^*$ and $\Xi(0, e_{ij}) = \delta(e_{ij})$, where $\delta(e_{ij})$ is the number of initial tokens on e_{ij}. The start time of the first aggregate firing of actor v_i is denoted by $s(v_i)$. We define the start time of aggregate firing f in schedule $\sigma(v_i, e_{ij})$ for v_i on queue e_{ij} by:

$$s(f, \sigma(v_i, e_{ij})) = s(v_i) + \frac{\Xi(f-1, e_{ij}) - \delta(e_{ij})}{\hat{r}(e_{ij})}. \tag{7}$$



(a) Production times and bound (b) Consumption times and bound

Fig. 13. Schedules of aggregate firings on output and input queue. Schedule is constructed such that the left bottom corner of aggregate firing shape is on the maximum required rate line, denoted by the dashed line. With productions at the finish of an aggregate firing and consumptions at the start, we bound production times from above and consumption times from below, given the constructed schedule of aggregate firings.

We require that all schedules defined for an actor have an equal start time for the first aggregate firing of this actor.

This schedule implies that token $\Xi(f-1, e_{ij})+1$ up to and including token $\Xi(f, e_{ij})$ are produced at $s(f, \sigma(v_i, e_{ij})) + \Upsilon(v_i, f)$ (i.e., at the finish time of the aggregate firing). With $s(f, \sigma(v_i, e_{ij}))$ given by Equation (7) and $\Upsilon(v_i, f) \leq \hat{\Upsilon}(v_i)$, an upper bound on the finish time of aggregate firing f is given by Equation (8).

$$s(f, \sigma(v_i, e_{ij})) + \Upsilon(v_i, f) \leq s(v_i) + \hat{\Upsilon}(v_i) + \frac{\Xi(f-1, e_{ij}) - \delta(e_{ij})}{\hat{r}(e_{ij})} \tag{8}$$

Let token x, with $x \in \mathbb{N}^*$, be produced by aggregate firing f, then $x = \Xi(f-1, e_{ij}) + y$, with $1 \leq y \leq \Xi(f, e_{ij}) - \Xi(f-1, e_{ij})$. Substitution of $x-y$ for $\Xi(f-1, e_{ij})$ in Equation (8) results in Equation (9).

$$s(f, \sigma(v_i, e_{ij})) + \Upsilon(v_i, f) \leq s(v_i) + \hat{\Upsilon}(v_i) + \frac{x - y - \delta(e_{ij})}{\hat{r}(e_{ij})} \tag{9}$$

Because $y \geq 1$, Equation (10) is an upper bound on the right-hand side of Equation (9). A linear upper bound on the production time of token x, on queue e_{ij} with schedule $\sigma(v_i, e_{ij})$ is, therefore, given by Equation (10).

$$\hat{\alpha}_p(x, e_{ij}, \sigma(v_i, e_{ij})) = s(v_i) + \hat{\Upsilon}(v_i) + \frac{x - \delta(e_{ij}) - 1}{\hat{r}(e_{ij})} \tag{10}$$

An example schedule of token production times of aggregate firings is shown in Figure 13(a). The start time of every aggregate firing is such that the bottom left corner of the firing shape is on the dashed line. This means that if the first aggregate firing of Figure 13(a) produces more tokens, then the start time of the second aggregate firing is delayed more. As illustrated in this figure, given this schedule of aggregate firings, the upper bound on token transfer times is determined by the aggregate firing that has the largest cumulative firing duration.

Let $\Lambda(f, e_{ij})$ be the cumulative number of tokens consumed from queue e_{ij} in aggregate firings one up to and including firing f, with $f \in \mathbb{N}^*$ and $\Lambda(0, e_{ij}) = 0$. We define the start time of aggregate firing f in schedule $\sigma(v_j, e_{ij})$ for v_j on queue e_{ij} by:

$$s(f, \sigma(v_j, e_{ij})) = s(v_j) + \frac{\Lambda(f-1, e_{ij})}{\hat{r}(e_{ij})}. \tag{11}$$

This means that token $\Lambda(f-1, e_{ij})+1$ up to and including token $\Lambda(f, e_{ij})$ are consumed at $s(f, \sigma(v_i, e_{ij}))$. Let token x, with $x \in \mathbb{N}^*$, be consumed by aggregate firing f, then $x = \Lambda(f-1, e_{ij}) + y$, with $1 \leq y \leq \Lambda(f, e_{ij}) - \Lambda(f-1, e_{ij})$. Substitution of $x - y$ for $\Lambda(f-1, e_{ij})$ in Equation (11) results in Equation (12).

$$s(f, \sigma(v_j, e_{ij})) = s(v_j) + \frac{x-y}{\hat{r}(e_{ij})} \tag{12}$$

Because $\Lambda(f, e_{ij}) - \Lambda(f-1, e_{ij}) \leq \hat{\Gamma}(e_{ij})$, Equation (13) is a lower bound on Equation (12). A linear lower bound on the consumption time of token x from queue e_{ij} with schedule $\sigma(v_j, e_{ij})$ is, therefore, given by Equation (13).

$$\check{\alpha}_c(x, e_{ij}, \sigma(v_j, e_{ij})) = s(v_j) + \frac{x - \hat{\Gamma}(e_{ij})}{\hat{r}(e_{ij})} \tag{13}$$

An example schedule of token consumption times of aggregate firings is shown in Figure 13(b). Similar to the schedule created for a token producing actor, also in the schedule of a token consuming actor, aggregate firings have a start time such that the bottom left corner of the firing shape is on the dashed line. This implies that a larger token consumption quantum leads to a larger delay of the next aggregate firing. Given this schedule, the lower bound on token consumption times is determined by the aggregate firing that has the largest cumulative consumption quantum.

Since, on any queue, tokens can only be consumed after they have been produced, we have that for every token x, $\check{\alpha}_c(x, e_{ij}, \sigma(v_j, e_{ij})) \geq \hat{\alpha}_p(x, e_{ij}, \sigma(v_i, e_{ij}))$ should hold. After substitution of Equations (10) and (13) in $\check{\alpha}_c(x, e_{ij}, \sigma(v_j, e_{ij})) \geq \hat{\alpha}_p(x, e_{ij}, \sigma(v_i, e_{ij}))$, we obtain Equation (14).

$$s(v_j) - s(v_i) \geq \frac{\hat{\Gamma}(e_{ij}) - \delta(e_{ij}) - 1}{\hat{r}(e_{ij})} + \hat{\Upsilon}(v_i) \tag{14}$$

4.5 Computation of Start Times of First Firings

The previous section derived for each queue a constraint on the minimum difference between start times of first firings of the actors adjacent to this queue. This constraint on the minimum difference between these start times is such that on this queue no tokens are consumed before they are produced, given the schedules that are determined for this queue in isolation. This section discusses how to compute the start time of the first firing of each actor given these constraints on the minimum difference between start times, this will explain Step 3 of the approach outlined in Section 4.1.

Similar to Wiggers et al. [2007a], we interpret the predefined number of initial tokens on a queue e (i.e., $\delta(e)$) as a constraint. This constraint on the maximum number of tokens implies a constraint on the maximum difference in start times of the adjacent actors. A constraint on the maximum difference between the start times of v_j and v_i can be reformulated as a constraint on the minimum difference between v_i and v_j (i.e., $s(v_j) - s(v_i) \leq 10$ is equivalent to $s(v_i) - s(v_j) \geq -10$). This reformulation of a constraint on the maximum difference in start times to a constraint on the minimum difference in start times is taken into account by the way the number of initial tokens affects the upper bound on token production times, leading to inclusion of $\delta(e)$ with a negative sign in Equation (14). These minimum differences form the constraints in the network flow problem in Algorithm 4.5 that minimizes the differences between all start times subject to the mentioned constraints. This is done by introducing a dummy-actor v_0 and minimizing all start times relative to the start time of a dummy-actor v_0. If there is a solution that satisfies the constraints, then solving this network flow problem results in the start times of the first firings of each actor.

The start times of the first firings of each actor are computed as follows. We construct a graph G_0 from a VPDF graph G as follows. We extend the VPDF graph with an additional actor v_0, $V_0 = V \cup \{v_0\}$, and extend the set E with queues from v_0 to every actor in V to obtain the set of edges E_0. We define the valuation function $\beta : E \rightarrow \mathbb{R}$ that, with each queue, e_{ij} associates the constraint on the minimum difference in start times as expressed by Equation (14); that is, for queue $e_{ij} \in E$, we define.

$$\beta(e_{ij}) = \frac{\hat{\Gamma}(e_{ij}) - \delta(e_{ij}) - 1}{\hat{r}(e_{ij})} + \hat{\Upsilon}(v_i), \tag{15}$$

while for every queue e adjacent to v_0, we have $\beta(e) = 0$. Subsequently, we solve the linear programming problem in Algorithm 4.5. By negating the start times and constraints, the problem in Algorithm 4.5 can be rewritten to a standard form of the dual of the uncapacitated network flow problem. Even more specifically, it is such that it is a shortest path problem, which can be solved with Bellman-Ford. See Bertsimas and Tsitsiklis [1997] for background information on linear programming, the dual of network flow problems, and their relation to shortest path problems. Detection of a negative cycle by Bellman-Ford implies infeasibility of the specified timing and resource constraints.

Algorithm 1. Start time computation

$$\min \sum_{v_i \in V} s(v_i)$$

subject to

$$s(v_j) - s(v_i) \geq \beta(e_{ij}) \qquad \forall e_{ij} \in E_0$$

$$s(v_0) = 0$$

4.6 Buffer Capacity Computation

This section discusses how the number of initial tokens on each queue is derived and explains Step 4 of the approach that is outlined in Section 4.1. Given the start times of the first firings and the linear bounds on token production and consumption times, which are defined in these start times, a sufficient number of initial tokens is computed on each queue. Since start times are only computed when all constraints are satisfied, the resulting number of initial tokens on each queue is smaller than or equal to the constraint on maximum number of initial tokens on that queue. The computation of the number of initial tokens is is done by rewriting Equations (14) through (16). We take the smallest integer value that satisfies the constraint in Equation (16) as the number of initial tokens.

$$\delta(e_{ij}) \geq \hat{\Gamma}(e_{ij}) - 1 + \hat{r}(e_{ij})(\hat{\Upsilon}(v_i) + s(v_i) - s(v_j)) \tag{16}$$

4.7 Computational Complexity

Overall, our approach to compute the required number of tokens, which we will show corresponds directly to the required buffer capacities has a computational complexity that is exponential in the number of parameters and polynomial in the graph size.

In more detail, our approach has the following steps. We first have a number of checks on the validity of the VPDF graph, (i) strongly consistent, (ii) strongly connected, (iii) scoping of shared parameters, (iv) repetition rates within scope of shared parameters, and (v) v_τ not in scope of shared parameter. The computational complexities of these checks is as follows, see Cormen et al. [2001]. To check for strong consistency, we solve a system of linear equations and have $O(E^3)$. To check that the graph is strongly connected, we have $O(V + E)$. To check for proper scoping of shared parameters and whether v_τ is not within the scope of a shared parameter, we have $O(V + E)$ for every shared parameter. To check for the repetition rates within scope of shared parameters, we solve a system of linear equation for each shared parameter, which results in $O(PE^3)$. Subsequently to these checks, (i) the maximum required transfer rates are determined, (ii) firing durations are checked, (iii) constraints on minimum differences in start times are determined, (iv) start times are computed, and (v) a sufficient number of tokens is determined. Determining the maximum required transfer rates and the check on firing durations both are exponential in the number of parameters, with complexities of $O(E2^P)$ and $O(V2^P)$, respectively. Determining the constraints requires determining the maximum cumulative consumption quantum and the maximum cumulative firing duration, which implies a complexity of $O(EK)$, with $K = \max(\{\theta_D(v_i)|v_i \in V\})$. Determining the number of tokens is linear in the number of edges (i.e., $O(E)$). Computing the start times can be done with Bellman-Ford, which has a complexity $O(VE)$. Therefore, the complexity of our complete algorithm is $O(V + VE + V2^P + EK + E^3 + PE^3 + E2^P)$.

4.8 Sufficiency of Buffer Capacities

In this section, we show that the computed number of initial tokens allows actor v_τ to fire with period τ for all possible parameter values. This is shown

in two steps. To explain these steps, we define all parameters to be *local* except when parameter values are communicated to another actor. Parameters of which values are communicated to another actor are called *shared*. In the first step, we assume that all parameters that are shared by two actors, see Figure 16, are constant (i.e., can only attain a single value). With that assumption, Lemmas 5 through 7 will help in establishing Theorem 3, which states that v_τ can execute wait-free for all possible values of local parameters. Subsequently, we show that this assumption can be removed. Lemmas 10 and 11 will help in establishing Theorem 4 that states that v_τ can execute wait-free, even if parameters are shared by two actors, given the number of initial tokens as computed in Section 4.6.

To compute the number of initial tokens, we have delayed start times of aggregate firings and used worst-case response times of code-segments. Theorem 5 will show that this is all allowed and that indeed the number of tokens computed for the VPDF graph as computed in Section 4.6 is a sufficient buffer capacity in the task graph that guarantees that the throughput constraint is satisfied. This is shown by using the result of Theorem 4, the one-to-one correspondence between tokens and containers in the dataflow graph and the task graph, and the fact that VPDF graphs have monotonic temporal behavior.

We now start by introducing a number of concepts required in the later proofs. The number of tokens is computed with schedules of aggregate firings per queue. However, it is important to note that an actor can only have a single schedule and not one schedule for each adjacent queue. The following definitions will relate the individual schedules of the queues adjacent to an actor to the resulting schedule of that actor. Subsequently, we will define shared parameters and define the assumption that shared parameters have constant values. This enables us to show in Section 4.8.1 that the number of initial tokens is sufficient in case there are no shared parameters, after which we show in Section 4.8.2 that also with shared parameters, the computed number of initial tokens is sufficient for all parameter values.

Definition 4. A queue-schedule is a function that given an aggregate firing f, with $f \in \mathbb{N}^*$, an actor v_i, and a queue e specifies the start time of the f-th aggregate firing of actor v_i when considering this queue with adjacent actors in isolation.

Definition 5. The start time of an aggregate firing in queue-schedule $\sigma(v_i, e)$ of actor v_i on queue e is given by Equation (7) in case e is an output queue of v_i and given by Equation (11) in case e is an input queue of v_i.

Queue-schedule $\sigma(v_i, e)$ can be linearly bounded and the start time of the first firing is computed in Algorithm 4.5.

