A Multi-Mode Real-Time Calculus

Linh T.X. Phan Samarjit Chakraborty P S Thiagarajan
Department of Computer Science, National University of Singapore
E-mail:{phanthix, samarjit, thiagu}@comp.nus.edu.sg

Abstract

The Real-Time Calculus (RTC) framework proposed in [Chakraborty et al.*, DATE 2003] and subsequently extended in [Wandeler* et al.*, Real-Time Systems 29(2-3), 2005] and a number of other papers is geared towards the analysis of real-time systems that process various types of streaming data. The main strength of RTC is a count-based abstraction, where arrival patterns of event streams are specified as constraints on the* number *of events that may arrive over any specified time interval. In this framework, algebraic techniques can be used to compute system properties in a compositional way. However, the main drawback of RTC is that it cannot model* state information *in a natural way. For example, when a scheduling policy depends on the fill-level of a certain buffer or there is a shift from one type of data stream into another. In this paper, we extend RTC in a manner that enables state information to be easily captured while limiting the state-space explosion caused by fine grained state-based models such as timed automata. Our model, called* multi-mode RTC*, specifies event streams as finite automata whose states are annotated with functions that specify constraints on the arrival patterns of event streams or the service available to process them. Our new framework combines the expressiveness of state-based models with the algebraic and compositional features of the RTC formalism. In particular, system properties within a single mode can be analyzed using the RTC-based algebraic techniques and state-space exploration can be used to piece together the results obtained algebraically for the individual modes. We show how to determine typical system properties with the focus on efficient approximate techniques and illustrate the advantages of multi-mode RTC using two case studies.*

1 Introduction

The increasing complexity of real-time embedded 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 that cannot be modeled using standard periodic/sporadic event models and fixed-priority or deadline-based scheduling policies.

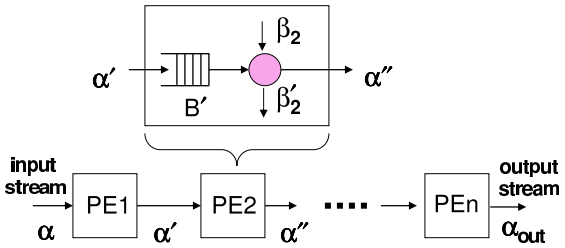

Figure 1: System model.

In this context, the Real-Time Calculus (RTC) framework was introduced in [4, 16] and subsequently extended in a number of other papers (e.g., see [17, 18]). RTC is designed to model and analyze heterogeneous real-time systems in a compositional manner. The key feature of RTC is that it relies on a *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 an event stream are specified as a constraint on the maximum and minimum number of events 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 lower- and upper-bounds on the event arrival process. Hence, $\alpha^l(\Delta)$ ($\alpha^u(\Delta)$) specifies the minimum (maximum) number of events 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, $\beta^l(\Delta)$ ($\beta^u(\Delta)$) will specify the minimum (maximum) number of events 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 – bounds on system properties such as the maximum delay suffered by the event stream and the maximum backlog of events 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* event 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 α_{out} of the output stream is computed (see Fig. 1).

In Fig. 1, $PE1, \ldots, PEn$ denote n resources which process an input stream in a pipelined fashion. Each PEi has an input buffer which stores the incoming events waiting to be processed. As explained above, the service provided by each PEi is bounded by β_i. Similar to α', it is possible to compute ${\beta_i}'$ which denotes bounds on the *remaining service*, that can be used to process other event/data streams. Apart from the buffer requirement and the delay suffered by the input stream at each resource PEi, it is also possible to compute the utilization of each resource, output jitter, the maximum end-to-end delay and the total buffer requirement in the system from the bounds α and α_{out}.

1.1 Our Contributions

Due to its functional nature, analysis in the RTC framework involves algebraic manipulations which allows very efficient computation of system properties in a fully compositional manner. However, it does not allow the modeling of *state information* in a natural way. As a result, 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. On the other hand, fine-grained modeling of state information, e.g. using timed automata [1, 7] or event count automata [5] will often lead to state space explosion when applied to realistic problems. To bridge this gap between expressiveness and ease of analysis, in this paper we extend the RTC framework to include state information in a natural way while retaining the fundamental link to RTC. Our model is a *multi-mode RTC* where the arrival process of an event stream or the availability of a resource are modeled as finite automata whose states are annotated with arrival/service functions. Our new framework combines the expressiveness of state-based models with the algebraic and compositional features of the RTC formalism. In particular, system properties within a single *mode* are analyzed using the RTC-based algebraic techniques, and state-space exploration is used to piece together the results obtained algebraically for the individual modes.

The system model in multi-mode RTC is the same as in the RTC framework (shown in Fig. 1). The key difference is that the functions α and β are now replaced by arrival and service automata $\mathcal{A}_{arr}$ and $\mathcal{A}_{serv}$ respectively. Each mode s in $\mathcal{A}_{arr}$ ($\mathcal{A}_{serv}$) is annotated with an arrival (service) function α_s (β_s) which represent bounds on the event arrivals (service rates) in that particular mode. The triggering of transitions in $\mathcal{A}_{arr}$ and $\mathcal{A}_{serv}$ will depend on the amount of time spent in the current mode and on external signals generated by the environment. Given $\mathcal{A}_{arr}$ and $\mathcal{A}_{serv}$ we show how the service automaton $\mathcal{A}'_{arr}$ that captures the timing properties of the processed stream and $\mathcal{A}'_{serv}$ that captures the bounds on the remaining service can be obtained. $\mathcal{A}'_{arr}$ can then be used as an input for the next resource, exactly as in the RTC framework and thus form the basis for a compositional analysis method. Similarly, $\mathcal{A}'_{serv}$ can be used to process other event/data streams. Further, we also show how $\mathcal{A}_{arr}$ and $\mathcal{A}_{serv}$ can be used to compute the buffer space required to store events waiting to be processed and the maximum delay suffered by the event stream. All of these questions involve the algebraic techniques used in RTC along with standard state-space exploration methods.

It may be noted that multi-mode RTCs are closely related in spirit to hybrid automata [9]. As a result, there is straightforward approach to obtaining the required analysis results by translation into event count automata (ECA) [5], which may be viewed as a simple form of hybrid automata. However, this will involve high computational costs since ECAs are a fine grained state-based model. Hence, we develop approximate analysis methods which are more efficient. Our methods are conservative over-approximations. For instance, the actual maximum buffer fill-levels are guaranteed to be less than the upper bounds we provide. We use standard RTC techniques to analyse the timing properties for a specific combination of modes of an arrival/service automaton and combine it with a state exploration techniques similar to the ones used for determining the boundedness of places in Petri nets [10]. We also identify the special case, where the synchronization between an arrival and service automaton is tightly bound with the help of external signals, so that their mode changes always take place together. In this, so called synchronous setting, our constructions are considerably simpler and the estimates become tighter. We also provide experimental validation of our analysis methods using two case studies.

1.2 Related Work

There are two broad lines of work that are related to the model proposed in this paper. The first is concerned with task and event models that generalize classical periodic or sporadic event models and assumptions on the fixed execution time requirements of tasks. Towards this, timed automata and related automata-theoretic formalisms have been recently used in a variety of setups to model and analyze task scheduling problems (e.g., see [1, 6, 7]).

To overcome the lack of state-based modeling in the RTC framework, we had proposed *event count automata* (ECAs) [5] that retain the count-based abstraction used in RTC, but at the same time allows for the detailed modeling of state information. ECAs, although syntactically similar to timed automata (and in fact hybrid automata), use *count* rather than clock variables that get incremented upon the arrival of events. However, they are semantically very different. In particular, in contrast to timed automata which – in this setting – explicitly record the arrival times of events, ECAs only record the *number* of events that arrive over any time interval. Often this is an adequate and more appropriate abstraction for event streams and service availability.

