Lightweight Modeling of Complex State Dependencies in Stream Processing Systems

Anne Bouillard[1] Linh T.X. Phan[2] Samarjit Chakraborty[2]
[1]ENS Cachan (Bretagne) / IRISA
[2]Department of Computer Science, National University of Singapore
E-mail: Anne.Bouillard@bretagne.ens-cachan.fr {phanthix, samarjit}@comp.nus.edu.sg

Abstract

Over the last few years, Real-Time Calculus has been used extensively to model and analyze embedded systems processing continuous data/event streams. Towards this, bounds on the arrival process of streams and bounds on the processing capacity of resources serve as inputs to the model, which are used to calculate end-to-end delays suffered by streams, maximum backlog, utilization of resources, etc. This "functional" model, although amenable to computationally inexpensive analysis methods, has limited modeling capability. In particular, "state-based" processing, e.g. blocking write – where the processing depends on the "state" or fill-level of the buffer – cannot be modeled in a straightforward manner. This has led to a number of recent proposals on using automata-theoretic models for stream processing systems (e.g. Event Count Automata [RTSS 2005]). Although such models offer better modeling flexibility, they suffer from the usual state-space explosion problem. In this paper we show that a number of complex state-dependencies can be modeled in a lightweight manner, using a feedback control technique. This avoids explicit state modeling, and hence the state-space explosion problem. Our proposed modeling and analysis therefore extend the original Real-Time Calculus-based functional modeling in a very useful way, and cover much larger problem domain compared to what was previously possible without explicit state-modeling. We illustrate its utility through two case studies and also compare our analysis results with those obtained from detailed system simulations (which are significantly more time consuming).

1 Introduction

The escalating complexity of stream processing systems has prompted the need for modeling and analysis techniques that go beyond those traditionally studied in the literature. Many of these systems process irregular data/event streams and rely on highly dynamic resource management policies

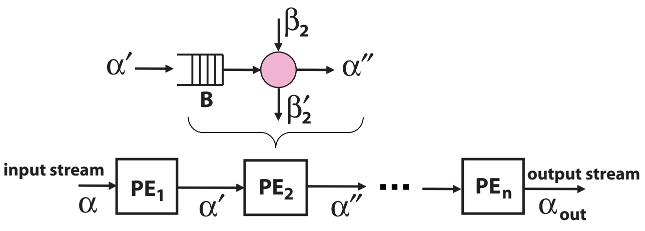

Figure 1. An example system model.

that cannot be modeled using standard periodic/sporadic event models and fixed-priority or deadline-based scheduling policies.

In this context, the Network Calculus framework [4, 7] – which was originally proposed for modeling communication networks – has been extensively adapted in recent years for the modeling and analysis of embedded systems processing continuous data and event streams (e.g., see [5, 22, 24, 26]). The resulting framework (often referred as Real-Time Calculus or RTC in the literature) is designed to model and analyze heterogeneous real-time systems in a compositional manner. The key feature of this framework is its use of *count-based abstraction* to model the timing properties of the input streams, as well as the availability of the resources. In particular, the timing properties of a data stream are specified as a constraint on the maximum and minimum number of data items that may arrive over every time interval of *length* Δ. A collection of such constraints for different values of Δ are captured as functions $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ that denote the lower- and upper-bound on the data arrival process. In other words, $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ specify the minimum and maximum number of items that may arrive within any time interval of length Δ. Clearly, these functions will admit a rich collection of arrival sequences. Standard event models such as periodic, sporadic and periodic with jitter turn out to be special cases of such a specification. Resource availability can also be specified in a similar fashion. Here, $\beta^l(\Delta)$ and $\beta^u(\Delta)$ shall specify the minimum and maximum number of items that can be processed by a resource within any time interval of length Δ. Given the functions α, denoting (α^l, α^u), and β denoting (β^l, β^u), it is possible to compute using purely algebraic techniques, the bounds on system properties such as the maximum delay suffered by the stream and the maximum backlog of data items in front of the resource. Further, it is also possible to compute $\alpha' = (\alpha^{l'}, \alpha^{u'})$ which denotes bounds on the timing properties of the *processed* stream. α' may now serve as the input to the next resource which further processes this stream, and the output from which may be denoted as α''. This is repeated for all resources until the timing properties of the output stream α_{out} are computed (see Figure 1).

Figure 1 shows an architecture consisting of n resources $PE_1, \ldots, PE_n$, which process an input stream sequentially. Each PE_i has an input buffer to store incoming data items waiting to be processed. The service rendered by each PE_i is constrained by β_i. Similar to α', the service function $\beta_i{}'$ that bounds the remaining resource which can be used to process other data streams. Besides buffer requirements and the delay incurred by the input stream at each resource PE_i, various other performance characteristics such as the utilization of each resource, output jitter, the maximum end-to-end delay and the total buffer requirement in the system can also be obtained from the bounds α and α_{out}.

1.1 Our contributions

Owing to the functional nature of RTC, analysis in this framework involves algebraic manipulations which allows for highly efficient computation of system properties in a fully compositional manner. However, modeling of complex *state* dependencies is awkward; common scenarios such as the one where the service offered by a resource depends on the fill-level of a buffer cannot be modeled easily. In constrast, fine-grained modeling of state information, e.g. using timed automata [1, 9] or event count automata [6] often leads to state space explosion when applied to realistic problems.

In this paper, we present a technique to model a variety of complex state dependencies in the existing RTC framework with a feedback control mechanism without resorting to explicit state-space modeling. Firstly, this technique significantly enhances the modeling power of the framework but without having the problems associated with state-space modeling. Secondly, our model of a system is a composition of multiple abstract components with each component capturing all the relevant state-dependencies as well as processing semantics. The properties of these components can be computed functionally using our results and thereby attain a high efficiency. Thirdly, our technique enables state-based scheduling policies to be modeled and efficiently analyzed in a modular manner.