Definition 6. For actor v_j, queue-schedule $\sigma(v_j, e_{ij})$ on input queue e_{ij} is valid, denoted by $\mathsf{valid}(\sigma(v_j, e_{ij}))$, if for every token x that is transferred over e_{ij} the production time of x resulting from queue-schedule $\sigma(v_i, e_{ij})$ is earlier than or equal to the consumption time of x resulting from queue-schedule $\sigma(v_j, e_{ij})$.

Definition 7. For two queue-schedules σ_1 and σ_2 that determine start times of an actor v_i on queue e, we have $\sigma_1 \leq \sigma_2$ if the start time of every firing f in σ_1 is not later than the start time of f in σ_2, that is:

$$\sigma_1 \leq \sigma_2 \Leftrightarrow \forall f \in \mathbb{N}^* \bullet s(f, \sigma_1) \leq s(f, \sigma_2). \tag{17}$$

In this work, the symbol $\bullet$ is used to separate the quantifiers from the proposition.

Definition 8. An actor-schedule is a function that given an aggregate firing f, with $f \in \mathbb{N}^*$, of actor v_i specifies the start time of the f-th aggregate firing of actor v_i.

Definition 9. For every actor v_i, actor-schedule $\sigma(v_i)$ is given by taking for every firing the latest start time of v_i in the queue-schedules constructed on the queues adjacent to actor v_i.

$$\sigma(v_i) = \max(\{\sigma(v_i, e) | e \in E(v_i)\}) \tag{18}$$

The schedule that is defined by Equation (18) is a schedule that is constructed independently of token arrival times. Definition 10 defines when this constructed schedule is valid with respect to token arrival times.

Definition 10. For actor v_j, schedule $\sigma(v_j)$ is valid, denoted $\mathsf{valid}(\sigma(v_j))$, if we have that $\forall e_{ij} \in E(v_j) \bullet \mathsf{valid}(\sigma(v_j, e_{ij}))$.

In Definition 6, a constructed schedule on an input queue is defined valid if token consumption times are not earlier than token production times. Definition 11, defines the schedule on an output queue of an actor to be valid if (i) the constructed schedule of the actor equals the schedule on this output queue, and (ii) on all input queues of this actor, tokens arrive in time to enable the constructed schedule of this actor.

Definition 11. For actor v_i and output queue e_{ij}, schedule $\sigma(v_i, e_{ij})$ is valid, denoted by $\mathsf{valid}(\sigma(v_i, e_{ij}))$, if $\mathsf{valid}(\sigma(v_i))$ and $\sigma(v_i) = \sigma(v_i, e_{ij})$.

Lemmas 3 and 4 derive a difference between subsequent start times in the queue-schedules of the producer and consumer on a queue. These results will later be used to compare queue-schedules and reason about equivalence of queue-schedules.

LEMMA 3. *The difference between the start time of aggregate firing $f + 1$ and the start time of aggregate firing f of actor v_i in queue-schedule $\sigma(v_i, e_{ij})$ is given by the ratio of the token production quantum of aggregate firing f and the maximum required rate on queue e_{ij}, that is:*

$$s(f + 1, \sigma(v_i, e_{ij})) - s(f, \sigma(v_i, e_{ij})) = \frac{\Pi(f, e_{ij})}{\hat{r}(e_{ij})}. \tag{19}$$

PROOF. Equation (7) says that

$$s(f + 1, \sigma(v_i, e_{ij})) = s(v_i) + \frac{\Xi(f, e_{ij}) - \delta(e_{ij})}{\hat{r}(e_{ij})} \tag{20}$$

and

$$s(f, \sigma(v_i, e_{ij})) = s(v_i) + \frac{\Xi(f-1, e_{ij}) - \delta(e_{ij})}{\hat{r}(e_{ij})}. \tag{21}$$

Subtracting Equation (21) from Equation (20) together with $\Xi(f, e_{ij}) - \Xi(f-1, e_{ij}) = \Pi(f, e_{ij})$ implies that this lemma holds. □

LEMMA 4. *The difference between the start time of aggregate firing* $f+1$ *and the start time of aggregate firing* f *of actor* v_j *in queue-schedule* $\sigma(v_j, e_{ij})$ *is given by the ratio of the token consumption quantum of aggregate firing* f *and the maximum required rate on queue* e_{ij}*, that is:*

$$s(f+1, \sigma(v_j, e_{ij})) - s(f, \sigma(v_j, e_{ij})) = \frac{\Gamma(f, e_{ij})}{\hat{r}(e_{ij})}. \tag{22}$$

PROOF. This proof is analogous to the proof of Lemma 3, however, now starting from Equation (11). □

The set of parameters shared by two actors is given by Definition 12. This definition is used in Assumption 1 to be able to specify that shared parameters are assumed constant.

Definition 12. A shared parameter is a parameter that is used by two actors. The set of shared parameters is given by:

$$S = \{p | \exists v_a, v_b \in V \bullet v_a \neq v_b \wedge p \in P_D(v_a) \cap P_D(v_b)\}. \tag{23}$$

ASSUMPTION 1. *Any parameter* p *that is shared by two actors has a constant value* $\hat{p}$ *that equals the maximum value that* p *can attain, that is,* $\hat{p} = \max(\phi_D(p))$.

$$\forall p \in S \bullet p = \hat{p} = \max(\phi_D(p)) \tag{24}$$

Under Assumption 1, only parameters that are local to an actor can attain different values. This means that for shared parameters as shown in Figure 16, it is assumed that they can only attain a single value, that is, shared parameters are assumed to be constants. In the next section, we will show that under this assumption the computed buffer capacities are sufficient. Subsequently in Section 4.8.2, we will show that Assumption 1 can be removed and that then still the computed buffer capacities are sufficient to guarantee that actor v_τ can execute wait-free.

4.8.1 *Local Parameters.* At the end of this section, Theorem 3 concludes that the number of initial tokens as computed by Equation (16) is sufficient given Assumption 1. Intuitively, our reasoning will show that schedules of actors that are closer to v_τ dominate schedules of actors that are further away from v_τ. These schedules dominate each other in the sense that delay is only pushed away from v_τ and that no actor can delay the schedule of an actor closer to v_τ. In the end, this will imply that no actor can delay the schedule of actor v_τ. This is proven by showing that every actor produces tokens on time on queues toward v_τ given that tokens arrive on time on queues away from v_τ. Lemma 5 will show that given that tokens arrive on time on all input queues for the

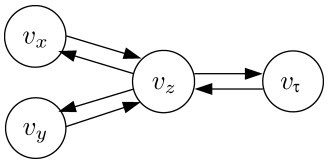
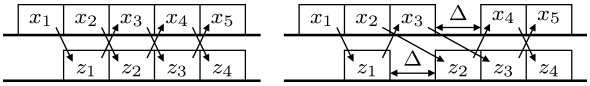


Fig. 14. Example VPDF graph that is used to explain the outline of Section 4.8.1.



(a) Original schedule (b) Schedule with Δ delay

Fig. 15. Illustration of linearity.

queue-schedules, that tokens arrive on time for the actor-schedule. Lemma 6 shows that the actor-schedule equals the queue-schedule on queues toward v_τ. Given that tokens arrive on time, Lemmas 5 together with Lemma 6 says that on queues toward v_τ, the queue-schedule of the token producing actor is valid. Lemma 7 shows that for any queue a valid queue-schedule by the token producing actor implies a valid queue-schedule by the token consuming actor. By repeated application of these lemmas, Theorem 3 is proven.

More specifically, for the example VPDF graph as shown in Figure 14, the reasoning is as follows. Note that it does not matter whether v_τ models an input or output interface. We first assume that on all queues away from actor v_τ tokens arrive on time. Given this assumption, we have that actors v_x and v_y produce their tokens on time. This implies that tokens arrive on time on all inputs of actor v_z, which implies that tokens arrive on time at actor v_τ. This is because the schedules constructed on the queues to and from actors v_x and v_y have such a transfer rate that they will never lead to delayed start times of actor v_z. Depending on the consumption rate of v_τ, actor v_z can have delayed start times, which causes delayed start times of v_x and v_y. Because of linearity of VPDF graphs, actors v_x and v_y will not in turn delay v_z and v_z will not in turn delay v_τ. Therefore, a delay caused by v_τ will not delay v_τ.

In Figure 15, linearity and its consequences are illustrated. In Figure 15(a), we show an example schedule with five firings of actor v_x and four firings of actor v_z, where the arrows denote dependencies between firings. In Figure 15(b), the second firing of actor v_z is delayed by $\Delta \geq 0$. In this example, this causes a delay of Δ for the fourth firing of actor v_x. Even though the fourth firing of actor v_z depends on the fourth firing of actor v_x, which is delayed by Δ, we have that the fourth firing of actor v_z is not delayed by firings of actor v_x, because v_z already delayed its own firings by Δ.

Lemma 5 states that under Assumption 1, we have that for any actor that if tokens arrive in time for all its queue-schedules, then tokens also arrive in time for its actor-schedule. This is a nontrivial result, because the actor-schedule has later start times than the queue-schedules, which implies that it produces tokens later on its output queues. Lemma 5 shows that, even though

the dataflow graph is strongly connected and thus has cycles, a $\Delta \geq 0$ later start time leads to a Δ later token production, which leads to token arrival times on the input queues that are maximally delayed by Δ and, therefore, arrive on time for the subsequent firings of this actor, which are delayed by Δ. More specifically, in the graph of Figure 14, we have that the actor-schedule of v_z has later start times than its queue-schedules. We will later show that in fact the queue-schedule on queue $e_{z\tau}$ will delay the other queue-schedules. This lemma shows that, if on queue e_{xz} tokens arrive on time to sustain the nondelayed queue-schedule, then a delay Δ in the production time on e_{zx} will maximally lead to a delay Δ in production times on e_{xz}, leading to arrival times on e_{xz} that are on time for the queue-schedule of v_z on queue e_{xz}, which is also delayed by the actor-schedule of v_z, as illustrated by Figure 15.

LEMMA 5. *Given Assumption 1. Then the following holds:*

$$\forall v_j \in V \bullet \left(\forall e_{ij} \in E(v_j) \bullet \mathit{valid}(\sigma(v_j, e_{ij}))\right) \implies \mathit{valid}(\sigma(v_j)). \tag{25}$$

PROOF. Equation (18) states that $\forall e \in E(v_j) \bullet \sigma(v_j) \geq \sigma(v_j, e)$, that is, actor-schedule $\sigma(v_j)$ has token production times that are later than or equal to any of its queue-schedules $\sigma(v_j, e)$.

We will first show that every path from v_j back to itself starts with a queue e_1 and ends with a queue e_2 that have the same queue-schedule (i.e., $\sigma(v_j, e_1) = \sigma(v_j, e_2)$). Subsequently, we will show that a delay in token productions on e_1 cannot cause tokens to arrive too late on e_2 because both queue-schedules are delayed by the same amount.

Consistency tells us that every simple cycle has an even number of occurrences of a nonconstant parameter p. Given Assumption 1, this means that every directed path from v_j to v_j that starts with a queue $e_1 \in E(p)$ ends with a queue $e_2 \in E(p)$ such that the ratio $\Pi(e_1)/\Gamma(e_2)$ is constant for all parameter values (i.e., $\exists \lambda \in \mathbb{Q} \bullet \Pi(e_1)/\Gamma(e_2) = \lambda$). We have that $\hat{r}(e_2) = \max_{\phi_D(P_D)}(\Gamma(e_2) \cdot z_j / z_\tau \cdot \tau)$ and $\hat{r}(e_1) = \max_{\phi_D(P_D)}(\Pi(e_1) \cdot z_j / z_\tau \cdot \tau)$. Substitution of $\lambda \cdot \Gamma(e_2)$ for $\Pi(e_1)$ in $\hat{r}(e_1)$ results in $\hat{r}(e_1) = \max_{\phi_D(P_D)}(\lambda \cdot \Gamma(e_2) \cdot z_j / z_\tau \cdot \tau) = \lambda \max_{\phi_D(P_D)}(\Gamma(e_2) \cdot z_j / z_\tau \cdot \tau) = \lambda \hat{r}(e_2)$. All schedules constructed for an actor have an equal start time for the first firing of this actor. Because $\Pi(e_1) = \lambda \cdot \Gamma(e_2)$ and $\hat{r}(e_1) = \lambda \cdot \hat{r}(e_2)$, we have $\Pi(e_1)/\hat{r}(e_1) = \Gamma(e_2)/\hat{r}(e_2)$. Lemmas 3 and 4 tell that $\Pi(e_1)/\hat{r}(e_1) = \Gamma(e_2)/\hat{r}(e_2)$ means that in $\sigma(v_j, e_1)$ and $\sigma(v_j, e_2)$ the difference between subsequent firings of j is the same. Because the start time of the first firing in both schedules is the same, we have that both queue-schedules are the same (i.e., $\sigma(v_j, e_1) = \sigma(v_j, e_2)$). This also holds if p has a constant value.

Since a VPDF graph has linear temporal behavior, a nonnegative delay of $\Delta = s(f, \sigma(v_j)) - s(f, \sigma(v_j, e_1)) \geq 0$ in token production times of firing f of v_j on queue e_1 leads to token arrival times on any corresponding queue e_2 that are maximally delayed by Δ. According to Definition 6, if $\mathsf{valid}(\sigma(v_j, e_2))$, then tokens arrive before the token consumption times that follow from schedule $\sigma(v_j, e_2)$. Because $\sigma(v_j, e_1) = \sigma(v_j, e_2)$, token production times following from $\sigma(v_j, e_1)$ and token consumption times following from $\sigma(v_j, e_2)$ are delayed by Δ in the same firing. Since a delay Δ in $\sigma(v_j, e_1)$ causes tokens to arrive on e_2 maximally Δ later where $\sigma(v_j, e_2)$ is already equally delayed by Δ, tokens

arrive in time to enable schedule $\sigma(v_j)$. Since this holds for all pairs e_1 and e_2, this implies that Equation (25) is true. □

Let $d(v_i)$ be the distance from actor v_i to actor v_τ be defined as the number of queues on the simple path from v_i to v_τ in the breadth-first tree as created by a breadth-first search from v_τ [Cormen et al. 2001]. Lemma 6 establishes that queue-schedules on the queues adjacent to an actor v_i that are toward actors v_j and have $d(v_j) \leq d(v_i)$ determine the actor-schedule of v_i. In other words, these queue-schedules dominate the other queue-schedules of an actor. For the example VPDF graph of Figure 14, this implies that the queue-schedule of v_z on $e_{z\tau}$ determines the actor-schedule of v_z.