While ECAs are much more expressive than the arrival/service functions used in RTC, they suffer from the state explosion problem. Hence, it is difficult to analyze large system architectures when modeled as a network of ECAs. The *multi-mode RTC* framework that we propose in this paper can be seen as sitting in between the detailed state-based ECA modeling and the original RTC framework that is completely stateless. Thus, our formalism allows sufficient expressiveness, while mitigating the accompanying state-space explosion problem.

Yet another direction of related work focuses on extending models and schedulability analysis techniques from the real-time systems literature to accommodate more complex behaviors. For example, the framework presented in [3] allows certain tasks to intentionally change their execution periods, which is a type of mode change. The associated scheduling technique then adapts the periods of the other tasks within allowable limits in order to maintain a schedulable system. Similarly, the model proposed in [15] allows a system to be in multiple modes, where each mode consists of a set of tasks possibly overlapping with other modes. The system uses Rate Monotonic scheduling for all the modes. The problem is then to select suitable parameters for all the tasks, such that the system is schedulable in all modes.

Different *mode change protocols* have been studied in [8, 14] and have been classified in [13]. Here, again, a system consists of multiple modes, where each mode consists of a set of tasks. A mode change is triggered by a mode change request (MCR), and transitions from an old to a new mode take non-zero time. During this transition, the system has tasks from both the old and the new modes, which might produce a temporal overload. However, MCRs cannot arrive during a transition period. The goal is to develop techniques that ensure that no deadlines are violated during the transition periods. Such techniques consist of suitable mode change *protocols* (e.g. to restrict mode changes only at pre-specified time instants, or allow only synchronous mode changes), as well as analysis techniques to *verify* the feasibility of the system in the different modes and during the transition periods.

In contrast to the above approaches, we make the simplistic assumption of instantaneous mode changes. However, as mentioned before, the arrival and service functions associated with our modes allow arbitrary event streams and resource availability which cannot be modeled using standard event models. Finally, our compositional approach readily fits into the framework reported in [11] where a network of heterogenous components modeled as RTCs and event count automata were analyzed by computing suitable interfaces between components of different types. Our framework, however, is more uniform in the sense all the components and data streams are modeled as automata and the interfaces are also computed as (arrival) automata.

1.3 Organization of the Paper

In the next section we present our multi-mode RTC framework. In particular, we define arrival and service automata and their behaviors. In Section 3 we develop our analysis methods. After sketching the means for doing exact – but expensive – analysis, we present in detail our approximate techniques to compute maximum buffer fill-levels as well as the output data stream generated by a processing element in the form of an arrival automaton. In Section 4 we present experimental results using two case studies derived from an MPEG-2 decoder to validate the accuracy and the efficiency of our analysis methods. The concluding section discusses the prospects for extending the study of multi-mode RTC initiated in this paper.

2 Arrival and Service Automata

We now formulate a multi-mode version of arrival and service functions. In particular, a *service automaton* consists of a finite state automaton (FSA) in which each state is associated with a service function. Mode changes are effected by invariants associated with the modes as well as guards associated with the transitions. These invariants and the guards are constraints based on time intervals. In addition, we also use external signals and buffer fill-levels to specify the guards.

2.1 Arrival and Service Functions

As outlined earlier, the basic idea in Real-Time Calculus (RTC) is to bound the number of data items/events that can arrive or be serviced within a specified length of time. An arrival function α typically specifies, for a finite number of time interval lengths $\Delta_1, \ldots, \Delta_k$, the maximum number of events that can arrive in *any* time interval of length Δ_i. Such a specification captures a *class* of arrival sequences $x(t)$ that satisfy the following inequalities:

$$x(t + \Delta_i) - x(t) \leq \alpha(\Delta_i),\ \forall t \geq 0 \ \wedge\ 1 \leq i \leq k$$

In fact, in RTC , one often describes an arrival pattern via a pair of arrival functions (α^l, α^u) where the arrival pattern is understood to be bounded by the inequalities:

$$\alpha^l(\Delta_i) \leq x(t+\Delta_i) - x(t) \leq \alpha^u(\Delta_i),\ \forall t \geq 0 \wedge 1 \leq i \leq k$$

Similarly, to model the varying processor bandwidth made available by a resource, a pair of *service functions* $\beta = (\beta^l, \beta^u)$ can be used to specify lower and upper bounds on the number of events that can be processed by the resource within any time interval of a specified length. Let $c(t)$ denote the number of events that can be processed during the time interval $[0, t]$. Such a *service* is bounded by $\beta = (\beta^l, \beta^u)$ if the following inequalities hold.

$$\beta^l(\Delta) \leq c(t + \Delta) - c(t) \leq \beta^u(\Delta),\ \forall t \geq 0, \Delta \geq 0$$

Again, the bound on the service would typically be specified for a finite number of time interval lengths. Now,

given the bounds (α^l, α^u) on the arrival process of a stream and the service bounds (β^l, β^u) offered by a resource, [4] presented a calculus that allows us to compute (i) the minimum buffer size required at the input side of the resource, (ii) the maximum delay that can be suffered by an event, (iii) bounds on the timing properties of the *processed stream*, and (iv) bounds on the *remaining* service. Further, when a stream is processed by multiple resources (see Fig. 1), these bounds can be successively transformed and the timing properties of the fully processed stream can be computed in a compositional manner.

However, as pointed out earlier, the calculus presented [4] is purely functional and can not model setups where the arrival patterns and the processing of a stream depend on the *state* of the system.

2.2 Arrival Automata

An arrival automaton models an incoming event/data stream. It is a FSA where each state s has a pair of arrival functions (α_s^l, α_s^u) associated with it. When the automaton's current mode is s, the number of events that arrive in any time interval of length Δ is at least $\alpha_s^l(\Delta)$ and at most $\alpha_s^u(\Delta)$. There is too an invariant in the form of a time interval $[L_s, U_s]$ that specifies the minimum amount of time L_s the automaton must spend in s and the maximum amount of time U_s it may spend in s after entering it.

An input signal and a time interval $[L, U]$ is associated with each transition (s_i, s_j). The automaton can only take the transition on the arrival of the input signal a *and* provided it has been staying at s_i for at least L and no longer than U units of time. We assume that upon entering a new state, all the bounds given by the arrival functions associated with the previous state are forfeited; only constraints given by the arrival functions of the present state are applicable. In the formal definition of arrival automata that follows, $\mathbb{N}$ denotes the set of non-negative integers and INT is the set of finite intervals over $\mathbb{N}$. In other words, each member of INT is of the form $[L, U]$ with $L, U \in \mathbb{N}$ and $L \leq U$.

Definition 1 (Arrival automaton)**.** *An arrival automaton is a tuple* $\mathcal{A}_{arr} = \langle S, s_{in}, \Sigma, Inv, \alpha, \longrightarrow \rangle$ *where*

- S *is a finite set of states*
- $s_{in} \in S$ *is an initial state*
- Σ *is a finite set of signals*
- $Inv : S \rightarrow INT$ *is an invariant function associated with the states, where* $Inv(s) = [L_s, U_s]$.
- α *is an arrival function that assigns to each* $s \in S$*, a pair of arrival functions* (α_s^l, α_s^u) *giving the lower and upper arrival functions associated with* s.
- $\longrightarrow \subseteq S \times \Sigma \times INT \times S$ *is a transition relation.*

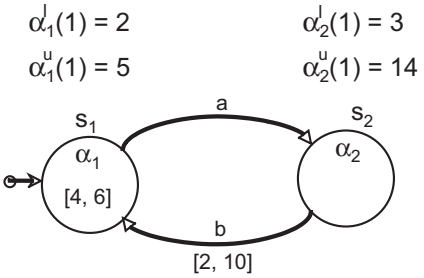

Figure 2: An arrival automaton.