Through case studies, we illustrate how our method can be seamlessly integrated into the current RTC framework, and at the same time we show the effects of capturing state-dependencies on the accuracy of the analysis. We also provide experimental validation of our analysis method against simulation. The analysis results obtained from both methods match well with each other, however our analysis is significantly faster than simulation.

1.2 Related work

The first line of related work is concerned with developing task and event models that generalize classical periodic or sporadic event models, which assume fixed execution times for tasks. Towards this, timed automata and related automata-theoretic formalisms have been recently used in various setups to model and analyze task scheduling problems (e.g., see [1,8,9]). To overcome the lack of state-based modeling in the RTC framework, we had proposed *event count automata* (ECAs) [6] that retain the count-based abstraction used in RTC. Although automata-based models are much more expressive and capable of representing a wide variety of state-dependencies, they suffer from the state explosion problem and can become inefficient when analyzing large system architectures.

The next line of work focuses on extending RTC to model complex event patterns and task activation schemes. For example, [10] presented a method to model conditional blocking-read on an input buffer. In particular, it modeled tasks that are triggered by events on multiple input streams using AND/OR-activation, where an OR-task is triggered whenever an event is available on either of the input streams and an AND-task is only activated when there is at least one event from each stream. [13] proposed a way to compute delay and output arrival functions of data streams that are split and joined during the system execution following the OR-activation and in-order activation semantics, while taking into account correlations in data streams and data distribution based on different types of delay. Correlation between jitter and response time of individual events were also considered in [12]. Analysis methods of more complex scheduling policies such as non-preemptive and scenario-aware scheduling of tasks were studied in [10] and [11] respectively. Timing properties of hierarchical event streams that are generated by the communication stack are modeled in [19]. Although extensive in variety, these proposed techniques do not handle state-modeling and control-feedback dependencies.

The back-pressure effect with finite buffer capacities has been studied in the context of data flow graphs [15]. For instance, in [27], an algorithm for computing the buffer capacities that satisfy throughput constraints was presented. Analysis of self-time scheduling for multirate data flow with finite buffer capacities was considered in [16]. Also, back-pressure was used in [23] as a mechanism to allow a semantics preserving implementation of synchronous models on Loosely Time Triggered Architectures. The methods used in this context however are not applicable into our setting.

There have also been hybrid frameworks that combine various analysis methodologies. For example, [21] unified the SDF [15] and SymTA/S [2] into a single framework that is able to model data-dependencies using SDF and to abstract event streams using SymTA/S. SymTA/S and RTC have been merged in [14] to capture more complex interactions with high accuracy. RTC and ECA can also be integrated using the interfacing technique provided in [18] to achieve higher accuracy than using RTC alone while being more efficient than using ECA alone. The method we propose here can be plugged into the integrated RTC-ECA framework to further increase the efficiency of the analysis, since we can now use RTC to analyze a number of state-dependent components in the system instead of using ECAs, which will in turn reduce the total analysis time.

1.3 Organization of the paper

In the next section we describe the basic concepts of the RTC framework. In Section 3 we present our analysis method. We begin with an example that will be used to illustrate our method, followed by an overview of our analysis technique in Section 3.2. Sections 3.3 and 3.4 establish the theoretical results that enable the analysis of a state-dependent component, which will be applied to model state-based scheduling in Section 3.5. In Section 4 we present experimental results using two case studies derived from an MPEG-2 decoder to illustrate the benefits of our analysis methods. Finally, we conclude with a discussion on the prospects for extending our study initiated in this paper.

2 The Real-Time Calculus Background

RTC is based on the (min,+) algebra [4, 7] and models data streams and services in a network with non-decreasing non-negative functions taking their values in the (min,+) semiring. More formally, $(\mathbb{R}_{\min+}, \min, +)$, with $\mathbb{R}_{\min+} = \mathbb{R}_+ \cup \{\infty\}$, is a commutative semi-ring, its zero element is ∞ and its unitary element is 0.

Consider the set $\mathcal{F} = \{f : \mathbb{R}_+ \to \mathbb{R}_{\min+} \mid \forall s < t,\ 0 \le f(s) \le f(t)\}$. One can define as follows two operators on $\mathcal{F}$: the minimum, denoted by $\oplus$ and the (min,+) convolution, denoted by $\otimes$:

for all f, g in $\mathcal{F}$, $\forall t \in \mathbb{R}_+$,

- $f \oplus g(t) = \min(f(t), g(t))$ and
- $f \otimes g(t) = \inf_{0 \le s \le t}(f(s) + g(t-s))$.

The triple $(\mathcal{F}, \oplus, \otimes)$ is also a commutative semiring and the convolution can be seen as an analogue to the classical (+,×) convolution of filtering theory, transposed in the (min,+) algebra. Its zero element is the function $\varepsilon : t \mapsto \infty$ and its unitary element is $e : 0 \mapsto 0; t \mapsto \infty$.

Two other important operators for RTC are the sub-additive colsure and the (min,+) deconvolution, denoted by $\oslash$: let $f, g \in \mathcal{F}$,

- $f^* = \bigoplus_{n=0}^{\infty} f^n$, where $f^0 = e$ and $f^{n+1} = f^n \otimes f$.
- $f \oslash g(t) = \sup_{u \ge 0}(f(t+u) - g(t))$.

The following lemma holds for the sub-additive closure operator.

Lemma 1. *([7, theorem 2.1.6]) Let $f, g, h \in \mathcal{F}$, and consider the inequation $f \le f \otimes g \oplus h$. Then we have*

$$f \le h \otimes g^*.$$

2.1 Arrival and service curves

Given a data stream traversing a system that contains a single processing element (PE), let A be its cumulative arrival function (i.e. $A(t)$ is the number of data items that have arrived until time t). Here a data item can be a network packet or a video/audio macroblock. We say that α is an (upper) *arrival curve* for A (or that A is upper-constrained by α) if $\forall s, t \in \mathbb{R}_+$, $A(t+s) - A(s) \le \alpha(t)$. This means that the number of items arriving between time s and $t+s$ is never larger than $\alpha(t)$. An important particular case of arrival curve is the affine functions: $\alpha(t) = \sigma + \rho t$. Then σ represents the maximal number of items that can arrive simultaneously (the maximal burst) and ρ the maximal long-term rate of arrivals.