LEMMA 6. *Given Assumption 1. Then the following holds:*

$$\forall v_i \in V \setminus \{v_\tau\}\ \forall e_{ij} \in E(v_i) \bullet v_i \neq v_j \ \wedge d(v_j) \leq d(v_i) \implies \sigma(v_i) = \sigma(v_i, e_{ij}). \quad (26)$$

PROOF. This follows from the proof of Lemma 10 that is presented in the next section, which includes shared parameters. □

Lemma 7 states that if the queue-schedule of v_i on a queue e_{ij} equals the actor-schedule of v_i, then on queue e_{ij} tokens arrive before they are needed by v_j according to the queue-schedule of v_j on queue e_{ij}. More specifically, in Figure 14, we have that $\mathsf{valid}(\sigma(v_z, e_{z\tau}))$ if $\mathsf{valid}(\sigma(v_z, e_{xz}))$ and $\mathsf{valid}(\sigma(v_z, e_{yz}))$ and $\mathsf{valid}(\sigma(v_z, e_{\tau z}))$ and $\sigma(v_z) = \sigma(v_z, e_{z\tau})$. This means that the schedule on the queue from v_z to v_τ is called valid if tokens arrive in time on all input queues and the schedule on this output queue determines the actor-schedule of v_z. If this is true, then we know that, on queue $e_{z\tau}$, the linear upper bound on token production times is a conservative bound.

LEMMA 7. *The following holds.*

$$\forall e_{ij} \in E \bullet \mathit{valid}(\sigma(v_i, e_{ij})) \implies \mathit{valid}(\sigma(v_j, e_{ij})) \quad (27)$$

PROOF. According to Definition 11, we have that $\mathsf{valid}(\sigma(v_i, e_{ij}))$ is true if and only if $\mathsf{valid}(\sigma(v_i))$ and $\sigma(v_i, e_{ij}) = \sigma(v_i)$. This means that $\mathsf{valid}(\sigma(v_i, e_{ij}))$ is true if, on all input queues of v_i, tokens arrive before they are consumed, and that the actor-schedule of actor v_i is equal to the queue-schedule of v_i on queue e_{ij}. This implies that token production times on e_{ij} are conservatively bounded by the linear upper bound on production times. Furthermore, token consumption times in schedule $\sigma(v_j, e_{ij})$ are conservatively bounded by the linear lower bound on consumption times. Since these bounds are taken into account when computing start times for schedules $\sigma(v_i, e_{ij})$ and $\sigma(v_j, e_{ij})$ in Algorithm 4.5, Equation (27) holds. □

In Theorem 3, the graph is traversed from actors that are furthest away from v_τ to actors that are closer to v_τ. For any actor v_j, we will show that if tokens arrive on time on all input queues of v_j from actors that are closer to v_τ (i.e., Assumption 2 holds), and tokens arrive on time on all input queues of v_j from actors further away from v_τ, then this actor produces its tokens on time on all output queues to actors closer to v_τ. As we traverse the graph, Assumption 2

applies to ever fewer actors until it only applies to v_τ. The proof is concluded by showing that in fact this assumption holds for v_τ.

ASSUMPTION 2. *For actor, v_j holds that on all input queues e_{ij} with $d(v_i) \leq d(v_j)$ holds that* valid$(\sigma(v_j, e_{ij}))$.

THEOREM 3. *Given Assumption 1, then the number of initial tokens as computed by Equation* (16) *is sufficient to let v_τ execute wait-free.*

PROOF. The proof is by structural induction over a breadth-first tree that has actor v_τ as its root.

Base step. Let actor v_i be a leaf of this tree, then given Assumption 2, it holds that $\sigma(v_i)$ is valid. This implies that, on any output queue e of actor v_i, we have valid$(\sigma(v_i, e))$. This is because Lemma 6 states $\sigma(v_i) = \sigma(v_i, e)$.

Induction step. For any actor v_j, if (i) on all queues e_{kj} from actors v_k, with $d(v_k) > d(v_j)$, it holds that valid$(\sigma(v_j, e_{kj}))$, and (ii) on all queues e_{hj} from actors v_h, with $d(v_h) \leq d(v_j)$, it holds that valid$(\sigma(v_h, e_{hj}))$, then valid$(\sigma(v_j))$ holds. With valid$(\sigma(v_j))$ true, we have, by Lemma 6, that on all queues e_{jl}, with $d(v_l) \leq d(v_j)$, valid$(\sigma(v_j, e_{jl}))$.

Together with Lemma 7, this means that starting from the leaves of the breadth-first tree, we can traverse the tree in a breadth-first manner back to v_τ to reach the conclusion that valid$(\sigma(v_\tau))$ given Assumption 2.

However, for actor v_τ, Assumption 2 holds by construction. This is because there are no actors with a smaller than or equal distance, which implies that we only need to check queues from v_τ to itself. We have that $\forall e \in E(v_\tau) \bullet \sigma(v_\tau, e) \leq \sigma(v_\tau)$. Any queue $e_{\tau\tau}$ from v_τ to itself needs, by consistency, to have the same token production and consumption quantum. This implies that the schedule of the token producer (i.e., $\sigma(v_\tau, e_{\tau\tau})$) and the schedule of the token consumer (i.e., $\sigma(v_\tau, e_{\tau\tau})$) are delayed by the same delay $\Delta \geq 0$, which implies that tokens arrive in time. □

4.8.2 *Shared Parameters.* In this section, Theorem 4 will show that Theorem 3 also holds without Assumption 1. This is done by adding a clause to the premise of Lemma 6 enabling us to remove Assumption 1, resulting in Lemma 10. The proof of Lemma 10 is based on the results of Lemmas 8 and 9, which will also support the proof of Lemma 14. Subsequently, Lemma 11 shows that given a valid actor-schedule the removal of Assumption 1 does not affect the validity of this actor-schedule. These two lemmas are used in the proof of Theorem 4 to show that parts of a VPDF graph cannot influence the schedule of v_τ. We will show that, for example, for the graph shown in Figure 16, variation in the value of parameter p does not influence the actor-schedules of actors v_i and v_j, and since by construction of VPDF graphs, actor v_τ is not in G'_p, variation in p does not influence the schedule of v_τ. This will imply that a VPDF graph can be reduced in Figure 16 through removal of G'_p to a VPDF graph for which Assumption 1 holds, implying that Theorem 3 holds.

Lemmas 8 and 9 will enable us to show in Lemma 14 that the queue-schedule on queues from an actor v_i that satisfy the following property determine the actor-schedule of this actor v_i.

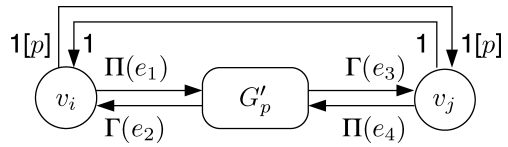


Fig. 16. VPDF graph with shared parameter p, where $\Pi(e_1)$, $\Gamma(e_2)$, $\Gamma(e_3)$, and $\Pi(e_4)$ are parameterized in p.

Definition 13. The property $D(e_{ij})$ is true if queue e_{ij} is from v_i to v_j, where these actors are different and actor v_j has a smaller than or equal distance to v_τ than actor v_i and the parameterized cumulative production quantum on e_{ij} is not parameterized in a shared parameter $p \in S$.

$$D(e_{ij}) \Leftrightarrow v_i \neq v_j \wedge d(v_j) \leq d(v_i) \wedge (\nexists p \in S \bullet \Pi(e_{ij}) \approx p)$$

LEMMA 8. *The following holds:*

$$\forall v_i \in V \setminus \{v_\tau\}\ \forall e_{ij}, e_{ik} \in E(v_i) \bullet D(e_{ij}) \implies \frac{\hat{r}(e_{ik})}{\hat{r}(e_{ij})} = \max_{\phi_D(P_D)} \left(\frac{\Pi(e_{ik})}{\Pi(e_{ij})} \right), \quad (28)$$

where e_{ik} is an output queue of v_i.

PROOF. By definition, $\hat{r}(e_{ik}) = \max_{\phi_D(P_D)} ({}^{\Pi(e_{ik}) \cdot z_i}/_{z_\tau \cdot \tau})$. By Lemma 1, we have ${}^{z_i}/_{z_j} = {}^{\Gamma(e_{ij})}/_{\Pi(e_{ij})}$. After substitution of ${}^{\Gamma(e_{ij}) \cdot z_j}/_{\Pi(e_{ij})}$ for z_i in $\max_{\phi_D(P_D)} ({}^{\Pi(e_{ik}) \cdot z_i}/_{z_\tau \cdot \tau})$, we have $\hat{r}(e_{ik}) = \max_{\phi_D(P_D)} ({}^{\Pi(e_{ik}) \cdot \Gamma(e_{ij}) \cdot z_j}/_{\Pi(e_{ij}) \cdot z_\tau \cdot \tau})$.

We have that $\Pi(e_{ik})$ and $\Pi(e_{ij})$ are parameterized in different parameters than ${}^{z_j}/_{z_\tau}$. This is because, by Lemma 1, ${}^{z_j}/_{z_\tau}$ is equal to a ratio of cumulative transfer quanta on a directed simple path from v_j to v_τ. Because $v_i \neq v_j$ and because of the restrictions on sharing of parameters, none of these cumulative transfer quanta is parameterized in a parameter of $\Pi(e_{ik})$ or $\Pi(e_{ij})$.

Furthermore, we have that $\Pi(e_{ik})$ and $\Pi(e_{ij})$ are parameterized in different parameters than τ because v_τ is not adjacent to a queue that communicates a parameter value.

Also $\Pi(e_{ik})$ and $\Pi(e_{ij})$ do not share a parameter with $\Gamma(e_{ij})$. For $\Pi(e_{ij})$, this is by definition of this queue. For $\Pi(e_{ik})$, suppose that $\Pi(e_{ik})$ shares a parameter with $\Gamma(e_{ij})$. Then, by the restrictions on sharing of parameters, $z_i = z_j$, and because $\Pi(e_{ij}) \cdot z_i = \Gamma(e_{ij}) \cdot z_j$ holds by consistency of the graph, we now have that $\Pi(e_{ij}) = \Gamma(e_{ij})$. However, by definition of e_{ij}, $\Pi(e_{ij})$ is not parameterized in a shared parameter, therefore, $\Gamma(e_{ij})$ cannot share a parameter with $\Pi(e_{ik})$.

Because $\Pi(e_{ij})$ and $\Pi(e_{ik})$ are parameterized in different parameters than ${}^{\Gamma(e_{ij}) \cdot z_j}/_{z_\tau \cdot \tau}$, we have $\hat{r}(e_{ik}) = \max_{\phi_D(P_D)} ({}^{\Pi(e_{ik}) \cdot \Gamma(e_{ij}) \cdot z_j}/_{\Pi(e_{ij}) \cdot z_\tau \cdot \tau}) = \max_{\phi_D(P_D)} ({}^{\Pi(e_{ik})}/_{\Pi(e_{ij})}) \cdot \max_{\phi_D(P_D)} ({}^{\Gamma(e_{ij}) \cdot z_j}/_{z_\tau \cdot \tau}) = \max_{\phi_D(P_D)} ({}^{\Pi(e_{ik})}/_{\Pi(e_{ij})}) \cdot \hat{r}(e_{ij})$. □

LEMMA 9. *The following holds:*

$$\forall v_i \in V \setminus \{v_\tau\}\ \forall e_{ij}, e_{hi} \in E(v_i) \bullet D(e_{ij}) \implies \frac{\hat{r}(e_{hi})}{\hat{r}(e_{ij})} = \max_{\phi_D(P_D)} \left(\frac{\Gamma(e_{hi})}{\Pi(e_{ij})} \right), \quad (29)$$

where e_{hi} is an input queue of v_i.

PROOF. This proof is analogous to the proof of Lemma 8. □

Lemma 10 states that the queue-schedules on the queues toward actor v_τ that are without shared parameters dominate the other queue-schedules of an actor.

LEMMA 10. *The following holds:*

$$\forall v_i \in V \setminus \{v_\tau\} \quad \forall e_{ij} \in E(v_i) \bullet D(e_{ij}) \quad \Longrightarrow \quad \sigma(v_i) = \sigma(v_i, e_{ij}). \quad (30)$$

PROOF. Let e_{ik} be an output queue of v_i and let e_{hi} be an input queue of v_i. In order to show that the queue-schedule of e_{ij} determines the actor-schedule of v_i, we need to show that the difference between subsequent start times of the queue-schedule on e_{ij} is for every firing larger than or equal to the difference in subsequent start times in the queue-schedules on e_{ik} and e_{hi}. According to Lemmas 3 and 4, this requires us to show that $\Pi(e_{ij})/\hat{r}(e_{ij}) \geq \Pi(e_{ik})/\hat{r}(e_{ik})$ and $\Pi(e_{ij})/\hat{r}(e_{ij}) \geq \Gamma(e_{hi})/\hat{r}(e_{hi})$, or, equivalently, we need to show that $\hat{r}(e_{ik})/\hat{r}(e_{ij}) \geq \Pi(e_{ik})/\Pi(e_{ij})$ and $\hat{r}(e_{hi})/\hat{r}(e_{ij}) \geq \Gamma(e_{hi})/\Pi(e_{ij})$. By Lemmas 8 and 9, respectively, these conditions are satisfied. □

Lemma 11 shows that if the actor-schedule is valid for the maximum value of a shared parameter, the validity of this actor-schedule is independent of the value of this shared parameter.

LEMMA 11. *Given a parameter $p \in S$. If for each $v_i \in V$ with $p \in P_D(v_i)$ holds that valid$(\sigma(v_i))$ for $\hat{p}$, then valid$(\sigma(v_i))$ for all values of p.*

PROOF. The fact $p \in S$ implies that, for every v_i with $p \in P_D(v_i)$, there is an $v_j \neq v_i$ with $p \in P_D(v_j)$. Furthermore, every path from v_i to v_j that starts with an output queue e_1 of v_i that has $\Pi(e_1)$ parameterized in p includes an input queue e_2 of v_j that has $\Gamma(e_2)$ that is parameterized in p. By consistency of the graph, we have that $\exists \lambda \in \mathbb{Q} \bullet \Pi(e_1) = \lambda \cdot \Gamma(e_2)$. Lemmas 3 and 4 tell us that $\sigma(v_i, e_1) = \sigma(v_j, e_2)$.