An example of an arrival automaton is shown in Fig. 2. It captures an event stream whose arrival patterns are governed by two modes. In the first mode – which is also the initial one – at least 2 events and at most 5 events arrive in every time unit. In the second mode at least 3 events and at most 14 events arrive in every time unit. Accordingly, $\alpha_1 = (\alpha_1^l, \alpha_1^u)$ and there is just a single constraint, namely for the time interval 1 with $\alpha_1^l(1) = 2$ and $\alpha_1^u(1) = 5$. Similarly, $\alpha_2 = (\alpha_2^l, \alpha_2^u)$ with $\alpha_2^l(1) = 3$ and $\alpha_2^u(1) = 14$. Starting from $t = 0$, at some time during $[4, 6]$, events start to arrive at a faster rate as signalled by the external signal a presumably generated by the network interface or the downstream component generating this event stream. Notice that there is no timing guard associated with the transition from s_1 to s_2. Similarly, the state s_2 does not have an invariant associated with it. To comply with the formal definition, one can introduce and use the default guard $[0, \infty]$ for this purpose. To complete our example, once the automaton enters the mode s_2, after 2 to 10 units, the arrival rates slow down to the one associated the mode s_1 and the corresponding transition from s_2 to s_1 is signalled by the conjunction of the arrival of the signal b and the guard $[2, 10]$.

We assume that the transitions are executed in an *urgent* fashion. If the automaton is in the mode s at time t and one of the outgoing transitions of s is enabled then one of the enabled outgoing transitions of s *will* be taken at t. In case more than one transition is enabled, the choice of which transition to take will be made non-deterministically.

An arrival automaton specifies a language of arrival sequences. By an arrival sequence we mean a sequence of positive integers $c_1 c_2 \ldots c_i \ldots$ where c_i represents the number of events that have arrived during the unit time interval $[i-1, i)$. To bring this out, let $\alpha = (\alpha^l, \alpha^u)$ be an arrival function with α^l and α^u specifying lower and upper bounds for the set of time intervals $\{\Delta_1, \Delta_2, \ldots \Delta_k\}$. Then an arrival sequence *according* to α is a sequence of positive integers $c_1 c_2 \ldots c_n$ such that for each i and j in $\{1, 2, \ldots, n\}$, if $i < j$ and $j - i = \Delta_m$ for some $m \in \{1, 2, \ldots, k\}$, then

$$\alpha^l(\Delta_m) \leq (c_{i+1} + c_{i+2} + \cdots + c_j) \leq \alpha^u(\Delta_m).$$

Thus 3 2 5 is an event sequence according to α_1 specified above, while 3 4 6 is not.

Next let $\mathcal{A}_{arr}$ be an arrival automaton as in Definition 1. Then an *execution sequence* of this automaton is a finite sequence of the form:

$s_0 \overset{\sigma_{t_1}}{\dashrightarrow} s_0 \xrightarrow{a_1} s_1 \overset{\sigma_{t_2}}{\dashrightarrow} s_1 \xrightarrow{a_2} s_2 \ldots s_{n-1} \overset{\sigma_{t_{n-1}}}{\dashrightarrow} s_{n-1} \xrightarrow{a_n} s_n$

such that $s_0 = s_{in}$ and the following conditions are satisfied for each i with $0 < i \leq n$:

- t_i is a positive integer and σ_{t_i} is an arrival sequence according to the arrival function $\alpha_{s_{i-1}}$ associated with the mode s_{i-1}.
- $t_i \in Inv(s_{i-1})$
- There exists a transition $(s_{i-1}, a_i, I, s_i) \in \longrightarrow$ such that $t_i \in I$.

We say that $\sigma_{t_1}\sigma_{t_2}\ldots\sigma_{t_n}$ is the arrival sequence induced by the above execution sequence. The set of all such arrival sequences is the language of arrival sequences specified by the arrival automaton. For the example shown in Fig. 2, 3 2 5 4 7 11 12 2 is an arrival sequence corresponding to the execution sequence $s_1 \overset{3\,2\,5}{\dashrightarrow} s_1 \xrightarrow{a} s_2 \overset{4\,7\,11\,12}{\dashrightarrow} s_2 \xrightarrow{b} s_1 \overset{2}{\dashrightarrow} s_1$. On the other hand, 3 9 7 11 2 5 4 is not an arrival sequence because in the second time unit, the system must be at s_1 and hence there cannot be 9 events that arrive.

2.3 Service Automata

A processing resource is modeled as a service automaton. Its syntax and semantics are identical to that of an arrival automaton except that the number of events that arrive in a unit time interval is now interpreted as the number of processor cycles (in terms of the number of events that can be processed) that are available in this time interval. In addition, the transitions can also have guards specifying linear constraints on the fill-levels of the input and output buffers associated with the processing element. In the present instance, where we do not consider multiple data streams, there is just one associated input buffer and one output buffer as shown in Fig. 1. In the general case, there is one input-output buffer pair for each stream.

Assume a finite set of (positive) integer valued variables $\mathcal{B}$ standing for buffers with B ranging over $\mathcal{B}$.
Buffer constraints: A buffer constraint is a conjunctive formula of atomic constraints of the form $B \sim n$ with $B \in \mathcal{B}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. We use Φ_B to denote the set of buffer constraints, ranged over by φ.

Definition 2 (Service automaton)**.** *A service automaton is a tuple $\mathcal{A}_{serv} = \langle S, s_{in}, \Sigma, Inv, \beta, \Phi_B, \longrightarrow \rangle$ where*

- *S is a finite set of states*
- *$s_{in} \in S$ is an initial state*
- *Σ is a finite set of signals*
- *$Inv : S \rightarrow INT$ is an invariant function associated with the states, where $Inv(s) = [L_s, U_s]$.*
- *β is an arrival function that assigns to each $s \in S$, a pair of service functions (β^l_s, β^u_s) giving the lower and upper service functions associated with s.*
- *$\longrightarrow \subseteq S \times \Sigma \times INT \times \Phi_B \times S$ is a transition relation.*

The new feature is that a mode change can be also be constrained by the current fill-levels of the input and output buffers. For defining execution sequences, we need to track the current number of events in the buffers. This however requires information about the arrival processes which are depositing events into the input buffers and the service processes that are removing events from the output buffers. Hence, we do not work out the details here.

3 Analysis Techniques

We now consider the multi-stage processing of an event stream as shown in Fig. 1. Our objective is to analyze the behavior of the system and determine properties such as the maximum fill-level of the various buffers, the end-to-end delay experienced by the input event stream, the timing characteristics of the output event stream and the remaining resource at each stage after processing the event stream. We start with the first stage where the input stream modeled as the arrival automaton $\mathcal{A}_{arr}$ deposits events into the input buffer B of $PE1$ which is then processed by the processing element $PE1$. The service provided by $PE1$ in terms of the number of events can be processed per unit time is modeled by the service automaton $\mathcal{A}_{serv}$. We wish to estimate the maximum fill-levels of the input buffer B and compute the characteristics -in the form of an arrival automaton- of the output stream being deposited into the buffer B'. We first sketch how these properties can be computed exactly. In [5] we showed how *Event Count Automata (ECAs)*, can be used to model arrival and service functions. Indeed, for each arrival function we can effectively construct an ECA such that the language of arrival sequences accepted by the ECA is exactly the set of arrival sequences defined by the arrival function. Similar assertions hold for service functions as well. It is easy to extend these constructions to arrival and service automata. Thus for each arrival (service) automaton we can effectively construct an ECA such that the two mechanisms define the same set of arrival (service) sequences. Furthermore, we also showed in [5] how a network of $ECAs$ can be translated into a Petri net such that the set of reachable markings corresponds to the set of reachable configurations of the network of ECAs. An exhaustive analysis of this Petri net can reveal whether any of the buffers can become unbounded. One can also determine -when all the buffers are bounded- the sequences of data streams being deposited into the internal and output buffers of the network. We can use these constructions involving networks of ECAs and Petri nets to determine the properties we are after. However this a computationally expensive route and scalability is difficult to achieve. Hence we propose below a more viable but conservative approximate scheme.