Consider D the cumulative departure function of the stream, defined similarly by the number $D(t)$ of items that have left the system until time t. The system provides a (minimum) *service curve* β , $D(t) \ge A \otimes \beta$. Particular cases of service curves are the *peak rate* functions with rate r (the system can process r items per unit of time and $\beta(t) = rt$) and the *pure delay* service curves with delay d: $\beta(t) = 0$ if $t < d$ and $\beta(t) = +\infty$ otherwise. The combination of those two service curves gives a *rate-latency* function $\beta : t \mapsto R(t-T)_+$ where a_+ denotes $\max(a, 0)$.

A *strict service curve* β is a service curve such that for all $t \in \mathbb{R}_+$, let u be the last instant before t when there is no packet in the system, then $D(t) \ge A(u) + \beta(t-u)$.

2.2 Performance characteristics and bounds

The worst-case backlog and the delay can be easily characterized in the RTC framework as below.

Definition 1. *Let A be the arrival function of a data stream through a system and D be its corresponding departure function. Then the* backlog *of the stream at time t is*

$$b(t) = A(t) - D(t)$$

and the delay (assuming FIFO order for processing items *of the stream) at time t is*

$$d(t) = \inf\{s \geq 0 \mid A(t) \leq D(t+s)\}.$$

Given an arrival curve and a service curve, it is possible to compute with the RTC operations the maximal backlog and delay.

Theorem 1 ([4, 7]). *Let A be the arrival function with an arrival curve α for a stream entering a system with service curve β. Let D be the departure function. Then,*

1. $b(t) \leq B_{\max} = \sup\{\alpha(t) - \beta(t) \mid t \geq 0\}$,
2. $d(t) \leq D_{\max} = \inf\{d \geq 0 \mid \forall t \geq 0,\ \alpha(t) \leq \beta(t+d)\}$.

The maximal backlog is the maximal vertical distance between α and β while the maximal delay is given by the maximal horizontal distance between those two functions. Figure 2 illustrates this fact.

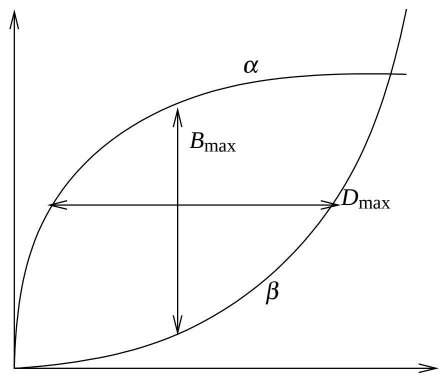

Figure 2. Guarantee bounds on backlog and delay.

Further, bounds on the output stream and the remaining resource of the system can be determined using Theorem 2.

Theorem 2 ([4]). *Assume a stream constrained by an arrival curve α entering a system with service curve β.*

1. *The output stream is upper-constrained by an arrival curve*
$$\alpha' = \alpha \oslash \beta.$$
2. *If β is a strict service curve, the remaining resource after processing the stream is bounded by a service curve*
$$\beta' = (\beta - \alpha)_+.$$

In this paper, we assume that service curves are strict to ensure for the positiveness of remaining service; however, our method is not restricted to this assumption.

From the results concerning systems with a single PE, one can obtain more general results for systems with multiple PEs, using the composition of the RTC operators. For example, if there are two PEs in sequence, for respective service curves β_1 and β_2, the overall service curve is $\beta_1 \otimes \beta_2$ (see [4, 7] for details). Such results have been based on the properties of the (min,plus) algebra.

3 Modeling complex state-dependencies

Modern stream-processing systems are usually heterogeneous networks of resources processing multiple data streams using complex scheduling policies. Often, the processing of a stream depends not only on the available service but also the internal state of the system. One typical example is when the amount of on-chip memory is limited and hence the internal buffers that hold the processed items can only accommodate up to a certain capacity. To avoid loss of data, the processor may implement blocking-write for its output buffers, i.e it stalls whenever the buffers are full. Otherwise, to save resource, it may proceed to process the next data streams based on some sharing policy, in case the output buffer that stores the currently processed stream is filled up.

Modeling and analysis of systems described above require us to take into consideration the state-dependencies that are imposed among the different elements of a system. The original RTC framework presented in Section 2 (i) does not express state-information and furthermore (ii) assumes that all buffers have infinite capacities. As a result, it is not able to represent and correctly analyze such systems. Automata-based approaches developed recently for stream processing systems [6, 17] can encapsulate state information; however, their analyses become inefficient for large systems due to the state-space explosion. In this section, we present a functional analysis technique, developed on top of the original RTC framework, which is capable of capturing the complex state-dependencies while achieving high efficiency. We shall illustrate our method with an example of stream-processing systems that is described below. More general systems can be easily modeled and analyzed using the same approach.

3.1 An illustrative example

Figure 3 sketches the system architecture of a picture-in-picture (PiP) application where two video streams are decoded. The first stream represents a set of regular video clips with high motion contents and the latter represents a set of still images. After fully decoded by the PEs, they will be displayed at the output device.

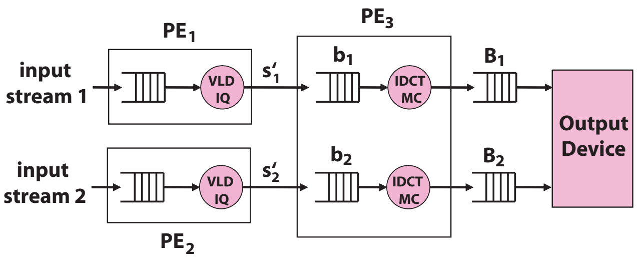

Figure 3. A PiP application.