Since every actor has a queue that is toward v_τ, Lemma 10 tells us that $\sigma(v_i)$ has later start times than queue-schedule $\sigma(v_i, e_1)$ in case of $\hat{p}$ by $\hat{\Delta}_i \geq 0$ and that $\sigma(v_j)$ has later start times than the queue-schedule $\sigma(v_j, e_2)$ in case of $\hat{p}$ by $\hat{\Delta}_j \geq 0$. For smaller parameter values than $\hat{p}$, these queue-schedules have a smaller difference in subsequent start times by Lemmas 3 and 4 and, therefore, have a larger difference in the start time of a particular firing compared to the start time in the actor-schedule. Since these queue-schedules are the same, this additional difference with the actor-schedule $\Delta \geq 0$ is the same for both schedules. Given that valid$(\sigma(v_i))$ and valid$(\sigma(v_j))$ for $\hat{p}$, linearity of VPDF graphs tells us that a Δ later token production on e_1 cannot lead to token arrival times that are delayed by more than Δ. Therefore, tokens arrive on time on queue e_2 because the consumption time according to the actor-schedule of v_j on e_2 was also delayed by Δ compared to the consumption time according to the queue-schedule of v_j on e_2. □

For any shared parameter p, the following theorem shows that the schedules of the actors in the subgraph G'_p, see for example Figure 16, do not influence the schedule of v_τ. This implies that these subgraphs can be removed from the

graph, resulting in a graph that does not have shared parameters. This implies that the assumption under which Theorem 3 holds is valid.

THEOREM 4. *The number of initial tokens as computed by Equation* (16) *is sufficient to let* v_τ *execute wait-free.*

PROOF. Theorem 3 shows that this theorem is true given Assumption 1. However, Lemma 10 states that for each actor, the actor-schedule equals the queue-schedule on a queue toward v_τ that has a cumulative token production quantum that is not parameterized in a shared parameter. By construction of VPDF graphs, each actor has such a queue. Further, Lemma 11 states that the value of shared parameters does not affect the validity of the actor-schedules of the actors that have cumulative token transfer quanta parameterized in these shared parameters. Together, this implies that for every actor with cumulative transfer quanta parameterized in shared parameters that the queue-schedules on these queues do not influence the actor-schedule. For every shared parameter, we can remove all queues with cumulative transfer quanta parameterized in this parameters. This disconnects a subgraph from the VPDF graph. By construction of VPDF graphs, this subgraph does not include v_τ, and we have just established that it does not affect the actor-schedules of the actors in the remaining graph. After disconnecting all these subgraphs, we have a remaining VPDF graph that does not have shared parameters. This means that Assumption 1 is irrelevant for the remaining VPDF graph that still includes v_τ. Since removal of these subgraphs did not change the actor-schedules of the remaining VPDF graph, this theorem holds because Theorem 3 holds. □

4.8.3 *Computed Number of Tokens and Required Buffer Capacity.* Theorem 4 established that the computed number of tokens is sufficient to let actor v_τ execute wait-free. The following theorem shows that this number of initial tokens corresponds with the required buffer capacity to satisfy the throughput constraint.

THEOREM 5. *The number of initial tokens as computed by Equation* (16) *is a sufficient buffer capacity to guarantee that the throughput constraint is satisfied.*

PROOF. There is a one-to-one correspondence between the number of tokens consumed and produced by subsequent firings of an actor and the number of containers consumed and produced by subsequent executions of a task. Aggregate firings are only enabled later and produce tokens later then normal firings. Actor firings are only enabled later and produce tokens later than task executions are enabled and produce their containers. By monotonicity of VPDF graphs, we have the following. A VPDF graph with normal firings does not have later token arrival times than a VPDF graph with aggregate firings. The task graph does not have later container arrival times than the token arrival times in the VPDF graph with normal firings. This implies that token arrival times in the VPDF graph with aggregate firings are conservative container arrival times. Since task executions are enabled by container arrivals, if the number of tokens is sufficient to let v_τ execute wait-free, then the corresponding buffer

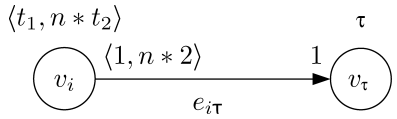


Fig. 17. VPDF graph with $n \in [0, \infty)$, that is, the second phase has an unbounded number of firings.

capacities are sufficient to let the sink, or source, of the task graph execute wait-free. Therefore, by Theorem 4 this theorem holds. □

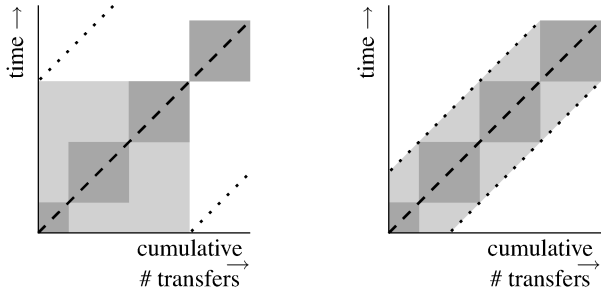
5. UNBOUNDED ITERATION

From the definition of an aggregate firing in Section 4.2, it is clear that the production time of tokens depends on the number of iterations (i.e., the more iterations the later an aggregate firing produces its tokens). An unbounded number of iterations leads with this approach to an unbounded production time and in the end to unbounded buffer capacities. Apart from a small change in the determination of the maximum required rate and feasibility, this section proposes a different definition of aggregate firings and discusses its consequences. The proposed definition of aggregate firings leads to more accurate results in case of bounded parameters and allows for unbounded firing parameters. An example graph with unbounded firing parameters is shown in Figure 17. In this figure and subsequent figures in this section, we omitted the queue from actor v_τ to actor v_i for reasons of clarity.

5.1 Outline of the Approach

In Section 4.2, we defined a rectangular aggregate firing. This rectangular aggregate firing abstracts a sequence of normal firings and consumes all tokens consumed by this sequence of normal firings at its start and produces all tokens produced by this sequence of normal firings at its finish. As remarked upon in Section 4.2, this abstraction leads to a loss in accuracy. However, when we want to analyze VPDF graphs with an unbounded number of firings per phase, then this abstraction is no longer applicable, because it leads to the requirement of unbounded buffer capacities. Figure 17 shows an example VPDF graph with an actor that has a phase with an unbounded number of firings.

In Figure 18(a), we show a sequence of four normal firings of actor v_i from Figure 17, where v_i has a firing duration of $t_1 = \tau$ for its first phase and a firing duration of $t_2 = 2\tau$ for firings of its second phase. The rectangular aggregate firing is not applicable for this actor because the production time of the first token is increased with an increasing number of firings of the second phase of actor v_i. Figure 18(a) shows, as an example, the aggregate firing corresponding with two firings of the second phase and the upper bound on token production times and lower bound on token consumption times that correspond with this rectangular aggregate firing. In this example, we have that with rectangular aggregate firings, the distance between these two bounds increases with an increasing number of firings of the second phase. In this section, we redefine aggregate firings such that they no longer have a rectangular token transfer



(a) rectangular aggregate firing (b) new aggregate firing

Fig. 18. Schedule of firings of actor v_i from Figure 17, with $t_1 = \tau$ and $t_2 = 2\tau$. The aggregate firing is depicted with light grey.

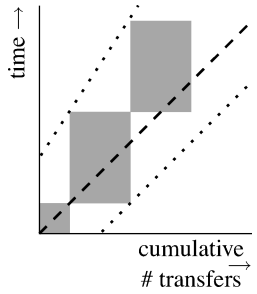
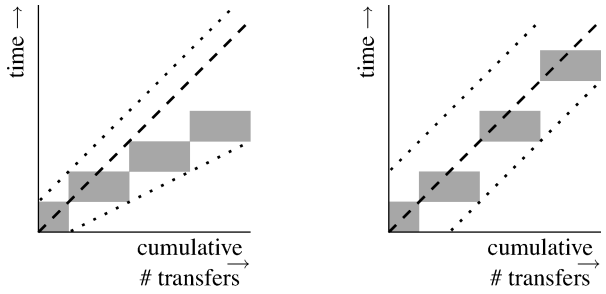


Fig. 19. Schedule of firings of actor v_i from Figure 17, with $t_1 = \tau$ and $t_2 = 3\tau$.

shape but instead a shape, as shown in Figure 18(b). This new shape grows along the required token transfer rate, resulting in a difference between the upper bound on token production times and a lower bound on token consumption times that no longer grows with the number of firings, in case there is no upper bound specified on this number of firings.

In Figure 18, we constructed a schedule of normal firings in which every firing starts immediately after the previous firing finished. This is the schedule of firings that was used to construct the rectangular aggregrate firings in Section 4.2. In Section 4.4, we delayed aggregate firings to let the token transfer rate of the constructed schedule match the required token transfer rate. However, if we construct this schedule of normal firings, then an unbounded number of firings of a phase can lead to an unbounded difference between the token transfer rate of this constructed schedule and the required token transfer rate. The VPDF graph shown in Figure 17 with $t_1 = \tau$ and $t_2 = 3\tau$ is already rejected by our check on the firing durations of actors, because with these firing durations actor v_i from Figure 17 cannot transfer tokens at a sufficient rate, as illustrated by the schedule in Figure 19.

However, the VPDF graph shown in Figure 17 with $t_1 = \tau$ and $t_2 = \tau$ is allowed. The schedule in Figure 20(a) starts a firing immediately after the previous has finished, which, in this case, results in a token transfer rate that is higher than the required rate. This schedule will result in an unbounded required buffer capacity.



(a) undelayed firings (b) delayed firings

Fig. 20. Schedule of firings of actor v_i from Figure 17, with $t_1 = \tau$ and $t_2 = \tau$.

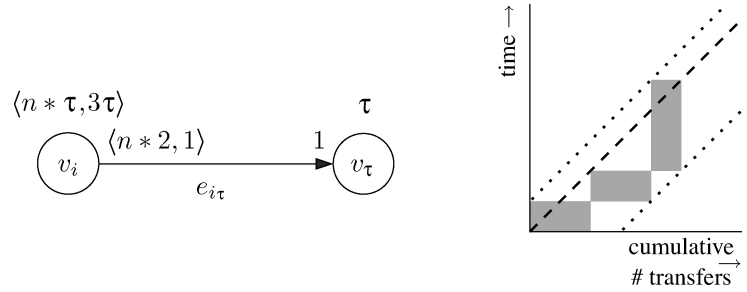
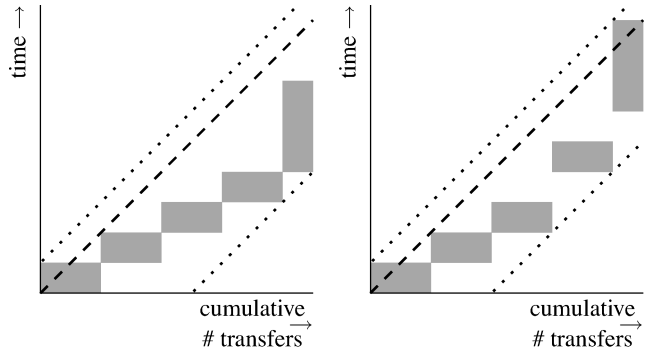


Fig. 21. VPDF graph with $n \in [2, \infty)$ and a schedule of firings of actor v_i for $n = 2$.

The schedule of normal firings that we will construct in this section will delay firings of phases with no upper bound on the number of firings to adapt the schedule of these firings to the required token transfer rate. Our adaptation of the schedule from Figure 20(a) results in the schedule as shown in Figure 20(b), in which the second and third firing of the second phase of v_i from Figure 17 are delayed.

However, the introduction of delay in the schedule of normal firings is non-trivial. This is because the firing durations of different phases can compensate each other, as, for example, actor v_i in Figure 21. This actor v_i requires n to have a lower bound of 2 in order to transfer tokens at a sufficient rate. The first two firings of the first phase of this actor cannot be delayed, otherwise the rate would be insufficient. However, as shown in Figure 22(a), if subsequent firings of the first phase are not delayed, then the rate will diverge from the required token transfer rate. In Figure 22(b), it is shown where we will introduce the required delay to match the token transfer rate of the constructed schedule to the required token transfer rate.

The basic idea is that the schedule of firings within an aggregate firing is constructed by starting with a reference schedule in which the phases with no upper bound on their number of firings have a minimal number of firings. In this reference schedule, we will insert the omitted firings of phases with no upper bound on their number of firings. The insertion of a firing will have as consequence that the subsequent firings in this schedule are translated over the maximum required token transfer rate. This process is illustrated by



(a) undelayed firings (b) delayed firings

Fig. 22. Schedules for firings of actor v_i from Figure 21, with $n = 4$.

Figures 21(b) and 22(b), where the schedule of firings shown in Figure 21(b) forms the reference schedule. In the schedule of Figure 22(b), two additional firings of the first phase have been inserted and the firing of the second phase is translated over the maximum required transfer rate compared to its position in the token transfer curve of Figure 21(b).

In Section 4, schedules of aggregate firings were determined per queue. The schedule of firings within aggregate firings was the same on every adjacent queue, that is, a firing within an aggregate firing always started immediately after the previous firing within the same aggregate firing finished. The proof of Theorem 5 depends on lemmas that show that combining these schedules of aggregate firings per queue to a schedule of aggregate firings per actor leads to the situation that the schedules on queues toward v_τ determine the schedule of the actor. This means that the start times of aggregate firings in the constructed schedules on the queues toward v_τ are later than the start times in the constructed schedules on other adjacent queues. The two main differences between the approach in this section and our approach of Section 4 are that we no longer abstract firings to rectangular aggregate firings and that we now introduce delay in the schedule of firings within an aggregate firing. The first main difference implies that we need to derive new bounds on token transfer times, given the new aggregate firings. The second main difference implies that we need to extend our previous reasoning and show that the schedule of firings in queue-schedules of queues toward v_τ has later start times than in queue-schedules of other adjacent queues. This is because we delay firings such that the schedule of firings has a transfer rate that matches the maximum required transfer rate. Therefore, on different adjacent queues firings are delayed by different amounts. We will show that firings are delayed most in the constructed queue-schedules of queues toward v_τ.

In this section, we will first adapt the determination of the maximum required rate in Section 5.2. Subsequently, we will present the new definition of aggregate firings in Section 5.3. Given this definition of aggregate firings, we derive new expressions for the linear bounds on token transfer times in Section 5.4. Section 5.5 shows that the number of initial tokens computed in

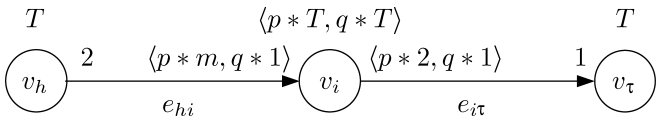


Fig. 23. Example VPDF graph, with $p, q \in [1, \infty)$ and $m \in [1, 3]$.