3.1 Worst-Case Analysis

Let $\mathcal{A}_{arr} = \langle S, s_{in}, \Sigma, Inv, \alpha, \longrightarrow \rangle$ be an arrival automaton describing the data stream being deposited into the input buffer B of $PE1$. Let $\mathcal{A}_{serv} = \langle S', s'_{in}, \Sigma', Inv', \beta, \Phi_B, \longrightarrow' \rangle$ be a service automaton describing the process cycles provided by $PE1$ that processes the events arriving at B. We first describe a method for computing -approximately- the maximum fill-level of B. This will then lead to the method for constructing an arrival automaton that captures the processed data stream being deposited into the output buffer B'. To start with, suppose we are given an arrival function α which specifies constraints for the temporal lengths $\{\Delta_1, \Delta_2, \ldots, \Delta_k\}$. Suppose further we are given a time interval $I = [L, U]$. Then we can extend α with constraints for each $t \in I$ as follows:

- $\alpha(t) = \alpha(\Delta_1)$, if $t \leq \Delta_1$;
- $\alpha(t) = \min\{\alpha(\Delta_i), \min\limits_{0 \leq d \leq t} \{\alpha(d) + \alpha(t-d)\}\}$, if $\Delta_{i-1} < t \leq \Delta_i$;
- $\alpha(t) = \min\limits_{0 \leq d \leq t} \{\alpha(d) + \alpha(t-d)\}$, if $t > \Delta_k$.

Thus we allow as many items as possible to arrive in a time interval of length t subject to the constraints specified by α being satisfied. A similar extension is defined for the service function β whose constraints are defined for the set of temporal lengths $\{\Delta'_1, \Delta'_2, \ldots, \Delta'_m\}$:

- $\beta(t) = 0$, if $0 \leq t < \Delta'_1$;
- $\beta(t) = \beta(\Delta'_i)$, if $t = \Delta'_i$, for $1 \leq i \leq k$;
- $\beta(t) = \max\limits_{0 \leq d \leq t} \{\beta(d) + \beta(t-d)\}$, if $\Delta'_{i-1} < t < \Delta_i$ or $t > \Delta'_m$.

Thus we permit as few service cycles as possible to be provided in a time interval of length t subject to the constraints specified by β being satisfied.

We can now turn to the problem of computing the maximum fill-level of the buffer B. We shall construct $\mathcal{T}$, a finite tree whose nodes are of the form $v = (s, s', [L_\pi, U_\pi], \pi, B_\pi)$ with $s \in S$, $s' \in S'$ and $[L_\pi, U_\pi] \in INT$. Further, π is the path from the root to v, L_π and U_π are the earliest and latest time instants at which the system enters v, and B_π is the maximum buffer fill-level when the system enters v. We will compute for each π the functions $\alpha_\pi = (\alpha^l_\pi, \alpha^u_\pi)$ and $\beta_\pi = (\beta^l_\pi, \beta^u_\pi)$ where $\alpha^l_\pi(\Delta)$ ($\beta^l_\pi(\Delta)$) and $\alpha^u_\pi(\Delta)$ ($\beta^u_\pi(\Delta)$) are the minimum and maximum number of events that arrive (can be processed) in the last Δ units of time before the system enters v, for $0 \leq \Delta \leq U_\pi$. It then follows that $B_\pi \leq bufmax(\pi)$, where

$$bufmax(\pi) \stackrel{\text{def}}{=} \max_{0 < \Delta \leq U_\pi} \{\alpha^u_\pi(\Delta) - \beta^l_\pi(\Delta)\} \qquad (1)$$

The set of nodes of $\mathcal{T}$ we wish to construct will be denoted as $\mathcal{C}$ and its edge relation by $\Rightarrow$. It will be convenient to define the transition relation $\dashrightarrow \subseteq (S \times S') \times (\Sigma \cup \Sigma' \times INT) \times \Phi(B) \times (S \times S')$:

- Suppose $s \stackrel{a,I}{\longrightarrow} s_1$ and $a \in \Sigma - \Sigma'$ and $s' \in S'$. Then $(s, s') \stackrel{a,I}{\dashrightarrow} (s_1, s')$.
- Suppose $s' \stackrel{a',I',\varphi'}{\longrightarrow'} s'_1$ and $a' \in \Sigma' - \Sigma$ and $s \in S'$. Then $(s, s') \stackrel{a,I',\varphi}{\dashrightarrow} (s, s'_1)$.
- Suppose $s \stackrel{a,I}{\longrightarrow} s_1$ and $s' \stackrel{a,I',\varphi'}{\longrightarrow'} s'_1$ and $a \in \Sigma \cap \Sigma'$. Then $(s, s') \stackrel{a,I\cap I',\varphi'}{\dashrightarrow} (s_1, s'_1)$. (if $I = [L, U]$ and $I' = [L', U']$, then $I \cap I' = [\max\{L, L'\}, \min\{U, U'\}]$).

In this paper we focus on the synchronous case where an arrival automaton and a service automaton communicating through a buffer must always make their moves together. The analysis for the general setting where arrival and service automata may make their transitions asynchronously is done similarly with a slight modification in the computation of the functions associated with each node. The details may be found in [12].

We start with $v_{in} = (s_{in}, s'_{in}, [0, 0], \epsilon, 0) \in \mathcal{C}$. As usual, ϵ denotes the null path. We declare v_{in} to be a *non-terminal* node. Furthermore, it is the root of $\mathcal{T}$. Since no event has arrived, $\alpha_\epsilon = \beta_\epsilon = 0$.

Suppose $v = (s, s', [L_\pi, U_\pi], \pi, B_\pi) \in \mathcal{C}$ is a non-terminal node with $\alpha(s) = (\alpha^l, \alpha^u)$ and $\beta(s') = (\beta^l, \beta^u)$. Assume inductively that $\alpha_{\pi'}$ and $\beta_{\pi'}$ have been defined for all sub-path π' of π that starts from v_{in}. Suppose $\tau = (s, s') \stackrel{a,I,\varphi}{\dashrightarrow} (s_1, s'_1)$ with $a \in \Sigma \cap \Sigma'$. The path to (s_1, s'_1) is then $\pi\tau$. Let $[L, U] = Inv(s) \cap Inv'(s') \cap I$. Roughly speaking, $[L, U]$ is the interval in which, if the signal a arrives, then the transition $(s, s') \stackrel{a,I,\varphi}{\dashrightarrow} (s_1, s'_1)$ can be taken. Thus, the earliest and latest instants at which τ is taken are $L_\pi + L$ and $U_\pi + U$. Since (i) there are at least $\alpha^l_\pi(x)$ and at most $\alpha^u_\pi(x)$ events that arrive in the last x units of time before the system enters v and (ii) the system must stay in v for at least L time units, the minimum and maximum number of events that arrive in the last Δ time units before the system enters (s_1, s'_1) are computed for all $0 \leq \Delta \leq U_\pi + U$ as follows:

- If $\Delta \leq L$, $\alpha^l_{\pi\tau}(\Delta) = \alpha^l(\Delta)$ and $\alpha^u_{\pi\tau}(\Delta) = \alpha^u(\Delta)$.
- Otherwise,

$$\alpha^l_{\pi\tau}(\Delta) = \min_{\substack{0 \leq x \leq U_\pi \\ L < \Delta - x \leq U}} \{\alpha^l_\pi(x) + \alpha^l(\Delta - x)\}$$

$$\alpha^u_{\pi\tau}(\Delta) = \max_{\substack{0 \leq x \leq U_\pi \\ L < \Delta - x \leq U}} \{\alpha^u_\pi(x) + \alpha^u(\Delta - x)\}$$

Similarly, we obtain $\beta_{\pi\tau}$ by substituting α with β in the above formulas. In addition, as the maximum buffer fill-level when the system enters v is B_π, the buffer fill-level when the system enters (s_1, s'_1) is at most

$$buf(B_\pi, s, s') \stackrel{\text{def}}{=} \max_{0 \leq \Delta \leq U} \{B_\pi + \alpha^u(\Delta) - \beta^l(\Delta)\}$$

Suppose B_φ is the largest value of B that satisfies the buffer constraint φ. Combine with Eq. 1, we obtain

$$B_{\pi\tau} = \min(buf(B_\pi, s, s'), bufmax(\pi\tau), B_\varphi).$$

Let Δ_1 be the smallest interval such that $\alpha^u_{\pi\tau}(\Delta_1) - \beta^l_{\pi\tau}(\Delta_1)$ satisfies φ and Δ_2 be the largest interval such that $\alpha^u_{\pi\tau}(\Delta_2) - \beta^l_{\pi\tau}(\Delta_2)$. Then, at all time points before Δ_1 and after Δ_2, the transition τ cannot be taken. Define $[L_{\pi\tau}, U_{\pi\tau}] = [L_\pi + L, U_\pi + U] \cap [\Delta_1, \Delta_2]$. Then $L_{\pi\tau}$ and $U_{\pi\tau}$ are the earliest and latest instant at which the system enters (s_1, s'_1).

We now add $v_1 = (s_1, s'_1, [L_{\pi\tau}, U_{\pi\tau}], \pi\tau, B_{\pi\tau})$ as a node in $\mathcal{C}$ and (v, v_1) as an edge in $\Rightarrow$. We declare v_1 to be an *un-bounded terminal* node in case there is an ancestor node u of v_1 in $\mathcal{T}$ with $u = (s_1, s'_1, [L_\sigma, U_\sigma], \sigma, B_\sigma)$ such that (i) none of the guards associated with the transitions from u to v_1 contains $B \leq n$ or $B < n$ for $n \in \mathbb{N}$ and (ii) $K > 0$ where $K = \max_{\Delta > L_{\pi\tau} - L_\sigma} \{\alpha^u_{\pi\tau}(\Delta) - \beta^l_{\pi\tau}(\Delta)\} - B_\sigma$. In case (i) does not satisfy or $K \leq 0$ we declare it to be a *normal terminal* node. If v_1 does not have an ancestor node with the same state components, then it is deemed to be a *non-terminal* node.

Note in the above that if $K > 0$, as the system cycles through the path from u to v (denoted as ρ), the maximum buffer fill-level each time the system enters (s_1, s'_1) is increased by at least K. If none of the buffer guards of the transitions in ρ impose upper bounds on the fill-level (i.e. condition (i) holds), by cycling through ρ an infinite number of times, the maximum buffer fill-level when the system enters (s_1, s'_1) will become unbouded. On the other hand, if $K \leq 0$, the maximum buffer fill-level as well as the arrival and service functions associated with the paths that repeat ρ one or more time will be bounded by that of σ. Similarly, if (i) is not satisfied, these functions are bounded by the functions of $\tau\pi$. Hence, we do not need to consider the successor nodes of v_1.

We then consider the next transition from v or mark v as *visited* if all out-going transitions from v have already been explored. We repeat the above process until all non-terminal nodes in $\mathcal{C}$ are visited.

We claim that $\mathcal{T}$ is a finite tree whose construction terminates after a finite number of steps. Suppose otherwise, there exists an infinite path π in $\mathcal{T}$. Since S and S' are finite, there are $s \in S$ and $s' \in S'$ such that (s, s') appears infinitely often in π. This contradicts to the construction that every node v of π with the same state components as that of its ancestor is a terminal node and is present exactly once. Hence, $\mathcal{T}$ is finite. Each non-terminal node in $\mathcal{C}$ is marked as visited exactly once, hence the construction terminates after a finite number of steps.

Buffer fill-level analysis: To compute the maximum buffer fill-level of B, we declare the buffer to be unbounded if there is an unbounded terminal node in $\mathcal{T}$. If not, we first compute the maximum fill-level $maxbuf(v)$ attained at each node v as follows. Suppose $v = (s, s', [L_\pi, U_\pi], \pi, B_\pi)$ with $\alpha(s) = (\alpha^l, \alpha^u)$, $\beta(s') = (\beta^l, \beta^u)$, $Inv(s) = [L_s, U_s]$ and $Inv'(s') = [L_{s'}, U_{s'}]$. The maximum number of events that arrive in any interval of length Δ that ends when the system is at v is

$$\alpha^u_v(\Delta) = \max_{0 \leq x \leq U_\pi} \{\alpha^u_\pi(x) + \alpha^u(\Delta - x) \mid 0 \leq \Delta - x \leq U_v\}$$

where $U_v = \min(U_s, U_{s'})$. This comes from the conditions that at most $\alpha^u_\pi(x)$ events arrive in the last x units of time before the system enters v and at most $\alpha^u(\Delta - x)$ events arrive in $\Delta - x$ time units while the system is at v. Analogously, the minimum number of events that can be processed during any interval of length Δ that ends when the system is at v is

$$\beta^l_v(\Delta) = \min_{0 \leq x \leq U_\pi} \{\beta^l_\pi(x) + \beta^l(\Delta - x) \mid 0 \leq \Delta - x \leq U_v\}$$

Thus, the buffer fill-level when the system is at v is at most $maxbuf'(v) = \max_{0 \leq \Delta \leq U_\pi + U_v} \{\alpha^u_v(\Delta) - \beta^l_v(\Delta)\}$. In addition, since B_π is the maximum buffer fill-level when the system enters v, the buffer fill-level at v is at most $maxbuf''(v) = \max_{0 \leq \Delta \leq U_v} \{B_\pi + \alpha^u(\Delta) - \beta^l(\Delta)\}$. As a result, the maximum buffer fill-level of B is $maxbuf(v) = \max(maxbuf'(v), maxbuf''(v))$. It follows that the maximum buffer fill-level of B is the maximum of $maxbuf(v)$ for all $v \in \mathcal{C}\}$.

End-to-end delay analysis: We compute the maximum end-to-end delay of the data stream based on $\mathcal{T}$. At any instant t, the system is in some node $v = (s, s', [L_\pi, U_\pi], \pi) \in \mathcal{C}$ with $L_\pi \leq t \leq U_\pi + U_v$. For all $0 \leq \Delta \leq t$, the maximum number of events that arrive in $[t - \Delta, t]$ is $\alpha^u_v(\Delta)$ and the minimum number of events that can be processed in $[t - \Delta, t + d]$ is $\beta^l_{v'}(\Delta + d)$ for $v' = v$ or $v' = (s_1, s'_1, [L_{\pi'}, U_{\pi'}], \pi') \in \mathcal{C}$ such that $\pi \subset \pi'$ and $L_{\pi'} - L_\pi \leq d$. Let $succ(v, d)$ be the set of all such v'. Thus the maximum delay of an event that arrive at t is the minimum of d such that $\alpha^u_v(\Delta) \leq \beta^l_{v'}(\Delta + d)$ for all $0 \leq \Delta \leq t$ and $v' \in succ(v, d)$. As a result, the maximum end-to-end delay of any event that arrives when the system is at a node v is $delay(v) = \min\{d \mid \alpha^u_v(\Delta) \leq \beta^l_{v'}(\Delta + d) \ \forall\ 0 \leq \Delta \leq U_\pi + U_v \wedge \forall\ v' \in succ(v, d)\}$. The maximum end-to-end delay of the event stream is then the maximum of $delay(v)$ for all $v \in \mathcal{C}$.