The system consists of three PEs on which the tasks of an

MPEG-2 decoder application are partitioned and mapped. As shown in the figure, the Variable Length Decoding (VLD) and Inverse Quantization (IQ) tasks run on each of PE_1 and PE_2 while the Inverse Discrete Cosine Transform (IDCT) and Motion Compensation (MC) tasks run on PE_3. PE_1 processes the first input video stream and PE_2 processes the second input video stream. The partially decoded streams from PE_1 and PE_2 (denoted by s_1' and s_2') are stored in the buffers b_1 and b_2 respectively, where they will further be processed by PE_3. The two fully decoded streams from PE_3 are then written to the playout buffers B_1 and B_2 before being read by the output device.

PE_3 schedules the two streams s_1' and s_2' using a fixed-priority scheduling policy, with s_1' having higher priority than s_2'. Further, PE_3 implements blocking-write on the playout buffer B_1; when B_1 is full, the processor will process s_2' if there are some items in b_2. This is done regardless of whether there are items in b_1.

Given the above system architecture, we are interested in answering questions concerning the behavior of the system such as (1) what is the maximum backlog of a buffer? (2) what is the maximum delay experienced by a stream? (3) is the system schedulable while guaranteeing none the buffers overflows? A correct evaluation of such properties are essential for designers to optimize the design of the system. As mentioned earlier, we cannot use the standard RTC framework to analyze the system since PE_3 implements state-based scheduling scheme.

3.2 A functional analysis approach

In constructing the model for the system, we describe each input stream to the system as an arrival curve and each processing resource of the system as a service curve. The processing of a stream by a resource is represented by an abstract component whose inputs are the arrival curve of the input stream and the service curve of the resource. The outputs of an abstract component are an arrival curve that bounds the output stream and a service curve that bounds the resource left after processing the input stream. By connecting the abstract components following the flow of the stream (from left to right) and the order at which the different streams are processed by a shared resource (from top to bottom), we obtain the complete abstract model of the system. Figure 4 depicts the abstract model of the system architecture in Figure 3.

In this figure, α_1 and α_2 denote the arrival curves of the two input video streams; β_1, β_2 and β_3 denote the service curves of PE_1, PE_2 and PE_3 respectively. Similarly, β_4 and β_5 are the service curves that bound the consumptions of the two fully processed streams. The processing of the streams by PE_1 and PE_2 are represented by the abstract components C_1 and C_2, whose output arrival curves are denoted by α_1' and α_2'. The processing of PE_3 on the output streams s_1' and

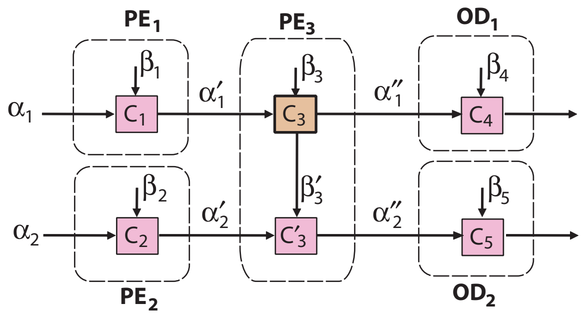

Figure 4. The abstract model of the system in Figure 3.

s_2' comprises two abstract components C_3 and C_3'. Since PE_3 processes s_1' before s_2', the remaining service of C_3 is connected as input to C_3'. Finally, the consumption of the items from B_1 and B_2 are modeled by C_4 and C_5. The connection of the arrival curves to the abstract components follows the sequence at which the corresponding streams are processed.

Our analysis proceeds component-wise where we evaluate each abstract component and thereafter combine the evaluated results. To analyze a component, we first determine the input arrival curve and *effective* service curve of the component if they are not yet known. An effective service curve is a service curve that bounds the actual resource used to process a stream taking into account the state dependencies in the system. Based on the obtained input arrival curve and effective service curve, we can compute the different performance characteristics and bounds of the component using Theorem 1 and 2.

In Figure 4, since C_1 and C_2 have no state dependency with the succeeding components, their effective curves are equal to β_1 and β_2, respectively. By the same reason, the output arrival curve α_1' from C_1 can be computed from α_1 and β_1 using Theorem 2. Similarly, the arrival curve α_2' can be derived from α_2 and β_2.

On the other hand, since the processing at PE_3 is contingent on the state of the playout buffer B_1, the actual resource that is used to process s_1' depends not only on the total available resource of PE_3 but also the readout rate of the output device and the capacity of B_1. Hence the effective service curve β_3^{eff} of C_3 is dependent on β_3, β_4 and the capacity of B_1. This effective service curve will in turn affect the remaining service curve β_3'. The computation of β_3^{eff} and β_3' will be described in the coming sections. With the obtained β_3^{eff} and β_3', we can apply Theorem 2 to compute the output arrival curves α_1'' and α_2'' that are inputs to C_4 and C_5. The effective service curves of C_4 and C_5 are exactly β_4 and β_5 since there is no state-dependency in these two components.

In the next three sections, we present our technique for computing the effective service curve of a component taking into account the state-dependency of the subsequent components in the system. Section 3.3 looks into the case where the processing of the stream in the component is dependent on only one buffer of the next component. Section 3.4 moves one step further to solve for the general case when the processing within the component is dependent on the buffer state of many components in tandem. The computation of the effective service curves of components that are scheduled using fixed-priority policy while being subjected to the state of the buffers in the system is described in Section 3.5. Before going into the details, we first prove the following lemma, which will be used in our formulation.

Lemma 2. *Let f and g be two functions and c be a constant. Then,*

1. $(f \otimes g) + c = (f + c) \otimes g = f \otimes (g + c)$
2. $(f \otimes g^*)^* = e + f^+ \otimes g^*$ *and* $(f \otimes g^*)^+ = f^+ \otimes g^*$,

where $f^+ = \bigoplus_{n>0} f^n$.