Section 5.4 corresponds with buffer capacities that are sufficient to let task w_τ execute wait-free (i.e., Section 5.5 shows that Theorem 5 still holds).

5.2 Maximum Required Token Transfer Rate

The determination of the maximum required rate as described by Theorem 1 in Section 4.3 is found by considering all combinations of extreme values that parameters can attain. In case a parameter does not have an upper bound, then there is no maximum value. We adapt the procedure from Theorem 1 as follows. For a queue e_{ij}, we consider all combinations of extreme values of the parameters in $\Pi(e_{ij}) \cdot z_i / z_\tau \cdot \tau$, except that for parameters with no upper bound we take the limit to positive infinity. This limit can be determined relatively straightforwardly. This is because parameters with no upper bound are local to a single actor, while further in Lemma 2, it is shown that $\Pi(e_{ij}) \cdot z_i / z_\tau \cdot \tau$ equals a product of factors, where each factor is a ratio containing only cumulative transfer quanta and cumulative firing duration of a single actor. This implies that the limit to positive infinity of multiple parameters that are associated with different actors is the product of the limit to positive infinity of the ratio in this product that corresponds with this actor. Because of the sequential nature of the phases of an actor, we do not need to consider the limit to positive infinity of multiple parameters of the same actor.

For example, for the VPDF graph from Figure 23, the maximum required transfer rates are determined as follows. We have symbolic repetition rates $z_h = p \cdot m + q$, $z_i = 2$, and $z_\tau = 4p + 2q$. On queue e_{hi}, we have $\Pi(e_{hi}) \cdot z_h / z_\tau \cdot \tau = 2 \cdot (p \cdot m + q) / (4p + 2q) \cdot \tau = p \cdot m + q / (2p + q) \cdot \tau$. Since p and q have no upper bound and belong to the same actor, we consider the following combinations of extreme values: (i) m, p, q minimal, (ii) m minimal and limit to positive infinity in p, (iii) m minimal and limit to positive infinity in q, (iv) m maximal and p, q minimal, (v) m maximal and limit to positive infinity in p, and (vi) m maximal and limit to positive infinity in q. Taking the minimal value for p and q, we obtain $1 \cdot m + 1 / (2 \cdot 1 + 1) \cdot \tau = m + 1 / 3\tau$, which means case (i) equals a rate of $2/3\tau$, and case (iv) equals a rate of $4/3\tau$. Evaluating $\lim_{p \to \infty} p \cdot m + q / (2p + q) \cdot \tau$ and $\lim_{p \to \infty} p \cdot m + q / (2p + q) \cdot \tau$, we obtain $m/2\tau$ and $1/\tau$, respectively. This implies that case (ii) has a rate of $1/2\tau$, case (v) has a rate of $3/2\tau$, and cases (iii) and (vi) have a rate of $1/\tau$. The maximum required rate on queue e_{hi} is, therefore, $\hat{r}(e_{hi}) = \max(\{2/3\tau, 4/3\tau, 1/2\tau, 3/2\tau, 1/\tau\}) = 3/2\tau$. The maximum required transfer rate on $e_{i\tau}$ is $\hat{r}(e_{i\tau}) = 1/\tau$.

The procedure from Theorem 2 to verify that the firing durations enable the satisfaction of the throughput constraint is extended similarly. For actor v_h, we have that $\max_{\phi_D(P_D)}(\Upsilon(v_h) \cdot z_h / z_\tau \cdot \tau) \leq 1$ holds, because we have that $\max_{\phi_D(P_D)}(\Upsilon(v_h) \cdot z_h / z_\tau \cdot \tau) = \max_{\phi_D(P_D)}(\tau \cdot (p \cdot m + q) / (4p + 2q) \cdot \tau) = \max_{\phi_D(P_D)}(p \cdot m + q / 4p + 2q)$,

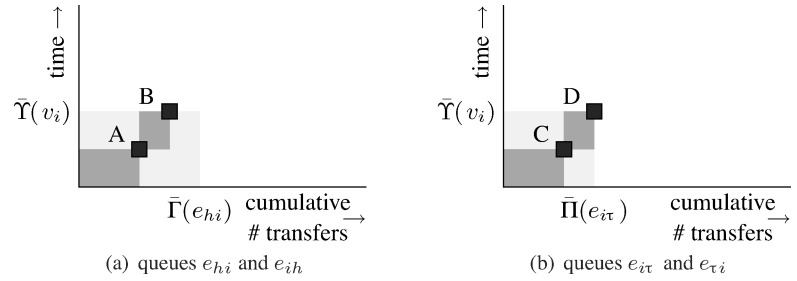
and evaluation of the extreme values of these parameters results in $\max(\{1/3, 1/4, 1/2, 2/3, 3/4\}) = 3/4 \leq 1$.

5.3 Aggregate Firing

As outlined in Section 5.1, our basic idea is to have aggregate firings that grow along the maximum required token transfer rate line instead of having rectangular aggregate firings. The schedule of firings within these new aggregate firings is constructed by first considering a reference schedule and then inserting additional firings. The reference schedule only schedules a minimum number of firings of the phases that have no upper bound on their number of firings. Say that the parameters of actor v_i in aggregate firing f have parameter values $P(v_i, f)$. Given the values $P(v_i, f)$, we derive the parameter values $P_0(v_i, f)$ in which the parameters that specify the number of firings of phases that have no upper bound on their number of firings obtain their minimal value. Our check on the firing durations of an actor ensures that for all possible parameter values the firing duration allows the throughput constraint to be satisfied. For the phases that can have an unbounded number of firings, this implies that these phases have a token transfer rate that is higher than or equal to the maximum required token transfer rate.

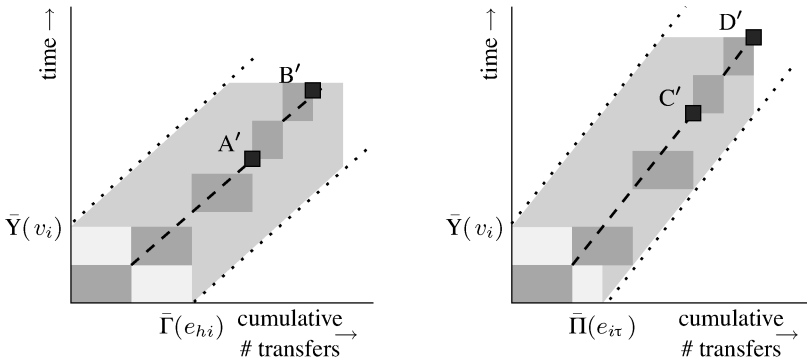
Consider, for example, aggregate firing f of actor v_i from Figure 21 with $n = 4$. To obtain $P_0(v_i, f)$, we minimize the number of firings of the first phase, since this phase does not have an upper bound on its number of firings. Therefore, in our reference parameter valuation $P_0(v_i, f)$, we have that $n = 2$. Given this reference parameter valuation, we construct a reference schedule in which each firing in the aggregate firing starts immediately after the previous firing in this aggregate firing has finished. This is the schedule shown in Figure 21. To obtain the schedule of firings within aggregate firing f, we insert firings in the reference schedule. We insert firings in this reference schedule that are present given parameter values $P(v_i, f)$, but that are not present given parameter values $P_0(v_i, f)$. In the reference schedule as shown in Figure 21, we insert the third firing of the first phase, which is present given $P(v_i, f)$, in which $n = 3$, but not present given $P_0(v_i, f)$, in which $n = 2$. Because we insert a firing that has a token transfer rate that is higher than the maximum required token transfer rate, we can insert this firing and translate subsequent firings over the maximum required transfer rate line, because this will only delay firings. This is shown in Figure 22(b), where the third firing of the first phase is inserted and the subsequent firing is translated over the maximum required transfer rate line compared to its position in the reference schedule from Figure 21. In the resulting schedule, as shown in Figure 22(b), no longer every firing starts immediately after the previous has finished. Instead delay is introduced in the schedule of firings within an aggregate firing.

To illustrate the construction of the schedules more precisely, our running example is actor v_i from Figure 23 with parameter values $P(v_i, f)$ equal to $p = 3$, $q = 2$, and $m = 2$. Since p and q could have no upper bound, we take the minimal value of p and q, which results in $P_0(v_i, f)$ equal to $p = 1$, $q = 1$, and $m = 2$. The reference schedule on queues e_{hi} and $e_{i\tau}$ as constructed using



(a) queues e_{hi} and e_{ih} (b) queues $e_{i\tau}$ and $e_{\tau i}$

Fig. 24. Schedule of firings for parameter valuation $P_0(v_i, f)$.



(a) queues e_{hi} and e_{ih} (b) queues $e_{i\tau}$ and $e_{\tau i}$

Fig. 25. Increased number of transferred tokens leads to start times that are shifted along required rate.

$P_0(v_i, f)$ is shown in Figure 24, while the schedule for $P(v_i, f)$ on these queues is shown in Figure 25.

In order to define this reference schedule more precisely, we define $\Phi(n, f)$ as the sum of firing durations of firing 1 up to and including firing n of aggregate firing f. We define $\Phi_0(n, f)$ as $\Phi(n, f)$; however, in $\Phi_0(n, f)$, we exclude firings that are not present given parameter valuation $P_0(v_i, f)$. This means that we exclude firings of phases with an unbounded number of firings, if these firings are additional to the minimum number of firings of that phase. More specifically, let a phase h have a firing parameter p (i.e., $\chi_D(v_i, h) = p$). Let p have value p_0 given parameter valuation $P_0(v_i, f)$, which we denote by $[\chi_D(v_i, h)]_{P_0(v_i,f)} = p_0$. In $\Phi_0(n, f)$, we exclude the firing duration of any firing $q > p_0$ of this phase h. We define $\Phi_0(n, f)$ by Equation (31), given that firing n in aggregate firing f is the q-th firing of phase h. In Equation (31), we have a term that corresponds with the cumulative firing duration of the current phase h and a term that corresponds with the previous phases. For the current phase, we exclude the firing duration of any firing after p_0. For the previous phases, we multiply the number of firings given parameter valuation $P_0(v_i, f)$ with the

firing durations of these firings.

$$\Phi_0(n,\, f) = \min(q, [\chi_D(v_i, h)]_{P_0(v_i, f)}) \cdot \rho(v_i, h) + \sum_{k=1}^{k=h-1} [\chi_D(v_i, k)]_{P_0(v_i, f)} \cdot \rho(v_i, k) \quad (31)$$

With $s(f, \sigma(v_i, e))$ equal to the start of aggregate firing f on queue e in the constructed schedule of aggregate firings $\sigma(v_i, e)$, we define the reference start time of firing n of f on e, $s_0(n,\, f, e)$, by Equation (32).

$$s_0(n,\, f, e) = s(f, \sigma(v_i, e)) + \Phi_0(n-1,\, f) \quad (32)$$

In our example, we have $P(v_i,\, f)$ for actor v_i from Figure 23 equal to $p = 3$, $q = 2$, and $m = 2$. For the five firings that follow from the parameter values $P(v_i,\, f)$, we have reference start times $s_0(1,\, f, e_{hi}) = 0$, $s_0(2,\, f, e_{hi}) = \tau$, $s_0(3,\, f, e_{hi}) = \tau$, $s_0(4,\, f, e_{hi}) = \tau$, $s_0(5,\, f, e_{hi}) = 2\tau$ on queue e_{hi} and the same reference start times on queue $e_{i\tau}$ all relative to the start time of aggregate firing f. This reference schedule is shown in Figure 24. In such a reference schedule, multiple firings can be assigned the same start time. For v_i from our example, firings 2, 3, and 4 all have the same start time denoted A in Figure 24(a) and denoted C in Figure 24(b). Times A and C will be the reference points, $s_0(n,\, f, e)$, to define the start times of firings 2, 3, and 4 that will be inserted in the reference schedule to obtain the schedules shown in Figure 25.

Given this reference schedule of firings, the schedule of firings of aggregate firing f on queue e is constructed as follows. Let firings 1 up to and including firing $n-1$ transfer y tokens on queue e, given parameter valuation $P_0(v_i,\, f)$. Let these firings transfer $y+z$ tokens on e, given parameter valuation $P(v_i,\, f)$. By construction of $P_0(v_i,\, f)$, we have that $z \geq 0$. We define the start time of firing n in aggregate firing f on queue e to be $s(n,\, f, e) = s_0(n,\, f, e) + {}^{z}/_{\hat{r}(e)}$.