Arrival automaton of the output stream: When the maximum buffer fill-level of the system is bounded, we can compute an arrival automaton representation of the processed data stream being deposited into the output buffer B' (see Fig. 1). To this end, we derive α^l_v and β^u_v, which give the lower bound on the number of events that arrive and the upper bound on the number of events that can be processed in any interval of a given length Δ that ends when the system is

at v for all $v \in \mathcal{C}$. This is done using similar arguments that derive α_v^u and β_v^l described above. We then compute for all $v \in \mathcal{C}$ the arrival function $\alpha'_v = (\alpha_v^{l'}, \alpha_v^{u'})$ that bounds the processed event stream when the system is in v as below.

Let $v \in \mathcal{C}$. At any instant t when the system is at v, the number of events that are processed in $[t-\Delta, t)$ is at most $\beta_v^u(\Delta)$. Moreover, it is upper bounded by the maximum number of events that are processed in $[t-\Delta-\lambda, t)$ subtracted by the minimum number of events that are processed in $[t-\Delta-\lambda, t-\Delta)$ for all $\lambda \geq 0$ and $\lambda \leq t-\Delta$. First, observe that at time $t-\Delta$, the system is either at v or some ancestor node $v' = (s_1, s'_1, [L_{\pi'}, U_{\pi'}], \pi', B_{\pi'})$ of v such that $L_\pi - L_{\pi'} \leq \Delta$. Thus, the minimum number of events that are processed in $[t-\Delta-\lambda, t-\Delta)$ is $\beta_{v'}^l(\lambda)$. Secondly, the maximum number of events that are processed in $[t-\Delta-\lambda, t)$ does not exceed $\alpha_v^u(d) + \beta_{v''}^u(\Delta+\lambda-d)$ for all $0 \leq d \leq \Delta+\lambda$ and $v'' = (s_2, s'_2, [L_{\pi''}, U_{\pi''}], \pi'', B_{\pi''})$, $\pi' \subseteq \pi'' \subseteq \pi$ and $L_\pi - L_{\pi''} \leq d$. As a result,

$$\alpha_v^{u'}(\Delta) = \min\Bigg\{ \max_{\substack{\lambda \geq 0 \wedge \pi_{v'} \subseteq \pi_v \\ L_{\pi_v} - L_{\pi_{v'}} \leq \Delta}} \Big\{ \min_{\substack{\pi_{v'} \subseteq \pi_{v''} \subseteq \pi_v \\ L_{\pi_v} - L_{\pi_{v''}} \leq d \leq \Delta+\lambda}} \{\alpha_v^u(d) + \beta_{v''}^u(\Delta+\lambda-d)\} - \beta_{v'}^l(\lambda) \Big\},\ \beta_v^u(\Delta) \Bigg\}$$

where π_v denote the path that leads to v in $\mathcal{T}$. Similarly, we can also compute $\alpha_v^{l'}$ and β'_v for all $v \in \mathcal{C}$.

Now let $V_{s,s'}$ be the set of all nodes v in $\mathcal{C}$ that contain (s, s') as their state components. The set of states $\mathcal{S}''$ of the arrival automaton $\mathcal{A}'_{arr}$ is the set of all (s, s') such that $V_{s,s'}$ is non-empty, with (s_{in}, s'_{in}) being the initial state. The invariant associated with each state $(s, s') \in \mathcal{S}''$ is $[L_{s,s'}, U_{s,s'}] = Inv(s) \cap Inv'(s')$. The arrival functions associated with (s, s') is $\alpha(s, s') = (\alpha^l_{s,s'}, \alpha^u_{s,s'})$ with $\alpha^l_{s,s'}(\Delta) = \min\{\alpha_v^{l'}(\Delta) \mid v \in V_{s,s'}\}$ and $\alpha^u_{s,s'}(\Delta) = \max\{\alpha_v^{u'}(\Delta) \mid v \in V_{s,s'}\}$ for all $\Delta \in [0, U_{s,s'}]$. The transition relation $\longrightarrow$ of $\mathcal{A}'_{arr}$ is defined using the edge relation of $\mathcal{T}$ and the transition relation $\dashrightarrow$. Specifically, suppose $(s, s') \overset{a, I', \varphi}{\dashrightarrow} (s_1, s'_1)$ is a transition in $\dashrightarrow$ such that the set of all edges of $\mathcal{T}$ that are constructed using this transition is not empty. Then $\big((s, s'), a, I', (s_1, s'_1)\big)$ is added to the transition relation of $\mathcal{A}'_{arr}$ with $I' = I \cap Inv(s) \cap Inv'(s')$.

We note that once $\mathcal{A}'_{arr}$ has been constructed, it can be combined with $\mathcal{A}^2_{serv}$, the service automaton capturing the service provided by $PE2$ to compute the maximum buffer fill-level of the output buffer B'. Thus the system level analysis of the whole cascade can be carried out in a compositional way.

Service automaton of the remaining service curve: The service automaton $\mathcal{A}'_{serv}$ that represents the residual service cycles of $PE1$ after processing the data stream arriving at B is the same as $\mathcal{A}'_{arr}$, except that each state (s, s') is associated with a service function $\beta_{s,s'}$ which is computed similarly to the computation of $\alpha_{s,s'}$. Furthermore, each transition $\big((s, s'), a, I', (s_1, s'_1)\big)$ is additionally guarded with a buffer constraint φ that comes from $(s, s') \overset{a, I, \varphi}{\dashrightarrow} (s_1, s'_1)$.

It can be derived from our analysis that the bounds we obtain are conservative ones in that the actual bounds are less than or equal to the ones we compute.

4 Case Studies

We present two case studies to illustrate the effect of mode switching on the timing properties of a system. These case studies also serve to demonstrate that our approach improves the performance estimates of realistic stream processing systems. In the first study, we analyze an MPEG-2 decoder in which the processor operates at two different frequencies. The mode switchings between the two frequencies are steered by the fill-levels of the input buffer. In the second study, we examine a system where its resource allocation is adapted to the arrival rates of the incoming event stream. In both case studies, we analyze the maximum buffer fill-level of the buffer as well as the end-to-end delay of the input stream and compared the results against the monolithic RTC framework. Additionally, we use Event Count Automata to compute the optimal solutions in order to compare the relative accuracy of our and the RTC frameworks.