Proof. 1. Let $h_c \in \mathcal{F}$, such that $h_c(0) = c$ and $h_c(t) = \infty$, $\forall t > 0$. Then, for every function $f \in \mathcal{F}$, $f \otimes h_c = f + c$. The formula follows from the associativity and the commutativity of the $\otimes$ operator.

2. $(f \otimes g^*)^* = \bigoplus_{n \geq 0}(f \otimes g^*)^n = e \oplus \bigoplus_{n>0}(f \otimes g^*)^n$. As $(g^*)^n = g^*$, then $(f \otimes g^*)^* = e \oplus g^* \bigoplus_{n>0} f^n$, hence the result.

□

3.3 Simple blocking-write at a single buffer

In this section, we model a setup where the processing of a stream is interrupted when an output buffer is full (i.e. blocking-write). This is done by extending the existing RTC framework using a feedback control mechanism. The system, shown in Figure 5 consists in two PEs in tandem, where the second one has a finite capacity B_2. The PEs have respective service curves β_1 and β_2, and when the backlog in PE_2 exceeds B_2, then the service at PE_1 is interrupted. The functions A_1, A_2 and A_3 are the respective arrival processes at the entrance of the system, after serviced by the first PE and at the output of the system.

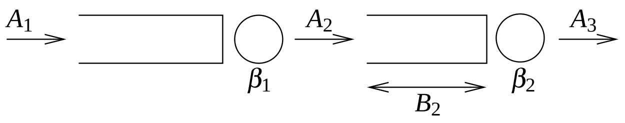

Figure 5. System with two PEs in tandem, the second PE has a finite capacity buffer B_2.

As the backlog on the second PE cannot exceed B_2, one must have $A_2 - A_3 \leq B_2$ and then $A_2 \leq A_3 + B_2$. A simple solution to ensure this is to put a feedback control before PE_1, admitting only the amount of data that ensures that the backlog constraint in PE_2 is satisfied. In entrance of PE_1, the arrival process then becomes $A_1' = \min(A_1, A_3 + B_2)$.

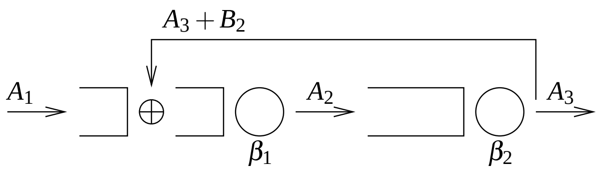

Figure 6. Feedback control to ensure non-overflow for the second buffer.

Figure 6 represents the system that can be translated into the following equations:

$$\begin{cases} A_2 & \geq & \min(A_1, A_3 + B_2) \otimes \beta_1 \\ A_3 & \geq & A_2 \otimes \beta_2, \end{cases}$$

which leads to the following inequation:

$$\begin{aligned} A_2 \geq \min(A_1, A_2 \otimes \beta_2 + B_2) \otimes \beta_1 \\ = \min(A_1 \otimes \beta_1, A_2 \otimes (\beta_2 + B_2) \otimes \beta_1). \end{aligned}$$

From Lemma 1, the solution if this inequality is:

$$A_2 \geq A_1 \otimes \beta_1 \otimes [(\beta_2 + B_2) \otimes \beta_1]^*,$$

which proves the following lemma:

Lemma 3. *The effective service curve for PE_1 taking into account the interruption of service when B_2 is full is:*

$$\beta_1^{\mathit{eff}} = \beta_1 \otimes [(\beta_2 + B_2) \otimes \beta_1]^*.$$

Suppose α is the arrival curve of the input stream. The maximal backlog and delay of the PE as well as the arrival curve of the output stream can be easily deduced from α and β_1^{eff} following Theorem 1 and 2.

3.4 Blocking-write at multiple buffers in sequence

Now suppose that there are several PEs in sequence, each of which has a buffer with finite capacity. Then, the effective service curve of the first PE (the one which has an infinite capacity) will depend on the capacities of all the buffers and on the other service curves: the feedback controls induce some cycles in the network, as shown in Figure 7.

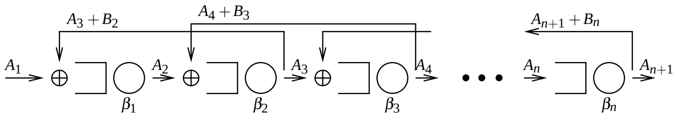

Figure 7. Line of PEs with backlog constraints.

Consider a system with n PEs processing sequentially, where $PE_2, \cdots, PE_n$ have respective buffer capacities $B_2, \ldots, B_n$. Theorem 3 gives the formula for computing the effective service curve of PE_1 considering the blocking-write at the subsequent PEs.

Theorem 3. *The effective service curve β_1^{eff} of PE_1, taking into account the feedback controls from the other PEs, is given by*

$$\beta_1^{eff} = \beta_1 \otimes \min_{i=0}^{n} \bigotimes_{j=1}^{i} [\beta_{j-1} \otimes (B_j + \beta_j)]^+.$$

Proof. The result is proved by induction. The initialization step exactly corresponds to Lemma 3. Suppose that the result holds for n PEs, and let us show it for $n+1$ PEs. The system obeys to the following equations:

$$\begin{cases} A_i & \geq & \min(A_{i-1}, A_{i+1} + B_i) \otimes \beta_{i-1} \\ & & \qquad \forall i \in \{2, \ldots n+1\}, \\ A_{n+2} & \geq & A_{n+1} \otimes \beta_{n+1}. \end{cases}$$