To define the schedule precisely, we first differentiate between a schedule on an output queue e_1 of v_i and a schedule on an input queue e_2 of actor v_i. To define the schedule on output queue e_1, we define $\Xi(n,\, f, e_1)$ as the sum of tokens produced in firing 1 up to and including firing n of aggregate firing f on queue e_1. As $\Phi_0(n,\, f)$ excluded firings not present given parameter valuation $P_0(v_i,\, f)$, $\Xi_0(n,\, f, e_1)$ excludes tokens not produced given parameter valuation $P_0(v_i,\, f)$. Given that firing n of aggregate firing f is a firing of phase h, we define $\Xi_0(n,\, f, e_1)$ by Equation (33).

$$\Xi_0(n,\, f, e_1) = \min(n, [\chi_D(v_i, h)]_{P_0(v_i, f)}) \cdot [\pi(e_1, h)]_{P_0(v_i, f)} + \sum_{k=1}^{k=h-1} [\chi_D(v_i, k)]_{P_0(v_i, f)} \cdot [\pi(e_1, k)]_{P_0(v_i, f)} \quad (33)$$

The start time of firing n of aggregate firing f on output queue e_1 of v_i is defined by Equation (34). This start time is determined by delaying its reference start time linearly in the number of additional tokens produced by previous firings. The number of additional tokens is determined by the difference in the

number of tokens produced given valuation $P(v_i, f)$ and the number of tokens produced in case of valuation $P_0(v_i, f)$. This difference in number of produced tokens can be caused by an increased number of firings or by an increase in token production quanta.

$$s(n, f, e_1) = s_0(n, f, e_1) + \frac{\Xi(n-1, f, e_1) - \Xi_0(n-1, f, e_1)}{\hat{r}(e_1)} \tag{34}$$

The schedule of aggregate firing f of v_i on input queue e_2 is defined analogously to the schedule on output queue e_1. To define the schedule on input queue e_2, we define $\Lambda(n, f, e_2)$ as the sum of tokens produced by firings 1 up to and including firing n of aggregate firing f. As $\Xi_0(n, f, e_1)$ excluded tokens not produced given parameter valuation $P_0(v_i, f)$, $\Lambda_0(n, f, e_2)$ excludes tokens not consumed given parameter valuation $P_0(v_i, f)$. Given that firing n is a firing of phase h, $\Lambda_0(n, f, e_2)$ is defined by Equation (35).

$$\Lambda_0(n, f, e_2) = \min(n, [\chi_D(v_i, h)]_{P_0(v_i, f)}) \cdot [\gamma(e_2, h)]_{P_0(v_i, f)} + \sum_{k=1}^{k=h-1} [\chi_D(v_i, k)]_{P_0(v_i, f)} \cdot [\gamma(e_2, k)]_{P_0(v_i, f)} \tag{35}$$

The start time of firing n of aggregate firing f on input queue e_2 of v_i is defined by Equation (36).

$$s(n, f, e_2) = s_0(n, f, e_2) + \frac{\Lambda(n-1, f, e_2) - \Lambda_0(n-1, f, e_2)}{\hat{r}(e_2)} \tag{36}$$

As shown in Figure 25, for the five firings that follow from valuation $P(v_i, f)$, we have the following start times. The first firing was already present in the reference schedule, $s(1, f, e_{i\tau}) = s_0(1, f, e_{i\tau}) = 0$. The second firing is inserted in the reference schedule, but because it is the first inserted, firing there are no additional tokens transferred by previous firings, $s(2, f, e_{i\tau}) = s_0(2, f, e_{i\tau}) + {}^0/_{\hat{r}(e_{i\tau})} = \tau + {}^0/_{\hat{r}(e_{i\tau})} = \tau$. For the third firing, we have that the previous firings transferred two additional tokens, $s(3, f, e_{i\tau}) = s_0(3, f, e_{i\tau}) + {}^2/_{\hat{r}(e_{i\tau})} = \tau + {}^2/_{\hat{r}(e_{i\tau})} = 3\tau$. The fourth firing was already present in the reference schedule, but its start time is translated because previous firings transferred four additional tokens, $s(4, f, e_{i\tau}) = s_0(4, f, e_{i\tau}) + {}^4/_{\hat{r}(e_{i\tau})} = \tau + {}^4/_{\hat{r}(e_{i\tau})} = 5\tau$. For the fifth firing, we have that the previous firings transferred four additional tokens, $s(5, f, e_{i\tau}) = s_0(5, f, e_{i\tau}) + {}^4/_{\hat{r}(e_{i\tau})} = 2\tau + {}^4/_{\hat{r}(e_{i\tau})} = 6\tau$. This schedule is shown in Figure 25(b). The schedule on queues e_{hi} and e_{ih} is shown in Figure 25(a). As illustrated in these figures, the start times of firings 2, 3, and 4 on queue e_{hi} are derived from point A of Figure 24(a) by a translation over a line with a slope $\hat{r}(e_{hi})$. On queue $e_{i\tau}$, these start times are derived from C by a translation over a line with a slope $\hat{r}(e_{i\tau})$.

This example, as shown in Figure 25, illustrates the main difference in this definition of aggregate firing and the definition from Section 4.2, which is that this definition already delays firings within an aggregate firing. We no longer have an aggregate phase that consumes $\Gamma(f, e_1)$ tokens at its start and produces

$\Pi(f, e_2)$ tokens $\Upsilon(v_i, f)$ later. Instead, we bound the token transfer times of the just defined schedule by observing that the reference schedule is contained in a rectangular shape and that all start times are a linear translation of a reference start time that depends on the number of additional tokens transferred.

5.4 Buffer Capacity Computation

Given the definition of aggregate firing from the previous section, this section bounds token production and consumption times on a queue e_{ij}. Given these bounds, the same reasoning is followed as in Sections 4.4, 4.5, and 4.6 to derive a sufficient number of initial tokens.

For an actor v_x, we define parameter valuation $\bar{P}(v_x)$ as follows. For every parameter of v_x that has an upper bound, we take the maximum value and for every parameter that does not have an upper bound we take the minimum value. It follows that for every aggregate firing f parameter valuation $P_0(v_x, f)$ always has parameter values that are smaller than or equal to the values in $\bar{P}(v_x)$.

Given parameter valuation $\bar{P}(v_i)$, we have a cumulative firing duration $\bar{\Upsilon}(v_i)$, as defined by Equation (37).

$$\bar{\Upsilon}(v_i) = \sum_{h=1}^{h=\theta_D(v_i)} [\chi_D(v_i, h)]_{\bar{P}(v_i)} \cdot \rho(v_i, h) \tag{37}$$

We define $W(v_i)$ as the set of phases of v_i that do not have an upper bound on their number of firings, and we define $L(v_i, e_{ij})$ as the set of phases of v_i that can produce tokens on queue e_{ij}, that is, these phases have a parameterized token production quantum on queue e_{ij} that can attain a value larger than zero. We define $\hat{\rho}(v_i, e_{ij})$ as the largest firing duration of the phases with an unbounded number of firings that can produce on e_{ij} (i.e., $\hat{\rho}(v_i, e_{ij}) = \max(\{\rho(v_i, h) | h \in W(v_i) \cap L(v_i, e_{ij})\})$).

The following lemma gives a linear upper bound on the token production times of an aggregate firing as defined in Section 5.3. The derivation of this bound can be intuitively understood as follows. The reference start time of any firing n of aggregate firing f is smaller than or equal to $\bar{\Upsilon}(v_i)$. Given these reference start times, all tokens produced on e_{ij} given valuation $\bar{P}(v_i)$ are produced at latest at $\bar{\Upsilon}(v_i)$. Further, any additional firing, relative to valuation $\bar{P}(v_i)$, has a firing duration smaller than or equal to $\hat{\rho}(v_i, e_{ij})$. Therefore, given the reference start times, all tokens of aggregate firing f are produced before $\bar{\Upsilon}(v_i) + \hat{\rho}(v_i, e_{ij})$. Additional token production, relative to valuation $\bar{P}(v_i)$, results in start times that are delayed. However, these start times are delayed linearly in the number of additional tokens along the maximum required rate line, thereby not violating the upper bound on token production times.

LEMMA 12. *An upper bound on the token production time of token number $y \geq 1$ produced in aggregate firing f of v_i on queue e_{ij} is*

$$s(f, \sigma(v_i, e_{ij})) + \bar{\Upsilon}(v_i) + \hat{\rho}(v_i, e_{ij}) + \frac{y-1}{\hat{r}(e_{ij})}. \tag{38}$$

Proof. Let token y be produced by firing n of aggregate firing f. Let n be a firing of phase h of v_i. Token y is produced at the finish time of firing n, which equals

$$s(n,\, f, e_{ij}) + \rho(v_i, h) = s_0(n,\, f, e_{ij}) + \frac{\Xi(n-1,\, f, e_{ij}) - \Xi_0(n-1,\, f, e_{ij})}{\hat{r}(e_{ij})} + \rho(v_i, h). \quad (39)$$

Therefore, we have that

$$s(n,\, f, e_{ij}) + \rho(v_i, h) \leq s_0(n,\, f, e_{ij}) + \frac{\Xi(n-1,\, f, e_{ij})}{\hat{r}(e_{ij})} + \rho(v_i, h). \quad (40)$$

We distinguish two cases based on the reference start time of n and then show that Equation (38) is an upper bound on the finish time of firing n in both cases.

If $s_0(n,\, f, e_{ij}) = s_0(n+1,\, f, e_{ij})$, then according to Equation (32), $\Phi_0(n-1,\, f) = \Phi_0(n,\, f)$. This implies that firing n is not included in $\Phi_0(n,\, f)$. Therefore, n is a firing of a phase $h \in W(v_i)$. Because token y is produced on e_{ij} by firing n, we have that $h \in L(v_i, e_{ij})$, and, therefore, $\rho(v_i, h) \leq \hat{\rho}(v_i, e_{ij})$. This implies that in this case an upper bound on the token production time of y is

$$s(n,\, f, e_{ij}) + \rho(v_i, h) \leq s_0(n,\, f, e_{ij}) + \frac{\Xi(n-1,\, f, e_{ij}) - \Xi_0(n-1,\, f, e_{ij})}{\hat{r}(e_{ij})} + \hat{\rho}(v_i, e_{ij}). \quad (41)$$

If $s_0(n,\, f, e_{ij}) \neq s_0(n+1,\, f, e_{ij})$, then, according to Equation (32), we have that $s_0(n+1,\, f, e_{ij}) = s_0(n,\, f, e_{ij}) + \rho(v_i, h)$. Substitution of $s_0(n+1,\, f, e_{ij}) = s_0(n,\, f, e_{ij}) + \rho(v_i, h)$ in Equation (40) implies that in this case an upper bound on the token production time of y is

$$s(n,\, f, e_{ij}) + \rho(v_i, h) \leq s_0(n+1,\, f, e_{ij}) + \frac{\Xi(n-1,\, f, e_{ij})}{\hat{r}(e_{ij})}. \quad (42)$$

Because $s_0(n,\, f, e_{ij}) \leq s(f, \sigma(v_i, e_{ij})) + \bar{\Upsilon}(v_i)$ and $s_0(n+1,\, f, e_{ij}) \leq s(f, \sigma(v_i, e_{ij})) + \bar{\Upsilon}(v_i)$, an upper bound on the production time of y in both cases is given by

$$s(n,\, f, e_{ij}) + \rho(v_i, h) \leq s(f, \sigma(v_i, e_{ij})) + \bar{\Upsilon}(v_i) + \hat{\rho}(v_i, e_{ij}) + \frac{\Xi(n-1,\, f, e_{ij})}{\hat{r}(e_{ij})}. \quad (43)$$

Because token y is produced by firing n of f, we have that $y \geq \Xi(n-1,\, f, e_{ij}) + 1$. Substitution of $y-1$ for $\Xi(n-1,\, f, e_{ij})$, therefore, implies that Equation (38) is an upper bound on Equation (39). □

The schedule $\sigma(v_i, e_{ij})$ of aggregate firings of actor v_i on output queue e_{ij} is defined by Equation (44), which equals the schedule as defined by Equation (7).

$$s(f, \sigma(v_i, e_{ij})) = s(v_i) + \frac{\Xi(f-1, e_{ij}) - \delta(e_{ij})}{\hat{r}(e_{ij})} \quad (44)$$

Let a token $x \in \mathbb{N}^*$ be produced on e_{ij} by aggregate firing f of v_i, we have that $x = \Xi(f-1, e_{ij}) + y$. With the start time of aggregate firing f given by

Equation (44), and an upper bound on the production time of token y given by Equation (38) of Lemma 12, a linear upper bound on the production time of token x is given by Equation (45).

$$\hat{\alpha}_p(x, e_{ij}, \sigma(v_i, e_{ij})) = s(v_i) + \bar{\Upsilon}(v_i) + \hat{\rho}(v_i, e_{ij}) + \frac{x - \delta(e_{ij}) - 1}{\hat{r}(e_{ij})} \tag{45}$$

We define $M(v_j, e_{ij})$ as the set of phases that can consume tokens from e_{ij}, that is, these are the phases that have a parameterized token consumption quantum on e_{ij} that can attain a value larger than zero. We define $\hat{\gamma}(e_{ij})$ as the largest token consumption quantum of the phases with an unbounded number of firings that can consume from e_{ij} (i.e., $\hat{\gamma}(e_{ij}) = \max(\{\gamma(e_{ij}, o) | o \in W(v_i) \cap M(v_i, e_{ij})\})$).

Given parameter valuation $\bar{P}(v_i)$, we have a cumulative token consumption quantum $\bar{\Gamma}(e_{ij})$ as defined by Equation (46), where $\hat{\gamma}(e_{ij}, h)$ is the maximum value that the parameter denoted by $\gamma(e_{ij}, h)$ can attain.

$$\bar{\Gamma}(e_{ij}) = \sum_{h=1}^{h=\theta_D(v_i)} [\chi_D(v_i, h)]_{\bar{P}(v_i)} \cdot \hat{\gamma}(e_{ij}, h) \tag{46}$$

The following lemma provides a linear lower bound on token consumption times in an aggregate firing f of actor v_j on input queue e_{ij}. The derivation can be understood as follows. Given the reference parameter valuation $P_0(v_j, f)$, the firings of f together consume maximally $\bar{\Gamma}(e_{ij})$ tokens. Any firing that consumes an additional number of tokens maximally has a token consumption quantum of $\hat{\gamma}(e_{ij})$ tokens. Therefore, maximally, $\bar{\Gamma}(e_{ij}) + \hat{\gamma}(e_{ij})$ tokens are consumed at once. Additional consumption relative to $\bar{\Gamma}(e_{ij})$ results in a delay of subsequent firings that corresponds with the number of additional consumed tokens and the slope of the bound on token consumption times. Therefore, these subsequent firings do not consume tokens earlier than the presented lower bound. Figure 25 illustrates that additional token consumption relative to the reference parameter valuation, which is shown in Figure 24, leads to delay of subsequent firings.