4.1 Case Study 1: An MPEG-2 Decoder

Fig. 3 shows an MPEG-2 decoder application that is partitioned and mapped onto two processing elements $PE1$ and $PE2$. The processor $PE1$ runs the VLD and IQ tasks while $PE2$ runs the IDCT and MC tasks. The (coded) input bitstream enters this system and it is stored in the input buffer B. The macroblocks in B are first processed by $PE1$ and the corresponding partially decoded macroblocks are stored in the buffer B' before being processed by $PE2$. The resulting stream of fully decoded macroblocks is written into a playout buffer B'' prior to transmission by the output video device. In the above system, the coded input event stream

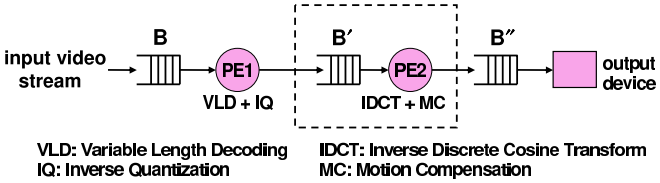

Figure 3: An MPEG-2 decoder application.

arrives at a constant bit-rate. $PE1$ operates at two different frequencies depending on the state of the input buffer B. It initially starts with frequency f_0 and changes to a higher frequency f_1 whenever the fill-level of buffer B is more than 50% of its capacity. $PE1$ is guaranteed to maintain the frequency f_1 for up to T time units before switching back to

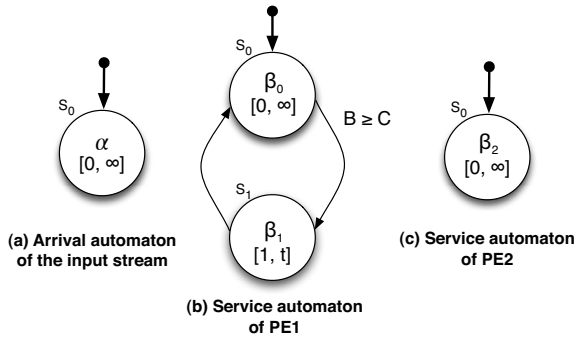

Figure 4: The multi-mode RTC model of the decoder in Fig. 3

.

frequency f_0 while $PE2$ is assumed to operate at a fixed frequency f_2. In this system, we are interested in computing the maximum fill-level of the buffer B' given the rate of the incoming input stream, the clock frequencies f_0, f_1 and f_2 of the two processors as well as the capacity of B.

4.1.1 Obtaining Arrival and Service Functions

We collected execution traces of the different tasks by simulating their execution using a customized version of the SimpleScalar instruction set simulator [2]. From these traces, we measured the execution demands of the VLD and IQ tasks for each macroblock in a video sequence while considering (i) the constant arrival rate of the compressed bit stream at the input of $PE1$ and (ii) the number of bits allocated to encode each macroblock in the stream. Based on these execution demands, we derive a function $x(t)$ which gives the number of macroblocks arriving at B during the time interval $[0, t]$. From $x(t)$, we computed the arrival function $\alpha = (\alpha^l, \alpha^u)$ of the input event stream according to the definition of an arrival function in Section 2.1.

Similarly, based on the obtained execution demands of the VLD and IQ tasks for each macroblock, we computed the workload function γ at the first PE, where $\gamma(k)$ gives the minimum and maximum number of cycles required by any k consecutive macroblocks. By combining γ and the frequencies f_0 and f_1, we deduce the corresponding service functions $\beta_0 = (\beta_0^l, \beta_0^u)$ and $\beta_1 = (\beta_1^l, \beta_1^u)$ for $PE1$. The service function $\beta_2 = (\beta_2^l, \beta_2^u)$ that represents the processing capability of $PE2$ is computed analogously using the execution demands of the IDCT and MC tasks for each macroblock and the frequency f_2.

4.1.2 Using Multi-mode RTC

The multi-mode RTC model of the application is depicted in Fig. 4. The arrival automaton $\mathcal{A}_{arr}$ of the event stream has a single mode with no out-going transitions. This mode is associated with an invariant $[0, \infty]$ and the arrival function α calculated above.

The service automaton $\mathcal{A}_{serv}$ that models $PE1$ has two modes s_0 and s_1 associated with invariants $[0, \infty]$ and $[1, T]$ respectively. The corresponding service functions of these modes are the obtained β_0 and β_1. The transition from s_0 to s_1 is guarded by a buffer constraint $B \geq C$, with C being half the capacity of B. As soon as there are at least C events in the buffer B, the automaton will switch from s_0 to s_1 and will stay there for at most T unit of times. Unlike $PE1$, the second PE has only a single mode that with the associated service function being β_2 and the invariant being $[0, \infty)$.

To analyze the fill-level of buffer B', we first compute the arrival automaton $\mathcal{A}'_{arr}$ for the output event stream generated by $PE1$. This is obtained from $\mathcal{A}_{arr}$ and $\mathcal{A}_{serv}$ with the help of the analysis technique described in Section 3. The resulting $\mathcal{A}'_{arr}$ is next composed with the service automaton $\mathcal{A}'_{serv}$ of the second PE to derive the maximum fill-level of buffer B'.

4.1.3 Using Standard RTC

Following the standard RTC approach, the event stream is modeled as $\alpha = (\alpha^l, \alpha^u)$, the arrival function described above. Since mode switching is not explicitly modeled in the RTC framework, the maximum resource offered by $PE1$ is assumed to be the upper bound of β_0^u and β_1^u whereas the minimum resource offered by $PE1$ is assumed to be the lower bound of β_0^l and β_1^l. In other words, the service offered by $PE1$ is modeled as the service function $\beta = (\beta^l, \beta^u)$ such that $\beta^u(\Delta) = \max\{\beta_0^u(\Delta), \beta_1^u(\Delta)\}$ and $\beta^l(\Delta) = \min\{\beta_0^l(\Delta), \beta_1^l(\Delta)\}$ for all $\Delta \geq 0$. As might be expected, $PE2$ is modeled as the service function β_2. Our analysis of this RTC model follows the standard methods associated with the RTC framework [4].

4.1.4 Analysis Results

Fig. 5 reports the estimated maximum fill-level of the buffer B as we increase the frequencies of f_0 for (i) our multi-mode RTC method, (ii) the standard RTC method, and (iii) the exact method using ECA. The frequency f_1 is chosen to be $f_0 + 100$ (Mhz) and the frequency f_2 is 600 (Mhz). The results shown here are obtained for a fixed value of T. However, our experiments with other values of T show similar performance patterns for the three methods. As shown in the figure, the fill-level of the buffer get smaller as we move from left to right. This reflects the expected behavior of the system in that, given a fixed input stream, the fill-level of an input buffer is a decreasing function of the processor frequency. Observe that the maximum fill-levels of the buffer corresponding to the multi-mode RTC is the same as the ones corresponding to the ECA method. These values are always smaller than that of the standard RTC method. This illustrates the chain effect of mode switching in $PE1$ on the output stream and subsequently the buffer B'. By capturing this characteristic of the system, our multi-mode RTC method is able to achieve the optimal results (as given by

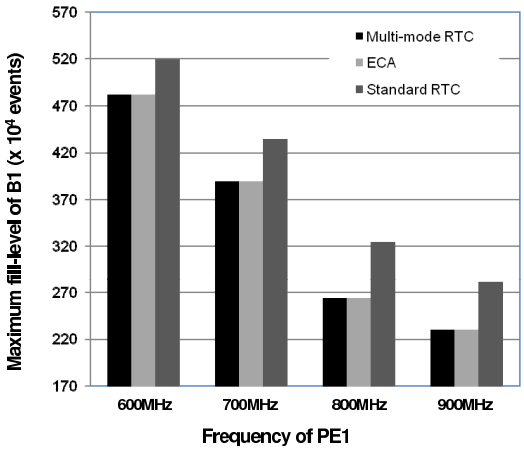

Figure 5: Case study 1: Maximum buffer fill-level analysis.

the ECA method). On the other hand, the standard RTC's estimates are more pessimistic.