In particular, the equations concerning PE_{n+1} are:

$$\begin{cases} A_{n+1} & \geq & \min(A_n, A_{n+2} + B_{n+1}) \otimes \beta_n \\ A_{n+2} & \geq & A_{n+1} \otimes \beta_{n+1}, \end{cases}$$

which is equivalent to $A_{n+1} \geq A_n \otimes \beta_n \otimes [(\beta_{n+1} + B_{n+1}) \otimes \beta_n]^*$. Now, one can replace the system with $n+1$ PEs in tandem with an equivalent system with n PEs, the $n-1$ first PEs having the same service curve, and the last PE having service curve $\beta'_n = \beta_n \otimes [(\beta_{n+1} + B_{n+1}) \otimes \beta_n]^*$. The capacities of buffers 2 to n remain the same. Consequently, on can apply the induction hypothesis to that system and obtain a service curve for the first PE (denoting $\beta_i \otimes (B_{i+1} + \beta_{i+1})$ by f_i and $\beta_{n-1} \otimes (B_n + \beta'_n)$ by f'_{n-1}):

$$\beta_1^{eff} = \beta_1 \otimes [e \oplus f_1^+ \oplus f_1^+ \otimes f_2^+ \oplus \cdots \oplus f_1^+ \otimes \cdots \otimes f_{n-2}^+ \otimes f'^+_{n-1}].$$

Consider only the last factor f'^+_{n-1} and replace β'_n by its expression:

$$\begin{aligned} f'^+_{n-1} &= [\beta_{n-1} \otimes (B_n + \beta_n \otimes [(\beta_{n+1} + B_{n+1}) \otimes \beta_n]^*)]^+ \\ &= [\beta_{n-1} \otimes (\beta_n + B_n) \otimes [(\beta_{n+1} + B_{n+1}) \otimes \beta_n]^*)]^+ \\ &= [f_{n-1} \otimes f_n^*]^+ = f_{n-1}^+ (e \oplus f_n^+), \end{aligned}$$

which leads the desired formula and finishes the proof. □

Remark that this formula takes into account the fact that if PE_i on the line has no backlog constraint (the buffer has an infinite capacity and $B_i = \infty$), then the service curve does not depend on the service curves β_j, $j \geq i$. Indeed, in that case $f_i = \infty$ and $f_i^+ \otimes \cdots \otimes f_j^+ = \infty$. Analysis bounds on the system can be easily computed in the same fashion as done in the simple case when blocking-write is imposed at a single buffer.

Similar networks have been studied (see [7]), but the goal in this reference is to compute the overall service curve. Here, we are only interested in findig the real service curve of the first server, in order to dimension the size of the buffer, taking into account the block-writing phenomenon.

3.5 State-based scheduling policies

Recall that in the illustrative example in Figure 3 (Section 3.1), the resource of PE_3 is shared between two streams s'_1 and s'_2. The processing policy used by PE_3 is dependent on the state of the playout buffer B_1 as well as the priorities of the streams. Specifically, PE_3 will first process s'_1 and only process s'_2 when b_1 is empty or B_1 is full.

The analysis for such a state-based scheduling policy is divided into two phases. First, we determine the effective service curve that is used to process each stream taking into consideration the state dependency. With the obtained effective service curves, we compute the various performance characteristics using the results presented in Section 2.

The effective service curve β_3^{eff} that is used to process s'_1 taking into account the state dependency is computed using Lemma 3 and Theorem 3 described above. The effective service curve that is used to process s'_2 is also the one that bounds the remaining resource after processing s'_1, which can be computed using Corollary 1 below.

Corollary 1. *Given a PE with service curve β processing a higher priority stream s with arrival curve α. Assume β^{eff} is the effective service curve used to process the stream taking into account the state-dependency. The remaining resource of the PE given to the lower priority streams is bounded by the service curve*

$$\beta' = \beta - \left(\alpha \oslash \beta^{eff}\right).$$

Proof. For any given $\Delta > 0$, the maximum number of items that are processed in any interval of length Δ is $(\alpha \oslash \beta^{eff})(\Delta)$ (Theorem 2.1). Since the PE starts processing the lower priority streams as soon as there is no more items from s to be processed or when the output buffer is full, the amount of service given to the lower priority streams in any interval of length Δ is at least $\beta(\Delta) - (\alpha \oslash \beta^{eff})(\Delta)$. This proves the corollary. □

Using the computed effective service curve, the maximum backlog, maximum delay the output arrival curve can be easily computed using Theorem 1 and 2.1.

4 Experimental Case Studies

We employed two case studies to evaluate our theoretical results and to illustrate the effect of state-dependency on the behavior of a system. The first case study aims to show the immediate ramification of processor stalling on the fill-levels of the buffers in a system. The second case study illustrates the intermediate effect of blocking of one stream on another, at the same time demonstrates how our technique can be used to evaluate systems with multiple input streams that are scheduled using fixed-priority scheduling policy with state-dependency.

4.1 Case study 1: Blocking-write of a stream

Figure 8 shows the architecture of an MPEG-2 decoder which consists of two PEs decoding an input video stream in sequence. The MPEG-2 application is partitioned and mapped onto these PEs where the VLD and IQ tasks run on PE_1 while the IDCT and MC tasks run on PE_2. The coded input stream arrives at the system at a constant rate and it is initially processed by PE_1. The partially decoded macroblocks of the stream are subsequently stored in the buffer b before being processed by PE_2. The resulting fully decoded macroblocks are written to the playout buffer B and finally transmitted to the output video device. We are particularly interested in the behavior of the components shown in the rectangle box.

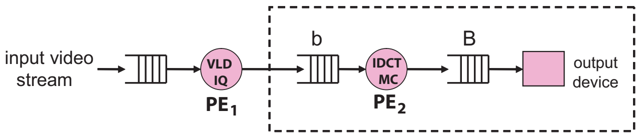

Figure 8. An MPEG-2 decoder application.

Although the input bit stream enters the system at a constant bit-rate, the execution times of the VLD + IQ tasks may vary. The number of bits constituting each partially decoded macroblock is also not constant. Consequently, the stream that is written to b is highly bursty. We assume that (a) the bursty behavior of this stream is specified by an arrival curve $\alpha(\Delta)$; (b) the variability in the execution requirements of the IDCT + MC tasks running on the PE_2 is captured by a service curve $\beta_f(\Delta)$ where f represents the clock frequency of PE_2; (c) PE_2 implements blocking-write for B where it stalls when B is full; (d) the output device reads the macroblocks from B at a constant rate r.