LEMMA 13. *A lower bound on the token consumption time of token number* $y \geq 1$ *consumed in aggregate firing* f *of* v_j *from queue* e_{ij} *is*

$$s(f, \sigma(v_i, e_{ij})) + \frac{y - \bar{\Gamma}(e_{ij}) - \hat{\gamma}(e_{ij})}{\hat{r}(e_{ij})}. \tag{47}$$

Proof. Let token y be consumed by firing n of aggregate firing f. Then, y is consumed at $s(n, f, e_{ij})$, which is given by

$$s(n, f, e_{ij}) = s_0(n, f, e_{ij}) + \frac{\Lambda(n-1, f, e_{ij}) - \Lambda_0(n-1, f, e_{ij})}{\hat{r}(e_{ij})}. \tag{48}$$

By construction of $\hat{\gamma}(e_{ij})$, we have that

$$\hat{\gamma}(e_{ij}) \geq (\Lambda(n, f, e_{ij}) - \Lambda_0(n, f, e_{ij})) - (\Lambda(n-1, f, e_{ij}) - \Lambda_0(n-1, f, e_{ij})), \tag{49}$$

which can be rewritten to

$$\Lambda(n-1,\, f, e_{ij}) - \Lambda_0(n-1,\, f, e_{ij}) \geq \Lambda(n,\, f, e_{ij}) - \Lambda_0(n,\, f, e_{ij}) - \hat{\gamma}(e_{ij}). \tag{50}$$

Since $\Lambda_0(n,\, f, e_{ij}) \leq \bar{\Gamma}(e_{ij})$, and $\Lambda(n,\, f, e_{ij}) \geq y$, this can be rewritten to

$$\Lambda(n-1,\, f, e_{ij}) - \Lambda_0(n-1,\, f, e_{ij}) \geq y - \bar{\Gamma}(e_{ij}) - \hat{\gamma}(e_{ij}). \tag{51}$$

Together with Equation (48), this implies that

$$s(n,\, f, e_{ij}) \geq s_0(n,\, f, e_{ij}) + \frac{y - \bar{\Gamma}(e_{ij}) - \hat{\gamma}(e_{ij})}{\hat{r}(e_{ij})}. \tag{52}$$

Since by Equation (32), $s_0(n,\, f, e_{ij}) \geq s(f, \sigma(v_i, e_{ij}))$, Equation (47) is a lower bound on the start time of firing n that consumes token y. Further, because token y is consumed at $s(n,\, f, e_{ij})$, Equation (47) is a lower bound on the consumption time of token y. □

The schedule $\sigma(v_j, e_{ij})$ of aggregate firings of actor v_j on input queue e_{ij} is defined by Equation (53), which equals the schedule as defined by Equation (11).

$$s(f, \sigma(v_j, e_{ij})) = s(v_j) + \frac{\Lambda(f-1, e_{ij})}{\hat{r}(e_{ij})} \tag{53}$$

Let a token $x \in \mathbb{N}^*$ be consumed from e_{ij} by aggregate firing f of v_j, we have that $x = \Lambda(f-1, e_{ij}) + y$. With the start time of aggregate firing f given by Equation (53), and a lower bound on the consumption time of token y given by Equation (47) of Lemma 13, a linear lower bound on the consumption time of token x is given by Equation (54).

$$\check{\alpha}_c(x, e_{ij}, \sigma(v_j, e_{ij})) = s(v_j) + \frac{x - \bar{\Gamma}(e_{ij}) - \hat{\gamma}(e_{ij})}{\hat{r}(e_{ij})} \tag{54}$$

Since, on any queue, tokens can only be consumed after they have been produced, we have that for every token x, $\check{\alpha}_c(x, e_{ij}, \sigma(v_j, e_{ij})) \geq \hat{\alpha}_p(x, e_{ij}, \sigma(v_i, e_{ij}))$ should hold. After substitution of Equations (45) and (54) in $\check{\alpha}_c(x, e_{ij}, \sigma(v_j, e_{ij})) \geq \hat{\alpha}_p(x, e_{ij}, \sigma(v_i, e_{ij}))$, we obtain

$$s(v_j) - s(v_i) \geq \frac{\bar{\Gamma}(e_{ij}) + \hat{\gamma}(e_{ij}) - \delta(e_{ij}) - 1}{\hat{r}(e_{ij})} + \bar{\Upsilon}(v_i) + \hat{\rho}(v_i, e_{ij}). \tag{55}$$

Given the constraint on the minimum difference between the start times of the first aggregate firings of two adjacent actors as specified by Equation (55), the constraints in the network flow problem of Section 4.5 are now given by Equation (56).

$$\beta(e_{ij}) \geq \frac{\bar{\Gamma}(e_{ij}) + \hat{\gamma}(e_{ij}) - \delta(e_{ij}) - 1}{\hat{r}(e_{ij})} + \bar{\Upsilon}(v_i) + \hat{\rho}(v_i, e_{ij}) \tag{56}$$

After the start times of the first aggregate firings of each actor are computed by solving the network flow problem of Algorithm 4.5 using Equation (56), the required number of initial tokens on queue e_{ij} is the smallest integer value that satisfies the constraint in Equation (57).

$$\delta(e_{ij}) \geq \hat{r}(e_{ij}) \cdot (\bar{\Upsilon}(v_i) + \hat{\rho}(v_i, e_{ij}) + s(v_i) - s(v_j)) + \bar{\Gamma}(e_{ij}) + \hat{\gamma}(e_{ij}) - 1 \tag{57}$$

5.5 Sufficiency of Buffer Capacities

In this section, we show that the number of initial tokens as computed in the previous section is a sufficient buffer capacity to let w_τ execute wait-free. This is done by showing that Theorem 5 still holds, which, in turn, is done by showing that Lemmas 10 and 11 from Section 4 still hold given the definition of aggregate firings from Section 5.

In Section 4, schedules of aggregate firings were constructed per queue for every parameter valuation. Lemma 10 showed that the constructed queue-schedule of aggregate firings on any queue e_{ij} that is on a simple directed path from v_i to v_τ has start times that are later than or equal to the start times in the schedules on other adjacent queues of v_i. Therefore, the schedule on e_{ij} determines the actor-schedule of v_i and delays the start times on other adjacent queues. However, while in Section 4, the schedule of firings within an aggregate firing is the same on every queue, this no longer holds for the aggregate firing as defined in Section 5.3. This means that, with the definition of an aggregate firing from Section 5.3, we have a schedule of firings that does not have to be the same on every queue. It is, therefore, no longer sufficient to show the relation between schedules of aggregate firings.

Lemma 16 will show that the start times in the constructed schedules of firings on any queue e_{ij} that is on a simple directed path from v_i to v_τ are later than or equal to the start times of firings of v_i in the schedules on other queues adjacent to v_i. Therefore, the schedule of actor v_i is determined by constructed queue-schedules of firings on the same queues as the queues that in Lemma 10 determined the schedule of aggregate firings.

In Section 4, Lemma 11 showed that if actors v_i and v_j share a parameter p, then the token arrival times on queues with cumulative transfer quanta that are parameterized in p do not influence the schedule of actors v_i and v_j. This result implied that the subgraph G'_p shown in Figure 16 does not influence the schedule of v_τ. In this section, Lemma 17 will show that the result of Lemma 11 is not affected by the change in definition of aggregate firings. Since only Lemmas 10 and 11 involve schedules of aggregate firings, Theorem 5 still holds.

The organization of this section is as follows. Lemma 14 shows that the schedule of aggregate firings on queues toward v_τ will delay the schedules of aggregate firings on queues away from v_τ. This will help in establishing Lemma 16 that states that this same property also holds for the schedule of firings. This section is concluded by Lemma 17.

LEMMA 14. *Lemma 10 holds for aggregate firings as defined in Section 5.3.*

PROOF. We need to show that in the schedules determined for queues individually, the start times of firings of v_i on queue e_{ij} are later than or equal to the start times of firings of v_i on queues e_{ik} and e_{hi}. Because the schedule of aggregate firings is the same as considered in Lemma 10, we have $s(f, \sigma(v_i, e_{ij})) \geq s(f, \sigma(v_i, e_{ik}))$ and $s(f, \sigma(v_i, e_{ij})) \geq s(f, \sigma(v_i, e_{hi}))$ for the same reasons as in the proof of Lemma 10. □

Lemma 16 shows that for an actor v_i the constructed schedule on a queue e_{ij} that is toward v_τ has later start times of firings than the constructed schedules on other queues adjacent to v_i.

For the VPDF graph shown in Figure 23, we have that, for actor v_i, queue $e_{i\tau}$ is a queue toward v_τ. In Figure 25, for instance, we see that the start time of the fourth firing is later in the schedule on $e_{i\tau}$, which is point C' in Figure 25(b), than its start time in the schedule on e_{hi}, which is point A' in Figure 25(a).

In order to show Lemma 16, we require the following lemma. The property $D(e)$ of an edge e, as used in the following lemma, is defined in Definition 13 and is true for an edge that is from an actor v on a path toward v_τ and has a cumulative production quantum that is not parameterized in a shared parameter.

LEMMA 15. *Lemmas 8 and 9 hold in case the VPDF graph has parameters with no upper bound. This means that Equation (58) and Equation (59) hold for output queues e_{ij} and e_{ik} and input queue e_{hi} of actor v_i.*

$$\forall v_i \in V \setminus \{v_\tau\}\ \forall e_{ij}, e_{ik} \in E(v_i) \bullet D(e_{ij}) \implies \frac{\hat{r}(e_{ik})}{\hat{r}(e_{ij})} = \max_{\phi_D(P_D)} \left(\frac{\Pi(e_{ik})}{\Pi(e_{ij})} \right) \quad (58)$$

$$\forall v_i \in V \setminus \{v_\tau\}\ \forall e_{ij}, e_{hi} \in E(v_i) \bullet D(e_{ij}) \implies \frac{\hat{r}(e_{hi})}{\hat{r}(e_{ij})} = \max_{\phi_D(P_D)} \left(\frac{\Gamma(e_{hi})}{\Pi(e_{ij})} \right) \quad (59)$$

PROOF. The proof of Lemmas 8 and 9 rests on showing independence of a number of terms in the ratio that determines the maximum required rate on a queue. This independence is not affected by allowing parameters to not have an upper bound, neither is the fact that we now need to determine limits to find the maximum required rate of influence on this independence. □

LEMMA 16. *The following holds*

$$\forall v_i \in V \setminus \{v_\tau\}\ \forall e, e_{ij} \in E(v_i) \bullet D(e_{ij}) \implies s(n, f, e_{ij}) \geq s(n, f, e) \quad (60)$$

PROOF. We will show that, for every output queue e_{ik} of v_i, $s(n, f, e_{ij}) \geq s(n, f, e_{ik})$. The proof that, for every input queue e_{hi} of v_i, $s(n, f, e_{ij}) \geq s(n, f, e_{hi})$ is completely analogous. We have that $s(n, f, e_{ij})$ is given by Equation (61).

$$s(n, f, e_{ij}) = s_0(n, f, e_{ij}) + \frac{\Xi(n-1, f, e_{ij}) - \Xi_0(n-1, f, e_{ij})}{\hat{r}(e_{ij})} \quad (61)$$

Further, we have that $s(n, f, e_{ik})$ is given by Equation (62).

$$s(n, f, e_{ik}) = s_0(n, f, e_{ik}) + \frac{\Xi(n-1, f, e_{ik}) - \Xi_0(n-1, f, e_{ik})}{\hat{r}(e_{ik})} \quad (62)$$

By construction, we have that $s_0(n, f, e_{ij}) = s_0(1, f, e_{ij}) + \Phi_0(n-1, f)$. Because Lemma 14 tells that $s_0(1, f, e_{ij})$ on queue e_{ij} is larger than or equal to $s_0(1, f, e_{ik})$ on queue e_{ik}, we need to show that Equation (63) is true.

$$\frac{\Xi(n-1, f, e_{ij}) - \Xi_0(n-1, f, e_{ij})}{\hat{r}(e_{ij})} \geq \frac{\Xi(n-1, f, e_{ik}) - \Xi_0(n-1, f, e_{ik})}{\hat{r}(e_{ik})} \quad (63)$$

The terms $\Xi(n-1,\ f, e_{ij}) - \Xi_0(n-1,\ f, e_{ij})$ and $\Xi(n-1,\ f, e_{ik}) - \Xi_0(n-1,\ f, e_{ik})$ denote increases in the number of tokens produced on queues e_{ij} and e_{ik} in the schedule of firings, which are relative to the number of tokens produced in the reference schedule of firings. This increase in the number of tokens produced on queues e_{ij} and e_{ik} is caused by an increase in the number of firings of phases of v_i that have an unbounded number of firings. The increase in the number of tokens produced on queues e_{ij} and e_{ik} equals the sum of the token production quanta on these queues of these additional firings. Let one of these additional firings be a firing of phase m of actor v_i, with a token production quantum $\pi(e_{ij}, m)$ on queue e_{ij} and a token production quantum $\pi(e_{ik}, m)$ on queue e_{ik}. We will show that Equation (64) holds for any such additional firing from a phase m.

$$\frac{\pi(e_{ij}, m)}{\hat{r}(e_{ij})} \geq \frac{\pi(e_{ik}, m)}{\hat{r}(e_{ik})} \tag{64}$$

The numerators of the left and right-hand side of Equation (63) are sums of token production quanta, where these sums have an equal number of terms because this number of terms is determined by the number of additional firings of actor v_i. Because for each of these terms, Equation (64) holds and Equation (63) holds. We have that Equation (64) can be rewritten into Equation (65).

$$\frac{\hat{r}(e_{ik})}{\hat{r}(e_{ij})} \geq \frac{\pi(e_{ik}, m)}{\pi(e_{ij}, m)} \tag{65}$$

Lemma 15 states that Equation (66) holds.

$$\frac{\hat{r}(e_{ik})}{\hat{r}(e_{ij})} = \max_{\phi_D(P_D)} \left(\frac{\Pi(e_{ik})}{\Pi(e_{ij})} \right) \tag{66}$$

We have that m is a phase with an unbounded number of firings. Let p be the parameterized number of firings of m, that is, $p = \chi_D(v_i, m)$, then we have that Equation (67) holds because taking the limit to infinity in p is only one of the cases considered when determining the maximum value of the ratio over all possible parameter values.

$$\max_{\phi_D(P_D)} \left(\frac{\Pi(e_{ik})}{\Pi(e_{ij})} \right) \geq \lim_{p \to \infty} \left(\frac{\Pi(e_{ik})}{\Pi(e_{ij})} \right) \tag{67}$$

A cumulative production quanta is a sum of products where these products have the parameterized number of firings and the parameterized token production quanta as factors. Taking the limit in the parameterized number of firings p evaluates in this case to the ratio of its coefficients in the numerator and denominator, which implies that this limit evaluates to the ratio of production quanta of phase m. This means that Equation (68) holds.

$$\lim_{p \to \infty} \left(\frac{\Pi(e_{ik})}{\Pi(e_{ij})} \right) = \frac{\pi(e_{ik}, m)}{\pi(e_{ij}, m)} \tag{68}$$

Substitution of terms in Equations (66), (67), and (68) results in Equation (65), which therefore holds for any such phase m. Since Equation (65) can be rewritten to Equation (64), which, therefore, holds for any of the terms of Equation (63). □

Given that v_i and v_j, with $v_i \neq v_j$, share parameter p, where on output queue e_{ik} of v_i it holds that $\Pi(e_{ik})$ is parameterized in p, and on input queue e_{hj} of v_j it holds that $\Gamma(e_{hj})$ is parameterized in p. Lemma 17 will show that the schedule of firings of v_i on queue e_{ik} and of v_j on e_{hj} as defined in Sections 5.3 and 5.4 are the same as defined in Section 4. Therefore, Lemma 11 still holds. Lemma 11 states that the constructed actor schedules of v_i and v_j are not influenced by the value of a shared parameter such as parameter p.

LEMMA 17. *Lemma 11 holds for aggregate firings as defined in Section 5.3.*

PROOF. Let actors v_i and v_j, with $v_i \neq v_j$, share a parameter p. Further, let v_i have an adjacent queue e_i with a cumulative token transfer quantum that is parameterized in p and let v_j have an adjacent queue e_j with a cumulative token transfer quantum that is parameterized in p.