4.2 Case Study 2

In this case study, we examined a system in which the resource offered by the processors is adapted to changes in the average arrival rate of the incoming event stream using common signals. The system has the same architecture as the previous case study with two PEs processing an event stream sequentially. The event stream arrives at two different rates R_1 and R_2. The stream arrives at the rate R_i for a minimum of L_i and a maximum of U_i units of time before it switches to the other rate. Whenever the input interface detects this change from one rate to another, it informs the processor by emitting a signal. Upon receiving this signal, the processor will switch to the corresponding level of service. Fig. 6 shows the arrival automaton of the event

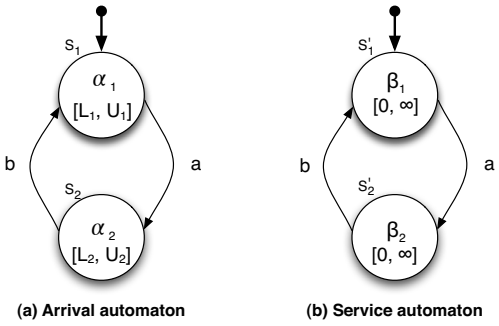

Figure 6: The multi-mode RTC model of Case study 2.

stream and the service automaton of the first PE. As shown in the figure, the arrival automaton has two modes s_1 and s_2 which correspond to the two arrival rates R_1 and R_2, with $R_1 < R_2$. Each mode s_i is associated with an arrival function $\alpha_i = (\alpha_i^l, \alpha_i^u)$ that is obtained based on the rate R_1 using the method described in case study 1. Additionally, s_i is associated with an invariant $[L_i, U_i]$ which gives the lower and upper bounds on the amount of time the automaton can stay in this mode. A transition of the automaton from s_1 to s_2 and vice versa are indicated by the signals a and b. Accordingly, the service automaton changes between two modes s'_1 and s'_2, where s'_1 has the associated service function $\beta_1 = (\beta_1^l, \beta_1^u)$ providing a lower level of service and s'_2 has the associated service function $\beta_2 = (\beta_2^l, \beta_2^u)$ providing a higher level of service. The service automaton of the second PE has the same structure, except that the service functions associated with its modes are different.

The RTC model of the system consists of an arrival function $\alpha = (\alpha^l, \alpha^u)$ and a service function $\beta = (\beta^l, \beta^u)$ for each PE such that $\alpha^u(\Delta)$ is the maximum of $\alpha_i^u(\Delta)$ and $\alpha^l(\Delta)$ is the minimum of $\alpha_i^l(\Delta)$. β is as defined as in case study 1. Fig. 7 shows the maximum fill-level of the second

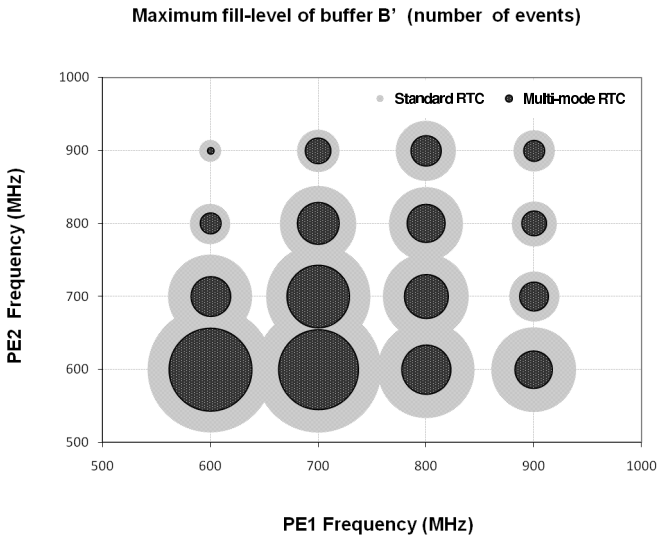

Figure 7: Case study 2: Maximum buffer fill-level analysis.

buffer for the multi-mode RTC and the standard RTC. In the figure, each bubble denotes a fill-level of the buffer. The larger its diameter, the higher the buffer fill-level. Observe that the bubbles corresponding to the multi-mode RTC fall nicely inside the ones corresponding to the standard RTC method. Similarly, the maximum end-to-end delay of the event stream for the multi-mode RTC and the standard RTC are given in Fig. 8. We have not shown the corresponding optimal values since they coincide with the multimode RTC values. Thus our multi-mode version of RTC improves the accuracy of the analysis -in comparison to the standard RTC approach- significantly.

4.3 Efficiency

To evaluate the efficiency of our method, we measured the average running time for computing the maximum buffer fill-levels for the three methods. As might be expected, the standard RTC costs the least computational effort. The multi-mode RTC comes in next with an average of 100 seconds. Not surprisingly, on the average, the exact method is an order of magnitude slower than the multi-mode RTC.

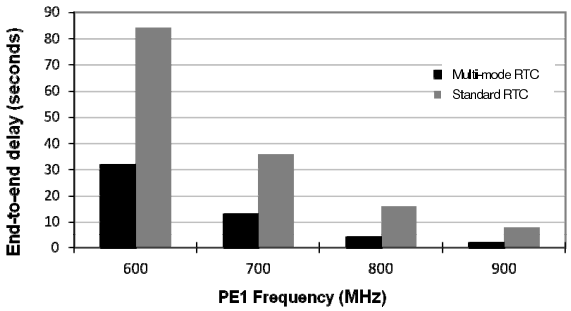

Figure 8: Case study 2: End-to-end delay analysis.

In summary, we have demonstrated through our case studies that while the simpler RTC model is computationally efficient, it does not achieve high accuracy. On the other hand, an exact method that explores the complete search space often suffers from inefficiency. Our multi-mode RTC is able to extend the standard RTC model in terms of expressive power while incurring acceptable additional computational costs. This method enables a more accurate analysis for systems that have state information which cannot be exploited by the standard RTC while being much more efficient (but possibly less accurate) than fine grained state-based models.

5 Concluding Remarks

We have formulated here a state-based extension of RTC called multi-mode RTC. The arrival and service automata that arise in this setting have a syntax very similar to that of hybrid automata. They can describe complex arrival and service patterns by associating an arrival or service function with each mode and using external signals, timing constraints and -in the case of service automata- buffer constraints to perform mode switching. We have sketched how exact analysis can be performed in this setting. We have also provided a more detailed description of a worst-case based approximate analysis method using which basic system properties can be computed accompanying with experimental evidence to validate our analysis techniques.

It will be interesting to explore this model in the setting where multiple data streams are being processed by each processing element. In this case the automata that represent the remaining service as well as scheduling policies will come into play. It will also be interesting to establish methods for tuning the parameters of the arrival and service automata in order to obtain the desired system level properties in terms of buffer capacities, end-to-end delays, etc.

It will be challenging but useful to establish bounds on the degree of approximations (say, for buffer fill-levels) as well as the time complexity of our analysis methods. Finally, arrival and service automata appear to be an expressive and convenient mechanism for describing the interfaces of networks of processing elements and data streams. Using such interfaces, one can begin to study the compatibility of components; i.e. when they can be safely connected together via buffers with pre-specified capacities.

References

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

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

[3] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *RTSS*, 1999.

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

[5] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *RTSS*, 2005.

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

[7] 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.

[8] G. Fohler. Changing operational modes in the context of pre run-time scheduling. *IEICE Transactions on Information and Systems*, E76-D(11):1333–1340, 1993.

[9] T. A. Henzinger. The theory of hybrid automata. In *LICS*, 1996.

[10] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1, 2 and 3 of *Monographs in Theoretical Computer Science*. Springer-Verlag, 1997.

[11] 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 *RTSS*, 2007.

[12] L.T.X. Phan, S. Chakraborty, and P.S. Thiagarajan. A multi-mode real-time calculus. Technical report, Department of Computer Science, National University of Singapore, October 2008. www.comp.nus.edu.sg/˜phanthix/papers/multimodeRTC.pdf.

[13] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26:161–197, 2004.

[14] L. Sha, R. Rajkumar, J. Lehoczsky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):244–264, 1989.

[15] Y. Shin, D. Kim, and K. Choi. Schedulability-driven performance analysis of multiple mode embedded real-time systems. In *Design Automation Conference (DAC)*, 2000.

[16] 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.

[17] 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.

[18] 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.