In order to understand the effect of blocking-write on the behavior of the system, we permuted different capacities of B and frequencies f of PE_2 to observe the resulting changes on the maximum backlog of b. The maximum backlog of b is computed following Lemma 3 and compared with our simulated results. The theoretical technique is implemented using the Java API provided by the Real-Time Calculus Toolbox [25].

4.1.1 Obtaining arrival and service curves

To obtain the arrival curve that characterizes the partially decoded stream, we collected execution traces of the different tasks by simulating their executions on a customized version of the SimpleScalar instruction set simulator [3]. From these traces, we measured the execution demands of the VLD and IQ tasks for each macroblock and derived a function $x(t)$, where $x(t)$ denotes the number of macroblocks arriving at b during the time interval $[0,t]$. The function $x(t)$ was used to compute the arrival curve $\alpha(\Delta)$ of the partially processed stream.

Similarly, based on the execution demands of the IDCT and MC tasks, we computed the workload function $\gamma(k)$ which gives the maximum number of cycles required by any k consecutive macroblocks at PE_2. By combining γ and the frequency f of PE_2, we derive the service curve $\beta_f(\Delta)$. The service curve that represents the consumption at the output device is given by $C(\Delta) = r\Delta$.

4.1.2 Analysis results

Figure 9 reports the estimated values of the maximum backlog of b against the different frequencies f of PE_2 for the varying capacities of buffer B.

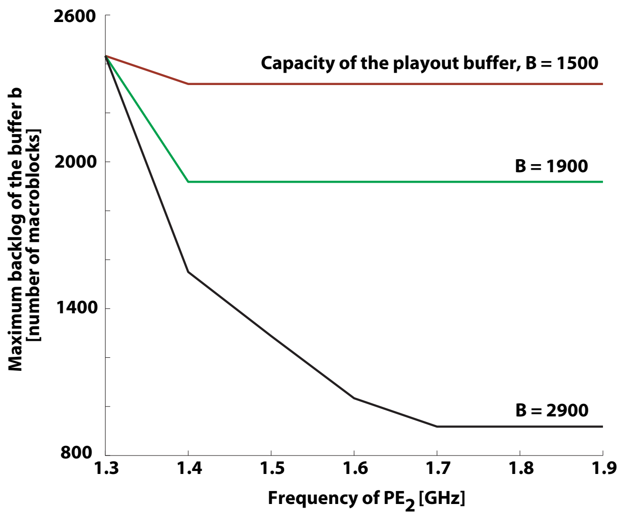

Figure 9. Maximum backlog analysis of b.

As f increases, more macroblocks are processed thus leading to declining backlog for all the different capacities of B. However, beyond a certain threshold frequency, the value of the maximum backlog of b stabilizes for a fixed capacity of B. This happens at the point where B is full and no other macroblocks in b can be processed until there is available space in B. Therefore, as we increase the capacity of B, the maximum backlog of b decreases, which is also illustrated in the figure.

Thus, running a processor beyond a threshold frequency reduces its utilization rate. With a fixed memory size, a designer can therefore determine the processor frequency and the corresponding capacities of the buffers in the system which maximize the utilization rate of the processor.

To compare our method against simulation-based approaches, we implemented a SystemC simulator and used it in conjunction with the SimpleScalar instruction set simulator in [3] to run a detailed SimpleScalar+SystemC simulation. Our simulated results match well with the ones computed using our analytical method described above. In particular, the analytical bounds are always less than 5% more than those obtained from simulation. On the other hand, performing the analysis using the RTC toolbox incorporated with our method is significantly faster than pursuing the pure simulation approach (less than a minute vs. several hours). Note also that results from simulation are unable to provide a formal guarantee on the maximum backlog that may incur in the system.

4.2 Case study 2: State-based scheduling of multiple input streams

In this case study, we analyze the example system described in Section 3.1. We assume the two input video streams arrive at a constant bitrate of 8 Mbps and their fully decoded macroblocks are read by the output device at a rate of 40,000 macroblocks per second. The frequencies of PE_1 and PE_2 are set to be $f_1 = 1.3GHz$ and $f_2 = 1.25GHz$, respectively. Given these values fixed, we are interested in the backlog of b_2 with respect to different values of the frequency of PE_3 and capacity of B_1.

First, we obtain the arrival curves α_1 and α_2 of the two input streams and the service curves β_1, β_2 and β_3 of the three PEs (see Section 4.1.1). From α_1 and β_1, we compute the arrival curve α_1' that characterizes the output stream from PE_1. Similarly, the arrival curve α_2' that represents the output stream from PE_2 is derived from α_2 and β_2. Based on α_1', β_3 and the capacity of B_1, we follow Lemma 3 to compute the effective service curve β_3^{eff} that processes the output stream from PE_1 considering blocking-write at B_1. The remaining service curve β_3' that is used to process the second stream is deducted from β_3^{eff} and α_1' using Corollary 1. The maximum backlog of b_2 is then calculated from α_2' and β_3' using Theorem 1.

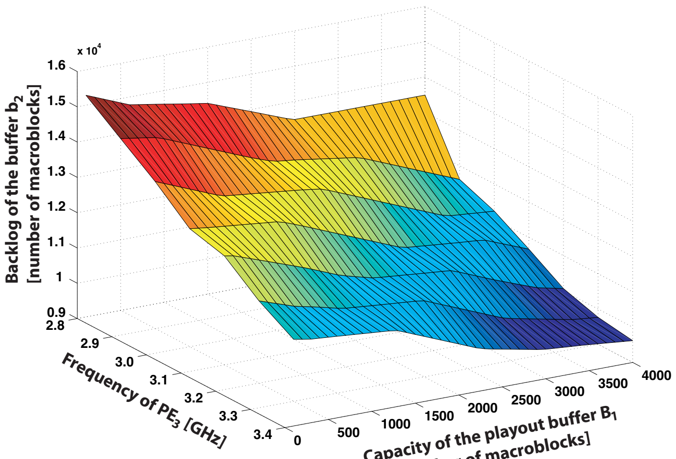

Figure 10. Case study 2: Maximum backlog of the lower priority stream.