By consistency of the graph, all parameters of the cumulative token transfer quanta of v_i on e_i and all parameters of the cumulative token transfer quanta of v_j on e_j are shared between v_i and v_j, or are constant, because these cumulative transfer quanta share parameter p. Because of the restrictions on parameter sharing, all shared parameters have an upper bound (i.e., have a maximum value). This implies that there is no parameter in the mentioned cumulative transfer quanta that has no upper bound. All differences between the schedule of firings on a queue e and the reference schedule of firings on this queue e are caused by insertion of firings from phases with an unbounded number of firings that transfer tokens on queue e. Since the mentioned cumulative transfer quanta are not parameterized in a parameter that has no upper bound, their queue-schedules are equal to their reference queue-schedules. The reference queue-schedule of firings as defined in Section 5 equals the schedule of firings as defined for the rectangular aggregate firings in Section 4. Therefore, the definitions of the queue-schedules in Section 5 and Section 4 result in the same schedule of firings in the queue-schedule of v_i on e_i and the same schedule of firings in the queue-schedule of v_j on e_j. □

6. EXPERIMENT

In this section, we show two example applications that can be modeled and analyzed by VPDF graphs. The first example is a fragment from an IEEE 802.11 receiver taken from Moreira and Bekoij [2007], which exemplifies loops with no known upper bound on the number of iterations. The second example is an H.263 video decoder, which exemplifies communication of parameter values.

6.1 802.11 Receiver

Figure 26 shows a simplified task graph of an 802.11 receiver, with a sink that periodically produces samples, a channel decoding task, CD, and a source

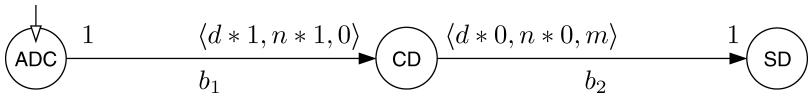


Fig. 26. Simplified task graph of an 802.11 receiver.

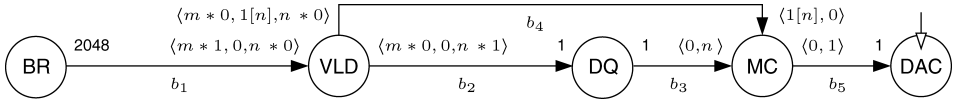


Fig. 27. Simplified task graph of an H.263 video decoder.

decoding task, SD. For every packet decoded by CD, first a signal from the base station needs to be detected and synchronized to. This is done in the first nonblocking code-segment, which executes $d \geq 2$ times. Subsequently, the packet is received, which implies reading n samples. Then m bytes are written into the buffer to the source decoder.

The number of samples that need to be processed before a signal is detected and synchronized to depends on various channel conditions. This implies that there is no known upper bound on d. In Moreira and Bekooij [2007], this aspect of the radio receiver could not be included in their model. Analysis of the VPDF model that corresponds with the task graph from Figure 26 results in buffer capacities that are sufficient to not lose samples at the sink in the different modes and transitions between these modes of this receiver.

In case the ADC has a response time of 10, the vector of response times of the CD is $\langle d*1, n*1, 10\rangle$, and the response time of the SD is 2, and furthermore $n \geq 0$ and $0 \leq m \leq 10$, then a buffer capacity of 5 on buffer b_1 and a buffer capacity of 16 on buffer b_2 is computed by our algorithm. Simulations in our dataflow simulator confirm that these are sufficient capacities.

6.2 H.263 Video Decoder

Figure 27 shows a simplified task graph of an H.263 video decoder. This decoder includes a BR task that reads bytes from a storage device, a VLD task that parses the byte stream and produces macro blocks, a DQ task that dequantises, a MC task that assembles a picture, and the sink that periodically consumes a picture. The DQ task processes macro blocks. Therefore, the MC task needs to be informed by the VLD task how many macro blocks comprise a complete picture. In contrast to VRDF [Wiggers et al. 2008], with VPDF, the value of m does not need to be known before the first execution of this nonblocking code-segment is executed. This allows to model a VLD task that has a loop in which it consumes bytes and exits this loop based on the processed byte stream.

Suppose that the BR has a response time of 10,000, the vector of response times of the VLD is $\langle m*3, 3, n*3\rangle$, the DQ has a response time of 14, the MC has response times $\langle 1{,}000, 32{,}000\rangle$ and the DAC has a response time of 33,000. Furthermore, suppose that $0 \leq m \leq 6{,}536$ and $0 \leq n \leq 2{,}376$, then this results in computed capacities of 15,859 on b_1, 4,302 on b_2, 4,752 on b_3, 3 on b_4, and 2 on b_5.

The presented algorithm is implemented in a tool using the GiNaC [Bauer et al. 2002] library for the algebraic manipulations. Buffer capacities are computed for all presented graphs and confirmed to be conservative with our dataflow simulator.

7. CONCLUSION

Stream processing applications have increasingly complex control structures. This article presented a dataflow model that allows tasks to consist of a sequence of loops, where these loops can have a number of iterations that depends on the processed stream. This data-dependent number of iterations leads to data-dependent task execution rates. For example, task execution in a digital radio receiver can depend on whether the receiver is in the synchronization or synchronized mode.

We have shown that the validity of instances of this dataflow model can be efficiently verified. Furthermore, for every valid instance, buffer capacities can be efficiently computed that are sufficient to satisfy a throughput constraint.

By exposing individual loop iterations of a task, buffer capacities can still be computed for tasks that include loops with an unbounded number of iterations. This is done by bounding, on each buffer, the variation in transfer rate relative to the required transfer rate. This in constrast to bounding the cumulative response time and transfer quanta, which are unbounded in case of unbounded iteration.

The results of this article provide a basis for a mapping flow that computes scheduler settings and buffer capacities such that end-to-end real-time requirements are satisfied for applications that include complex control structures.

APPENDIX

LIST OF SYMBOLS

T	Task graph	7
W	Set of tasks	7
B	Set of buffers	7
P_T	Set of parameters of task graph	8
ζ	Buffer capacity	7
η	Initial number of full containers in buffer	7
κ	Worst-case response time of execution of non-blocking code-segment	8
ϕ_T	Set of values associated with a parameter of the task graph	8
ξ	Parameterized container production quantum	8
λ	Parameterized container consumption quantum	8
θ_T	Number of nonblocking code-segments of a task	7
χ_T	Parameterized number of executions of a non-blocking code-segment of a task	7

$\hat{\zeta}$	Constraint on maximum buffer capacity	9
$\hat{\eta}$	Constraint on maximum number of initially full containers	9
$\mathbb{N}$	Set of nonnegative natural numbers	7
$\mathbb{N}^*$	Set of positive natural numbers	7
γ	Parameterized token consumption quantum	10
π	Parameterized token production quantum	10
θ_D	Number of phases	9
χ_D	Parameterized number of firings of a phase	10
ϕ_D	Set of values associated with a parameter of the dataflow graph	10
P_D	Set of parameters of the dataflow graph	10
$\Gamma(e)$	Parameterized cumulative token consumption quantum on queue e	10
$\Pi(e)$	Parameterized cumulative token production quantum on queue e	10
z_i	Parameterized repetition rate of actor v_i	11
φ	Parameter communicated on this queue	13
S	Set of shared parameters	29
$\hat{r}(e)$	Maximum required token transfer rate on queue e	19
$\Gamma(f, e)$	Token consumption quantum of aggregate firing f on queue e	18
$\hat{\Gamma}(e)$	Maximum cumulative consumption quantum on queue e	18
$\Pi(f, e)$	Token production quantum of aggregate firing f on queue e	18
$\Upsilon(v_i, f)$	Firing duration of aggregate firing f of actor v_i	18
$\hat{\Upsilon}(v_i)$	Maximum firing duration of an aggregate firing of actor i	18
$s(v_i)$	Start time of first aggregate firing of actor v_i	22
$s(f, \sigma)$	Start time of aggregate firing f of actor v_i given schedule σ	22
$\sigma(v_i, e)$	Constructed queue-schedule of actor v_i on queue e	27
$\sigma(v_i)$	Actor-schedule of actor v_i	28

$\check{\alpha}_c(x, e, \sigma)$	Lower bound on consumption time of token x on queue e given schedule σ	22
$\hat{\alpha}_p(x, e, \sigma)$	Upper bound on production time of token x on queue e given schedule σ	22
$\Lambda(f, e)$	Cumulative token consumption quantum on queue e in aggregate firings one up to and including f	24
$\Lambda_0(n, f, e)$	Cumulative token consumption quantum by firings 1 up to and including firing n of aggregate firing f on queue e, given parameter values $P_0(v_i, f)$	45
$\Lambda(n, f, e)$	Cumulative token consumption quantum by firings 1 up to and including firing n of aggregate firing f on queue e	45
$\Xi(f, e)$	Cumulative token production quantum on queue e in aggregate firings one up to and including f	22
$\Xi_0(n, f, e)$	Cumulative token production quantum by firings 1 up to and including firing n of aggregate firing f on queue e, given parameter values $P_0(v_i, f)$	44
$\Xi(n, f, e)$	Cumulative token production quantum by firings 1 up to and including firing n of aggregate firing f on queue e	44
$\Phi_0(n, f)$	Cumulative firing duration of firing 1 up to and including firing n of aggregate firing f given parameter values $P_0(v_i,)$	43
$\Phi(n, f)$	Cumulative response time of firing 1 up to and including firing n of aggregate firing f	43
$P(v_i, f)$	Parameter values of aggregate firing f of actor v_i	42
$P_0(v_i, f)$	Reference parameter values of aggregate firing f of actor v_i	42
$\bar{P}(v_i)$	Upper bound on reference parameter values of aggregate firings of actor v_i	46
$s(n, f, e)$	Start time of firing n of aggregate firing f on queue e	44
$s_0(n, f, e)$	Reference start time of firing n of aggregate firing f on queue e	44
$\bar{\Gamma}(e_{ij})$	Cumulative token consumption quantum on queue e_{ij} given parameter values $\bar{P}(v_i)$	48
$\bar{\Upsilon}(v_i)$	Cumulative firing duration of actor v_i given parameter values $\bar{P}(v_i)$	46

$\hat{\rho}(v_i, e)$	Maximum firing duration of phases that can have additional firings relative to reference parameter values	46
$\hat{\gamma}(e_{ij})$	Maximum token consumption quantum of phases that can have additional firings relative to reference parameter values	48

REFERENCES

BAUER, C., FRINK, A., AND KRECKEL, R. 2002. Introduction to the GiNaC framework for symbolic computation within the C++ programming language. *J. Symbolic Comput. 33*, 1–12.

BERTSIMAS, D. AND TSITSIKLIS, J. N. 1997. *Introduction to Linear Optimization*. Athena Scientific, Nashua, NH.

BHATTACHARYA, B. AND BHATTACHARYYA, S. S. 2001. Parameterized dataflow modeling for DSP systems. *IEEE Trans. Signal Process. 49*, 10, 2408–2421.

BHATTACHARYA, B. AND BHATTACHARYYA, S. S. 2002. Consistency analysis of reconfigurable dataflow specifications. In *Embedded Processor Design Challenges*. Springer, Berlin, 1–17.

BILSEN, G., ENGELS, M., LAUWEREINS, R., AND PEPERSTRAETE, J. 1996. Cyclo-static dataflow. *IEEE Trans. Signal Process. 44*, 2, 397–408.

BUCK, J. 1993. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. Ph.D. thesis, University of California, Berkeley.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

GAO, G. R., GOVINDARAJAN, R., AND PANANGADEN, P. 1992. Well-behaved dataflow programs for DSP computation. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. IEEE, Los Alamitos, CA.

GIRAULT, A., LEE, B., AND LEE, E. A. 1999. Hierarchical finite state machines with multiple concurrency models. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 18*, 6, 742–760.

HAID, W. AND THIELE, L. 2007. Complex task activation schemes in system level performance analysis. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. ACM, New York.

JERSAK, M., RICHTER, K., AND ERNST, R. 2005. Performance analysis of complex embedded systems. *Int. J. Embedded Syst. 1*, 1-2, 33–49.

LEE, E. A. 1991. Consistency in Dataflow Graphs. *IEEE Trans. Parallel Distrib. Syst. 2*, 2, 223–235.

LEE, E. A. AND PARKS, T. M. 1995. Dataflow process networks. *Proc. IEEE 83*, 5, 773–801.

MAXIAGUINE, A., ZHU, Y., CHAKRABORTY, S., AND WONG, W. 2004. Tuning SoC platforms for multimedia processing: identifying limits and tradeoffs. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. ACM, New York.

MOREIRA, O. M. AND BEKOOIJ, M. J. G. 2007. Self-timed scheduling analysis for real-time applications. *EURASIP J. Adv. Signal Process.*

NEUENDORFFER, S. AND LEE, E. A. 2004. Hierarchical reconfiguration of dataflow models. In *Proceedings of the International Conference on Formal Methods and Models for Co-Design*. ACM, New York.

PANKERT, M., MAUSS, O., RITZ, S., AND MEYR, H. 1994. Dynamic dataflow and control flow in high-level DSP code synthesis. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. IEEE, Los Alamitos, CA.

SEN, M., BHATTACHARYYA, S. S., TIEHAN, L., AND WOLF, W. 2005. Modeling Image Processing systems with homogeneous parameterized dataflow graphs. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. IEEE, Los Alamitos, CA.

STILIADIS, D. AND VARMA, A. 1998. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Trans. Networking 6*, 5, 611–624.

STUIJK, S., GEILEN, M. C. W., AND BASTEN, T. 2008. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Trans. Comput. 57*, 10, 1331–1345.

WAUTERS, P., ENGELS, M., LAUWEREINS, R., AND PEPERSTRAETE, J. A. 1996. Cyclo-dynamic dataflow. In *Proceedings of the Workshop on Parallel and Distributed Processing*. IEEE, Los Alamitos, CA.

WIGGERS, M. H., BEKOOIJ, M. J. G., JANSEN, P. G., AND SMIT, G. J. M. 2007. Efficient computation of buffer capacities for cyclo-static real-time systems with back-pressure. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. IEEE, Los Alamitos, CA.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2007a. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proceedings of the Design Automation Conference*. ACM, New York.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2007b. Modelling runtime arbitration by latency-rate servers in dataflow graphs. In *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*. Springer-Verlag, Berlin.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2008. Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. IEEE, Los Alamitos, CA.

Received September 2008; revised April 2009; accepted April 2009