Figure 10 plots the maximum backlog of b_2 in relation to the capacity of B_1 and the frequency of PE_3. Observe from the figure that the maximum backlog of b_2 is smallest around the region with highest frequencies of PE_3 and largest capacities of B_2. Reversely, b_2 has smallest backlogs in the area where PE_3 has lowest frequencies and B_2 has smallest capacities. In fact, the result shown in the figure demonstrates that the maximum backlog of b_2 is always proportional to the frequency of PE_3 and the capacity of B_1. Thus the maximum backlog of the input buffer corresponding to the lower priority stream exhibits the same pattern as that the higher priority stream (as seen in case study 1). This demonstrates the effect of feedback control at one stream on the other streams that share the same resource.

Table 1. Total buffer space required for PE_3.

Total backlog of b_1 and b_2 (macroblocks)				
PE_3 Freq.	$B_1 = 1000$	$B_1 = 2000$	$B_1 = 3000$	No feedback
2.8 GHz	31170	29498	28043	14382
2.9 GHz	30490	28624	27169	12430
3.0 GHz	29619	28332	26347	10868
3.1 GHz	29359	27478	25825	9104
3.2 GHz	28603	26885	25439	7396

Table 1 reports the total maximum backlogs of the two buffers b_1 and b_2 in reference to the capacity of B_1 and the frequency of PE_3. The last column of the table gives the values for the case when there is no state dependency. It is observed consistently for all frequencies that when the capacity of B_1 is finite and blocking-write is imposed on the system, the total backlog at PE_3 is much larger than the computed backlog in the case of no state-dependency. Thus, assuming the buffers having infinite capacity may potentially lead to inaccurate results, as shown in this case study. It is therefore important for us to capture the state-dependency in the model and analysis of such systems.

For simplicity, we assume in both case studies that blocking-write is implemented only at a single buffer in the system. The analysis for the case when blocking-write is imposed on multiple buffers can be done similarly, except that one may additionally need to use Theorem 3 to compute the effective service curve offered by a processor to a stream. The same technique presented here can also be employed to analyze systems with more than two streams.

Additionally, we have also modeled the given case studies using the ECA and carried out the analysis using the SAL (Symbolic Analysis Laboratory) model checker [20]. It is observed from our experiments that the analysis time when using ECA is much slower than when using RTC. For example, in case study 1, using ECA takes in average more than 10 times as compared to using RTC. As the system becomes more complex, the speed up is near exponential on the number of PEs in the system. This shows that analyzing blocking-write using our method is considerably more efficient than using an explicit state-based model.

5 Concluding Remarks

We have formulated a method that enhances the original RTC framework by modeling and analyzing a variety of state-dependencies using a feedback control technique. Our analysis is purely functional and thereby avoids the state-space explosion problem faced by explicit state-modeling in automata-based approaches. Since a number of state-based components that were originally modeled by ECA can now be analyzed functionally, it will be meaningful to integrate our proposed method into the hybrid framework of RTC and ECA [18]. It will also be interesting to extend our technique to capture more complex state-dependencies, for instance synchronization between streams.

References

[1] Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[2] R. Henia A. Hamann M. Jersak R. Racu K. Richter R. Ernst. System level performance analysis - the symta/s approach. In *Computers and Digital Techniques*, 2005.

[3] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[4] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume LNCS 2050. Springer, 2001.

[5] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.

[6] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[7] C.-S Chang. *Performance Guarantees in Communication Networks*. Springer, 2000.

[8] E. Fersman, P. Krcál, P. Pettersson, and W. Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

[9] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science*, 354(2):301–317, 2006.

[10] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *CODES+ISSS*, 2007.

[11] R. Henia and R. Ernst. Scenario aware analysis for complex event models and distributed systems. In *RTSS*, 2007.

[12] R. Henia, R. Racu, and R. Ernst. Improved output jitter calculation for compositional performance analysis of distributed systems. In *IPDPS*, 2007.

[13] K. Huang, L. Thiele, T. Stefanov, and E. Deprettere. Performance analysis of multimedia applications using correlated streams. In *DATE*, 2007.

[14] S. Künzli, A. Hamann, R. Ernst, and L. Thiele. Combined approach to system level performance analysis of embedded systems. In *CODES+ISSS*, 2007.

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

[16] O. Moreira and M. Bekooij. Self-timed scheduling analysis for real-time applications. *EURASIP Journal on Advances in Signal Processing*, 2007.

[17] C. Norström, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *6th International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 1999.

[18] L. T. X. Phan, S. Chakraborty, P. S. Thiagarajan, and L. Thiele. Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In *IEEE Real-Time Systems Symposium (RTSS)*, 2007.

[19] J. Rox and R. Ernst. Modeling event stream hierarchies with hierarchical event models. In *DATE*, 2008.

[20] Symbolic Analysis Laboratory. `http://sal.csl.sri.com`.

[21] S. Schliecker, S. Stein, and R. Ernst. Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis. In *DATE*, 2007.

[22] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *39th Design Automation Conference (DAC)*, 2002.

[23] S. Tripakis, C. Pinello, A. Benveniste, A. Sangiovanni-Vincentelli, P. Caspi, and M. Di Natale. Implementing synchronous models on loosely time triggered architectures. *IEEE Transactions on Computers*, 2008.

[24] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems*, 29(2-3):205–225, 2005.

[25] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[26] E. Wandeler and L. Thiele. Workload correlations in multi-processor hard real-time systems. *Journal of Computer and System Sciences (JCSS)*, 73(2):207–224, 2007.

[27] M. Wiggers, M. Bekooij, P. Jansen, and G. Smit. Efficient computation of buffer capacities for multi-rate real- time systems with back-pressure. In *International Conference on Hardware/Software Codesign and System Synthesis*, 2006.